Automated Service-Oriented Impact Analysis and Recovery Alternative Selection

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

David Schmitz

Tag der Einreichung: 10. Juni 2008
Tag der mündlichen Prüfung: 16. Juli 2008

1. Berichterstatter: **Professor Dr. Heinz-Gerd Hegering**,
Ludwig-Maximilians-Universität München

2. Berichterstatterin: **Professor Dr. Gabrijela Dreo Rodosek**,
Universität der Bundeswehr München

Automated Service-Oriented Impact Analysis and Recovery Alternative Selection

Dissertation

an der

Fakultät für Mathematik, Informatik und Statistik
der
Ludwig-Maximilians-Universität München

vorgelegt von

David Schmitz

Tag der Einreichung: 10. Juni 2008
Tag der mündlichen Prüfung: 16. Juli 2008

1. Berichterstatter: **Professor Dr. Heinz-Gerd Hegering**,
Ludwig-Maximilians-Universität München

2. Berichterstatterin: **Professor Dr. Gabrijela Dreo Rodosek**,
Universität der Bundeswehr München

Acknowledgements

This thesis has been written as a part of my work as a researcher at the Leibniz Supercomputing Center of the Bavarian Academy of Sciences and Humanities. In addition to this it was done in close cooperation with the research group of Prof. Dr. Heinz-Gerd Hegering at the University of Munich (LMU).

First of all, I would like to thank my doctoral advisor Prof. Dr. Heinz-Gerd Hegering for his continued support and helpful advice as well as especially for the numerous valuable discussions throughout the whole development of this thesis. I would also give special thanks to my second advisor, Prof. Dr. Gabi Dreo Rodosek, for giving me advice on the general research direction as well as for many valuable discussions about particular research issues concerning this thesis.

Second, I would like to thank my supervisors Dr. Victor Apostolescu and Dr. Helmut Reiser at the Leibniz Supercomputing Center, who both provided me the opportunity to integrate the work of the thesis into the work of our network monitoring project.

Furthermore, I would also like to thank my colleagues of the Munich Network Management Team at the Leibniz Supercomputing Center, the University of Munich, and the University of Federal Armed Forces in Munich. Especially I want to thank Dr. Andreas Hanemann, and Dr. Martin Sailer, who both conducting related research contributed with many valuable discussions. In general, many times the Munich Network Management Team provided the opportunity for discussing particular research issues of this thesis. In this context, I would like to thank Timo Baur, Latifa Boursas, Dr. Michael Brenner, Dr. Thomas Buchholz, Dr. Vitalian Danciu, Nils Otto vor dem gentschen Felde, Dr. Markus Garschhammer, Matthias Hamm, Iris Hochstatter, Dr. Wolfgang Hommel, Dr. Bernhard Kempter, Ralf König, Silvia Knittl, Annette Kostelezky, Dr. Michael Krause, Feng Liu, Dr. Harald Rölle, Thomas Schaaf, Dr. Michael Schiffers, Dr. Georg Treu, and Mark Yampolskiy.

In addition to that, I would also like to thank Patricia Marcu and Marta Galochino, who both conducted related student work which I have supervised and which has been helpful for the preparation of this thesis.

Last but not least, I would like to thank my mother for her continued support during the preparation of the thesis.

Munich, August 2008

This work was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

Summary

The realization of today's IT services are often very complex and depending on a lot of used resources. In the event of a failure or degradation of one or multiple of such resources there are two critical aspects: the impact analysis, that is the analysis of the impact on the dependent services, as well as the corresponding recovery of these services.

Concerning the recovery, for instance the selection of the resource to be fixed first, the estimation of the necessary effort, and the response time are very important.

In this thesis a framework for the impact analysis of currently occurring or only assumed resource degradations on the provider's services is developed. This comprises impact on the functionality of the services, the quality of service (QoS), the particular customers and service level agreements (SLAs), as well as impact on the provider's business by e.g., SLA violation costs, revenue loss, or further related influences on finances or reputation.

Moreover, the framework includes an analysis and decision support for the selection of appropriate recovery measures for a fast and efficient, partial or complete compensation of the given resource degradations.

At last, the support for tracking actually realized recovery measures is treated in order to allow the consolidation with potentially performed IT changes as well as the notification of affected customers.

Each of the above mentioned parts of the developed framework includes a treatment of the involved workflows, a correspondingly comprehensive and integrated data modeling, as well as the application to concrete IT service scenarios.

Kurzfassung

Heutige IT-Dienste sind oft sehr komplex realisiert und von vielen zugrunde liegenden Ressourcen abhängig. Im Fehlerfall einer oder mehrerer solcher Ressourcen sind die schnelle Analyse von Auswirkungen (impact analysis) auf die damit realisierten Dienste sowie die Wiederherstellung (recovery) der Dienste kritische Faktoren.

Für die Wiederherstellung ist z.B. die Wahl der zuerst zu reparierenden Ressource, die Abschätzung des erforderlichen Aufwands und der Reaktionszeit sehr wichtig.

In dieser Arbeit wird ein Rahmenwerk für die Analyse von Auswirkungen durch aktuell auftretende oder nur angenommene Ressourcenbeeinträchtigungen auf die Dienste eines Anbieters entwickelt. Dies beinhaltet zunächst Auswirkungen auf die Funktionalität der Dienste, die Quality of Service (QoS), die Kunden und Service Level Agreements (SLAs) sowie weitergehend auch Folgen für das Business des Providers in Form von z.B. SLA-Strafzahlungen, entgangenem Einkommen oder sonstigen finanziellen oder den Ruf betreffenden Auswirkungen.

Weiter beinhaltet das Framework eine Analyse und Entscheidungshilfe für die Wahl von geeigneten Wiederherstellungsmaßnahmen zur schnellen und effizienten Kompensation oder sogar vollständigen Aufhebung der gegebenen Ressourcenbeeinträchtigungen.

Abschließend wird ebenfalls die Unterstützung der Verfolgung von den tatsächlich durchgeführten Wiederherstellungsmaßnahmen, zwecks der Konsolidierung von möglichen erfolgten IT Änderungen und der Benachrichtigung von betroffenen Kunden, behandelt.

Jeder der oben genannten Teile des entwickelten Frameworks schließt jeweils die Behandlung der beteiligten Workflows, eine entsprechend umfassende und integrierte Datenmodellierung, sowie die Anwendung auf konkrete IT Service-Szenarien ein.

CONTENTS

Chapter 1

Introduction

Contents

today's business highly depending on IT services

Nowadays, many companies are utilizing IT services to support their business, and so are often depending on them to a large degree. In most cases, IT service provisioning is a complex task and additionally the underlying IT infrastructure itself is rapidly changing - concerning hardware and software - today. Therefore, most companies concentrate onto their core business, whereas *IT services* are outsourced to specialized *IT service providers*.

SLAs agreed upon to ensure proper service quality

Because reliability of these outsourced IT services is critical for the supported business, so-called *service level agreements (SLAs)* are agreed between customers and IT service providers to ensure proper IT service quality. Besides a description of the provided service functionality, these contract include so-called *quality of service (QoS)* parameters and corresponding *service level definitions* based on them, which together describe the agreed and targeted quality and performance of the service in question. If the service levels are not met, specifically defined penalties paid by the provider have to cover resulting consequences for the customer.

management paradigm shift from resource-orientation to service-orientation

Consequently, it is essential for a provider to be able to provide services with guaranteed quality. That is why a paradigm shift in the area of management has been occurring from *device-oriented management* to *service-oriented management* affecting all of the well-known FCAPS (fault, configuration, accounting, performance, security) management functional areas. This requires the adaption to service-orientation for configuration management, which determines the manner how services are provided, as well as performance management, where achieved service performance has to be monitored. Moreover, it also also requires a service-oriented adaption for accounting, security management, and fault management. Concerning especially the latter, today it is not sufficient any more to deal only with resource-oriented errors and failures in the network, end systems, and applications. Instead, also service

failures and service quality degradations and their relationship to the resource-oriented failures and quality degradations have to be taken into account.

1.1 A Framework for Impact and Recovery Analysis

service quality degradation as a critical key factor

In today's service management the timely resolution of service failures and service quality degradations is a critical key factor for the whole business. To guarantee short outage times, it is necessary to identify the actual root causes in underlying infrastructure resources responsible for service degradations as fast as possible. But even if these originating failures or quality degradations in resources have been found, it has still to be decided which actions on which time-scale have to be taken to resolve these infrastructure degradations and thereby to reestablish the service quality. That is why it is essential to find out the actual impact of resource degradations currently taking place, that is to determine which services and which customers are affected, which SLAs are violated, and what is the whole resulting impact on the business. Only based on this impact analysis, appropriate priorities for all currently occurring resource degradations can be assigned and correspondingly appropriate resolution actions can be selected to handle degradations with greater impact first and as fast as possible.

design of a framework for I/R analysis

In this thesis a framework for impact analysis, prioritization, and resolution decision making concerning resource degradations with respect to the actual service and business degradations caused is designed. Here the term *degradation* can denote a complete failure or a quality degradation. For one or multiple given, currently occurring or only assumed resource degradations the impact with respect to the provided services is determined, the determined degradations are prioritized, and appropriate resolution actions are selected. The term *impact and recovery analysis (I/R analysis)* will be used to subsume impact analysis itself, i.e., the determination of impact of resource degradations on the business, and the following decision support for the resolution phase.

The main benefit of this framework will be a decision support for the appropriate and efficient resolution from occurring resource degradations in order to reestablish proper service operations and service quality while minimizing the resulting service and business impact.

different time-ranges of degradation handling

In today's IT fault management it is often differentiated between *incident management* and *problem management*. On the one hand, incident management is responsible for a fast, short-time solution to minimize impact of currently occurring degradations in the near future. This often only consists of a temporary recovery or workaround without finding a long-term solution concerning the degradation, e.g., the restart of a crashed server process without finding

the real cause of the crash. On the other hand, the issue of problem management, in a more long-term manner, is to find and recover from the real cause of a degradation, e.g., to locate and fix a software bug in a server program to avoid it crashes in the future.

Because today both levels of fault management - incident and problem management operating with different level of detail and on different time-scale - are necessary, both should be supported by an impact/recovery analysis framework. If necessary, the two-level approach might be even more generalized into multiple levels, supporting different search depths for the impact/recovery analysis, even starting at different level of detail for given failure information.

reactive vs. proactive handling of degradations

In addition to the area of reactive fault management concerned with currently occurring degradations, such a framework could be applied to only assumed resource degradations in order to proactively find out critical resources, whose degradations would have the greatest impact on the business, and which resolution actions would be necessary. Such application of the framework would be concerned with the area of IT *availability management*.

Concluding, it is strived for a framework being usable in the following management areas:

- incident management
- problem management
- availability management (as a kind of proactive problem management)

The main issues which arise in the context of such a framework are the following.

impact analysis/recovery process: Starting from a workflow to perform the impact/recovery analysis, necessary components have to be identified. A detailed process covering all necessary interactions among these components has to be developed.

information modeling: Modeling of the various kinds of information for the framework is important. Such a modeling at least has to comprise the services and resources they are based on, including a special focus on their dependencies and quality parameters, SLAs, and different kinds of degradations on resource and service level.

techniques: The techniques which shall be applied for the processing of resource, service, and business degradation/impact information, prioritization, and, recovery information have to be investigated. Moreover, it has to be examined whether existing approaches especially from the network and systems management can be adapted to perform these tasks.

instantiation: While the framework has to be designed to be generically applicable to many kinds of services, an instantiation methodology is re-

quired to apply the framework to concrete real-world service provisioning scenarios. This methodology should consist of a step-by-step application process.

1.2 Deficiencies of Today's Impact Analysis and Recovery Planning Approaches

In today's IT management, the existing approaches for impact analysis and recovery planning from resource degradations suffer from severe problems. The performing of impact analysis and recovery planning is often only done manually by applying best-practice techniques: Necessary information and process flow are mostly not accurate modeled or even undocumented, and only known by experts.

best practices applied manually today for I/R analysis

Today's most IT solutions do not offer integrated tool support regarding information and process flow of impact analysis and recovery. Although such tools are emerging recently, they lack a consistent top-down approach with regard to modeling and process support. In general, such tools are not easily and not generically applicable to a particular IT scenario, or they lack a consistent instantiation methodology.

Nevertheless, impact analysis and a corresponding recovery is necessarily performed by many IT administrators today, usually done manually and by using undocumented expert knowledge. To actually carry out impact analysis and recovery an administrator might access different sources of information, e.g., device configurations, performance statistics, application documentation, customer databases, and might also use various tools, e.g., for network checking, debugging purposes, and SLA management.

Recent related work, such as in the areas of SLA management, service provisioning, and service problem management is often only concerned partially with the information and process workflow needed for an appropriate and accurate impact and recovery analysis. A consistent integration into the existing service management and provisioning infrastructure, a generic modeling, and an appropriate workflow support, fitting all required aspects and areas involved, are not yet covered in the existing research. This makes it hard for today's IT providers to actually perform impact/recovery analysis efficiently in terms of costs and effort for each of their services offered.

support today's best practices of I/R analysis by consistent automation and integration

The aim of this thesis is to address all these important deficiencies. Thus, the idea of this thesis is to develop a consistent and top-down oriented framework for impact and recovery analysis (I/R analysis). This framework should be generically applicable to any given service scenario. Furthermore, it should adequately and without great effort integrate into the existing service provisioning and management infrastructure. Most important, it should greatly help the experts to perform an appropriate and accurate impact analysis, and based on this to decide an adequate recovery, both in a faster, more efficient, and more reliable manner than it is done today only by hand. The framework to be developed should assist the IT administrators to perform impact/recovery analysis by providing all necessary information in a consistent way and further by automating required steps as much as possible.

reuse and adaptation of existing concepts and techniques

Impact analysis/recovery is concerned with different areas of management such as SLA management or service modeling and therefore is using different pieces of information from different sources, such as management tools, databases. That is why the basic idea for the development of the framework is to identify existing managements concepts and components (such as tools and databases) involved in today's impact/recovery analysis and to integrate them in a consistent way by providing an appropriate modeling and workflow as well as well-defined interfaces. At first, it has to be analyzed which information, which concepts, which components, and which actions are involved in impact/recovery analysis, as performed today by experts using only best-practices and undocumented experience. Afterwards, appropriate integrated information models, workflows, and interfaces between the identified management components have to be designed.

Particular recent related work for this thesis is shortly outlined in the following: Fig. 1.1 gives an overview of related work in general, while Fig. 1.2 illustrates particular relationships between this thesis and specific related work.

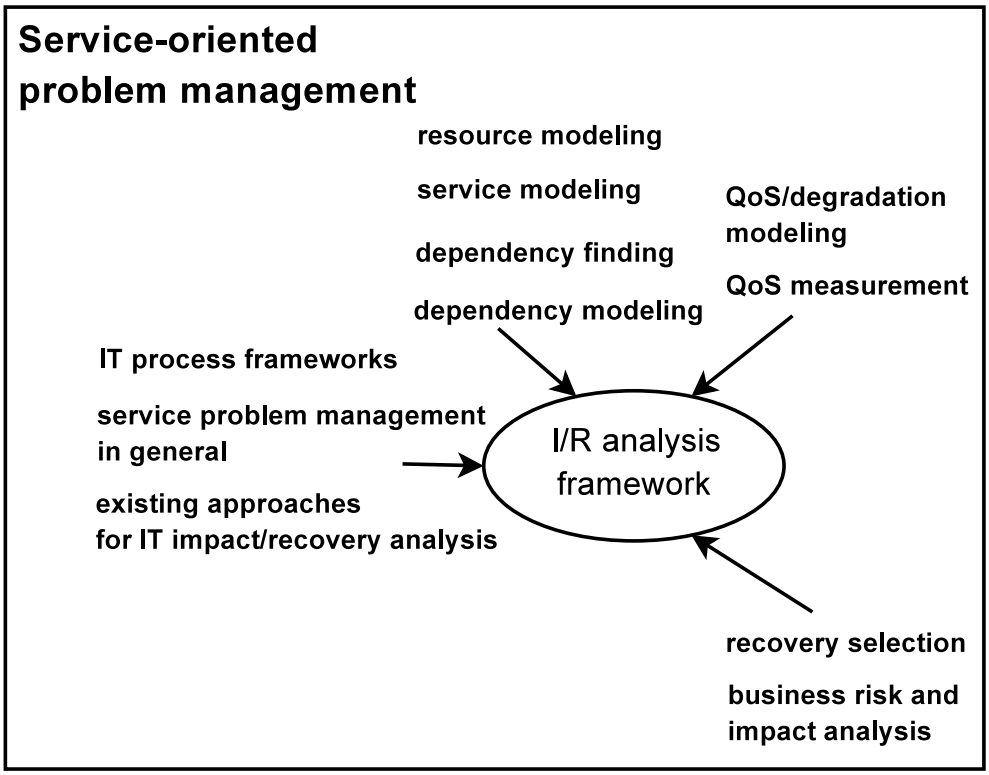

Figure 1.1: Related work

Service MIB approach

The framework will have - as mentioned above - to deal with different areas of management and various sources of information. M. Sailer [Sai07] has developed a so-called *Service MIB* - a generic information model suitable for every aspect and type of information necessary for service provisioning and service management in general. This model provides a generic basis for the integration of framework as part of IT service management within a provider's service management and provisioning environment: It basically provides an in-

tegrated and consistent repository for storing dependencies between services and resources necessary for impact/recovery analysis. Nonetheless, the detail level of the dependencies proposed in [Sai07] is not yet as accurate and granular enough to be used for a detailed impact/recovery analysis. The development of such an appropriately consistent, detailed modeling will be part of this thesis. For the actual implementation this refined modeling can be integrated with the approach of [Sai07], which especially also already provides generic measurement and access interfaces to most pieces of information necessary for impact/recovery analysis in general.

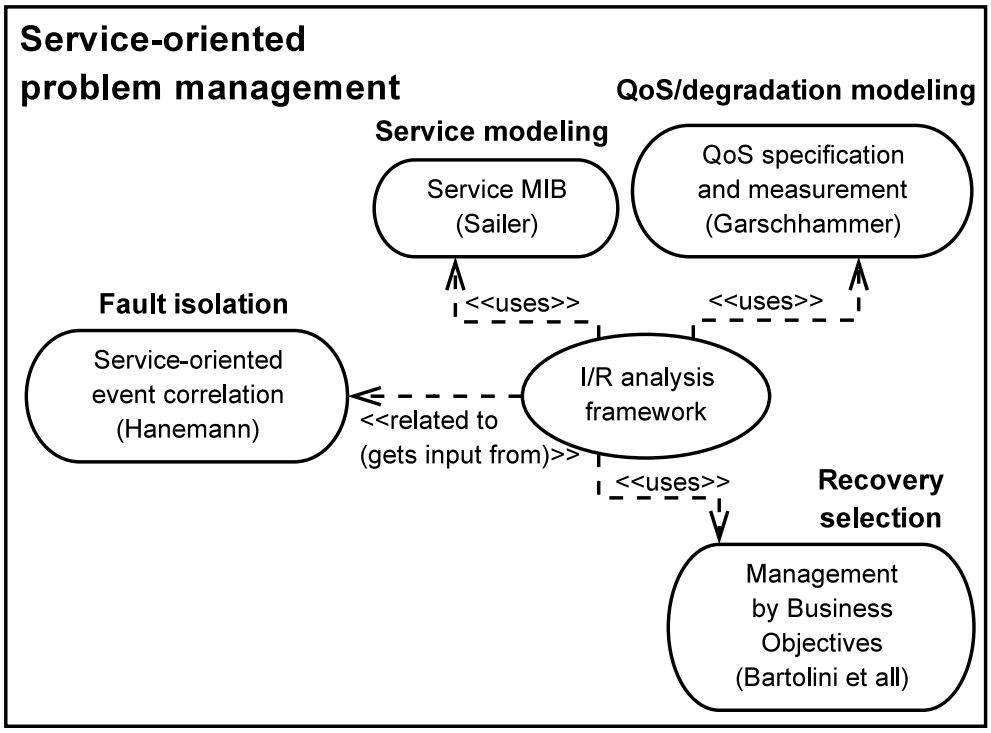

Figure 1.2: Detailed related work

customer-oriented QoS framework

Impact/recovery analysis as defined in this thesis is especially targeting towards business-orientation and customer-orientation, i.e., determining actual service and business impact caused from given resource degradation in accurate and enough granular level of detail, including quality aspects. M. Garschhammer [Gar04] has designed a framework for the definition and actual measurement of Quality of Service (QoS) parameters in a customer-oriented way by monitoring them at the service access point. That is why the approach of [Gar04] seems to be a promising generic input source for customer-oriented quality information needed for an appropriate impact/recovery analysis

Management by Business Objectives approach

The *Management by Business Objectives* group [SB04] has already performed some research concerning the decision of recovery alternatives by utilizing mathematical optimization models. So, generally this work is related to and may provide a possibility for the framework's recovery planning. Nevertheless it is not really concerned with a generic modeling of recovery actions and

recovery plans, a task which therefore has to be approached by the framework itself.

phases of problem management and I/R analysis

IT impact analysis/recovery is a specific part of service problem management: By using known actual resource degradations the impact on the services is determined and the decision for an appropriate recovery alternative is assisted. Problem (and incident) management in general consists of four phases: first *failure location*, i.e., observation of failure symptoms, second *failure isolation*, i.e., the determination of the actual root cause for the observed symptoms, and third *failure diagnosis* (i.e., impact analysis), and fourth *recovery* from the actual root cause. The recovery can divided up in *recovery selection*, actual *performing of the recovery*, and *recovery tracking* for re-consolidating the used modeling with potentially occurred IT changes during recovery and for keeping affected customers informed. The I/R analysis framework in this thesis is only concerned with the selection and the tracking of the recovery, not with the actual performance of the recovery itself. To sum it up, the subject of this thesis, I/R analysis is concerned with the third and partially with the fourth (decision help and tracking) of the phases of problem management. Fig. 1.3 illustrates this relationship of service problem management and I/R analysis.

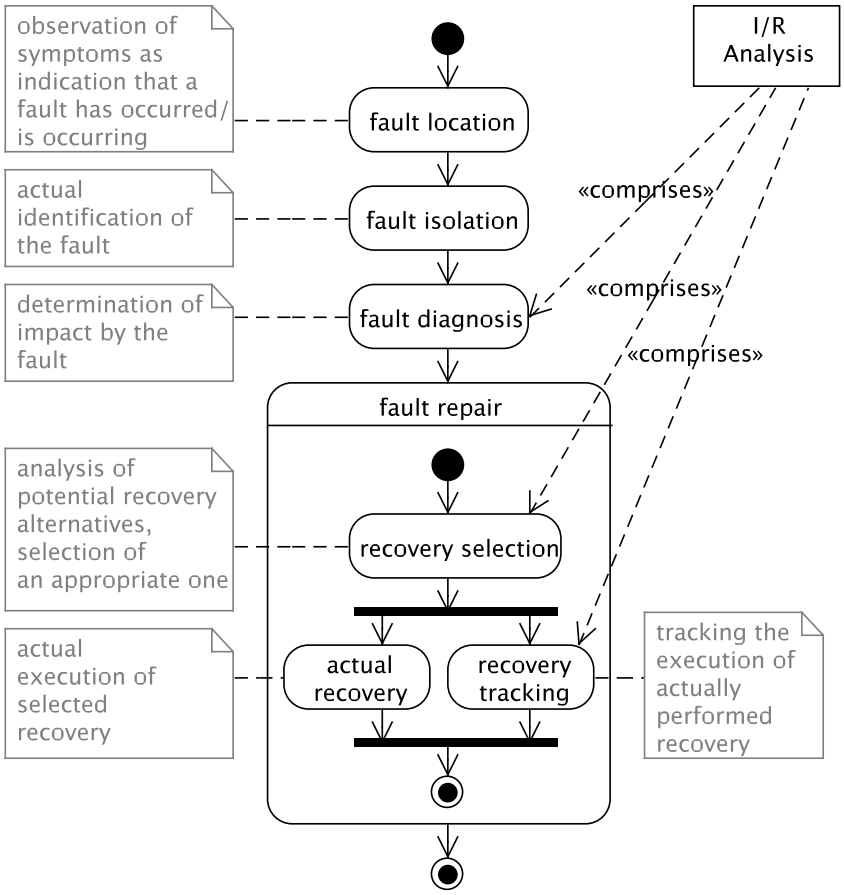

Figure 1.3: Relationship between service problem management in general and I/R analysis

Before performing an impact analysis, one needs to know about the actual failures, i.e., the actual location, and isolation of faults has to be done first. In this work, it is assumed that these current faults/degradations are already known, and are regarded as the essential input.

service-oriented event correlation

A. Hanemann [Han07] has developed a framework for service-oriented event correlation which is mainly concerned with the second one of the above mentioned phases: Current service failure/degradation reports from customers are correlated and combined with failure/degradation information from lower service and resource layers to isolate one or multiple possible root causes of the reported service degradations. The final outcome, that is list of identified potential root causes (degradations on the resource layer) can be used as input for the framework of impact/recovery analysis developed here.

1.3 Thesis Outline

In the following, the structure of the thesis is presented. Fig. 1.4 shows an overview in which dashed arrows denote input/output of the steps performed during the course of the thesis.

requirements analysis

Chapter 2 is concerned with an requirements analysis for an I/R analysis framework. First, important definitions of terms which will be used throughout the thesis are introduced. These definitions are based on the MNM Service Model [GHH+01, GHK+01, GHH+02], because this model basically allows for an universal, generic, and consistent modeling of any given IT service scenario, by providing consistent and generic definitions for service, service management and related terms. Moreover, the Leibniz Supercomputing Center is used as a typical example of a large-scale IT service provider. Its services "Web Hosting Service" and "E-Mail Service" are used as a further motivation for the necessity of the research towards an improved fault management for services and to derive requirements for the framework. At the end of that chapter a requirement catalog for the framework is identified.

related work

In Chapter 3 related work is analyzed and assessed with respect to its reusability or integration in the framework. The related work taken into account comprises IT process management frameworks especially ITIL problem, incident, and availability management, service and resource modeling approaches, approaches for dependency modeling and discovery, QoR/QoS modeling and measurement, financial business impact analysis and risk analysis, service usage prediction, SLA modeling, and existing approaches for IT impact/recovery analysis.

framework design

The framework for service-oriented impact and recovery analysis meeting the requirements from Chapter 2 is designed and discussed in Chapter 4. The framework is generic in that it aims to be applicable for any IT service scenario. Chapter 4 covers this framework in general, while its particular instantiation to a concrete service scenario will be addressed later on in Chapter 5.

generic framework

The generic development in Chapter 4 is performed in various steps. First, the *basic framework* is introduced which covers the whole impact and recovery analysis on a conceptual level, i.e., without going yet too much into details concerning the necessary data structures: Three refinement substeps of a workflow for I/R analysis, namely the abstract workflow, the refined abstract workflow, and the realized workflow, each covering the whole impact/recovery analysis, are treated. The second substep comprises already a rough abstract modeling of necessary data structures, while the third one also comprises an abstract component architecture for the realization of impact/recovery analysis. Based on the basic framework, three extension frameworks, namely *impact analysis framework*, *recovery analysis framework*, and *recovery tracking framework*, are developed. Each extension framework is concerned with a separate part of impact/recovery analysis, and for its respective

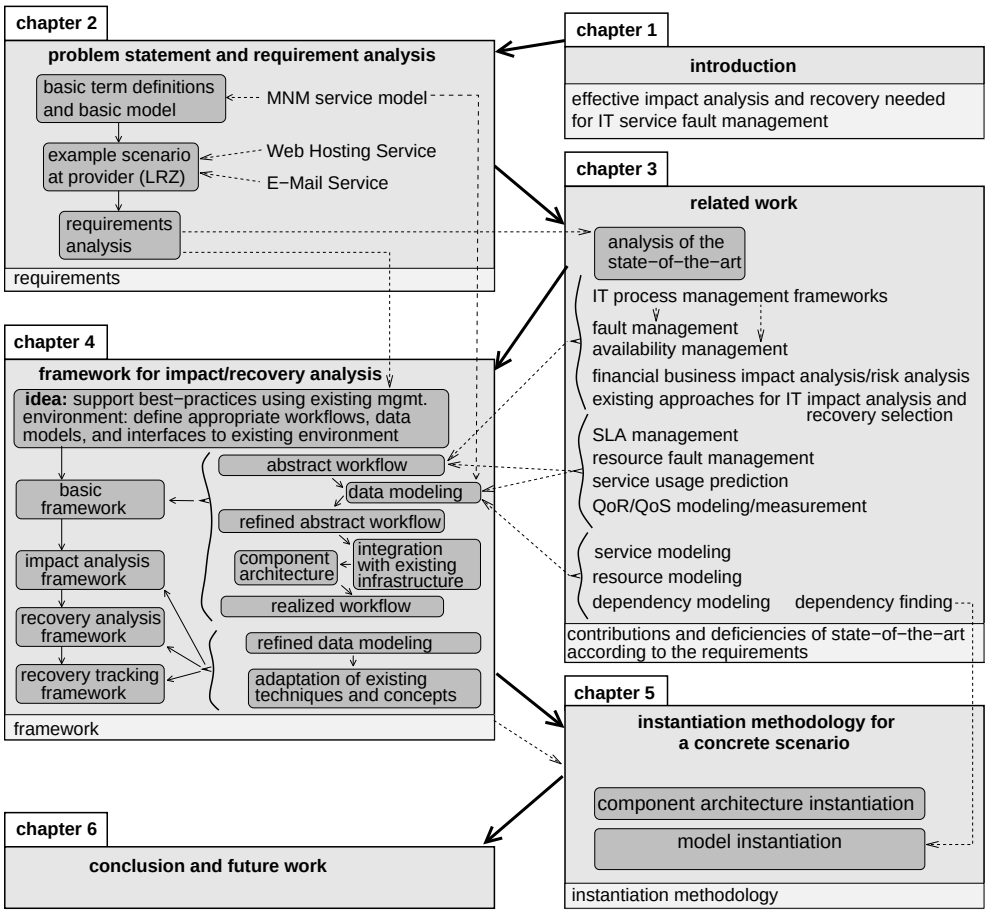

Figure 1.4: Structure of the thesis

part it includes a detailed and refined modeling of the necessary data structures, their particular use, and a potential implementation based on this.

instantiation concept

The instantiation of the generic framework developed in Chapter 4 for a real-world scenario of an IT service provider is discussed in Chapter 5: Basically, instantiation of the framework is divided into component architecture instantiation and model instantiation. Especially, a top-down oriented instantiation methodology, being based on business policies and SLA definitions, is presented. This instantiation methodology is exemplified with the example services offered by the Leibniz Supercomputing Center, which already have been presented in Chapter 2.

conclusion

The last chapter concludes this thesis and presents extensions and remaining open issues which are subject to future work.

Chapter 2

Requirements Analysis

Contents

In this chapter general requirements to be posed on a framework for I/R analysis are identified.

At first, in Sect. 2.1 some important terminology is defined and in Sect. 2.2 the MNM Service Model, which can be used for basic modeling of any given service, is introduced. In Sect. 2.3 an example scenario, which will serve for illustration throughout the whole thesis, is presented using the MNM service model. At last, the requirements are identified in Sect. 2.4.

2.1 Definition of Important Terms

This section introduces important terminology which is used throughout the whole course of the thesis.

Provider: A *provider* offers services to customers. The provider himself can act as a customer in case of having ordered subservices offered by other providers.

Customer: A *customer* subscribes to a service. He grants the possibility to use this to a set of *users*, often being also members of the customer's organization.

Service: In contrast to other definitions where a service is limited to a specific domain or technology, a service is defined here in a generic way. It is specified as a set of *functionalities* that are offered by a *service provider* to a customer at a *customer provider interface* with a certain *quality of service (QoS)*. The customer may allow a set of users to access the service at the *user provider interface*. This notion provides a common understanding between customer and provider, functionality and quality issues of the service provisioning are defined by *service level agreements (SLAs)* in a customer-oriented manner. The service provisioning is realized by using the service's own *resources* which are always provider-internal and other services called *subservices* which might be either provided by the provider himself (*provider-internal subservices*) or by a *subprovider* (*provider-external subservices*).

Service functionalities: The complete functionality of a service consists of *service usage functionality* as well as *service management functionality*. The service usage functionality relates to the normal purpose of the service and is accessed by users (e.g., sending an e-mail with an e-mail service), whereas the service management functionality is concerned with contracting, controlling, and customizing the service between provider and customer (e.g., adding an e-mail user account). Both types of functionality can be divided further into single functionalities to distinguish different use cases of interactions between user/customer and provider (e.g., sending an e-mail in contrast to accessing an e-mail folder).

Service instance: The term *service* here shall refer to a service in general, which may be operated by a provider for several customers, whereas the term *service instance* shall refer to a specific instance of the service provided for a specific customer. Each service instance of a service has its own service level agreement which may contain individual QoS parameters and individual values ranges for these QoS parameters as well as other global parameters of the service instance.

Service level agreement: A *service level agreement (SLA)* is a contract between customer and provider about one or multiple offered services. It

comprises a legal part, concerned e.g., with payments from the customer, and with *SLA violation penalty costs* from the provider for not meeting specific *service levels*, as well as a part concerned with the provisioning and usage of the service itself. This part completely describes the services, comprising e.g., service functionalities (including management functionalities), appropriate service access points and CSM access points, and customer-oriented QoS. Moreover, for each QoS parameter agreed thresholds for specific time intervals, i.e., respective service levels, are defined by appropriate constraints. The provider guarantees for meeting these constraints. Otherwise, the agreed SLA violation penalties have to be paid to the customer.

Subservice: A subservice is a service that is used by other services. This service can be offered to customers or can be used provider-internally. By using subservices offered by other providers it is possible to form provider hierarchies.

Resource: A resource is used by services for provisioning and operating these services. A service is regarded as an abstraction over its underlying resources (and its used subservices). That is why a failure of a service has not to be located in the service itself, but at least in one of the resources which are used for its realization. A resource can e.g., be a network link, an end system, main memory, a hard disk drive, or an application process. Moreover, including a more high-level perspective, the term resource covers also aspects like staff, expert knowledge, and IT processes used for the provisioning of a service.

Service access point: A user accesses the service usage functionalities, to which its customer has subscribed, at the service access point.

Customer Service Management access point: The *Customer Service Management access point (CSM access point)* is the interface to the service management functionalities between the customer and the provider. It allows the exchange of management information, as well as access to certain agreed-upon management functions, e.g., ordering of new services, access to service performance reports, or the exchange of fault management information. Particularly, the CSM access point of a used subservice can be accessed to perform management functions of a dependent service.

Degradation: *Degradation* is used as common term for problems or events occurring which might have an impact on the proper service provisioning. A degradation can be on either layer, on the resource layer (*resource degradation*) or the service layer (*service degradation*). On the service layer it can further be differentiated between a service in general and a specific service instance (i.e., customer).

A degradation on either layer can be a complete failure or only a gradual deterioration, e.g., a QoS or performance degradation.

(Pre-recovery) impact analysis: *Pre-recovery impact analysis* or *impact analysis* (tersely, if nothing else is specified) subsumes all activities to determine the existing or threatening *impact* of one ore multiple resource degradations on the business over time (e.g., described as a function of time) without considering any recovery or repair measures. It comprises the analysis of impact of resource degradations on services in general, on specific service functionalities, on service instances (i.e., customers), or on specific functionalities of service instances. Furthermore, it includes the determination of impact on QoS parameters (provider-oriented or customer-oriented) of these service or service functionalities.

Additionally included is - based on the activities mentioned above - the derivation of the development of QoS violations and related SLA violation costs over time (financial impact over time, directly derived from SLA contract). These formal SLA violation costs can be combined and extended with further financial business impact information, e.g., current and expected, future service usage information. Examples of further financial and reputational aspects to be included are revenue loss, customer satisfaction, and the public image. The latter two examples are concerned with reputational factors which relate to financial impact in the near or far future, e.g., current customers canceling the service contract in the near future because of dissatisfaction or the lack of new customers because of a bad image.

Consequently, the output of the (pre-recovery) impact analysis is the development of the financial/reputational impact - as a function of time or duration - of one or multiple given resource degradations over time, with no consideration about mitigating recovery measures.

Recovery analysis: *Recovery analysis* follows (pre-recovery) impact analysis. After performing the impact analysis and thereby deriving the financial/reputational impact of given resource degradations over time without considering any recovery, the recovery analysis considers and evaluates possible alternatives of recovery plans in order to find an optimal or at least an approximately optimal one. Here, repair costs for different choices of recovery plans are considered. This includes decisions concerning priority and order for handling the degradations, and for each single degradation the time-range, the specific effort, and specific measures to be taken. The findings are evaluated to find the most appropriate and efficient approach to handle the given resource degradations and to be of valuable help for the recovery decision. Included is the determination or estimation of the *reduced impact* which is the impact being left after performing the selected recovery alternative (including factors such as the costs for the recovery) in comparison to the pre-recovery impact without any recovery. All these activities following impact analysis, which are in fact an extension of the impact analysis, because the estimated reduced impact being left after the execution of the selected recovery alternative is determined, are comprised in the term *recovery*

analysis tersely. But alternatively, also the more explicit term *analysis of impact with recovery* could be used instead.

Impact and recovery analysis (I/R analysis): The term *impact and recovery analysis* or tersely *I/R analysis (I/RA)* denotes the consecutive performing of impact and recovery analysis. The essential input is one or several given resource degradations. The essential output is one or multiple recovery plans, comprising scheduling and order of degradation measures, corresponding time-ranges, specific effort to use, detailed, specific measures for handling each of the given resource degradations in a fast and efficient way. Furthermore, each recovery plan has attached information about the estimated reduced impact, in comparison to the pre-recovery impact, resulting after this recovery plan will be realized.

In this thesis a framework for I/R analysis is developed.

2.2 MNM Service Model

consistent, generic definitions of terms

The MNM Service Model [GHH+01, GHK+01, GHH+02] is a generic model for IT service modeling designed to be applicable to any given service scenario. This is basically achieved by a providing generic and consistent definitions for all terms involved in the area of service management: A *service* here is basically defined as a set of interactions between different roles: A distinction is made between *customer side* and *provider side*. The customer side contains the basic roles *customer* and *user*, while the provider side contains the role *provider*. The provider realizes the service and makes it available for access to the customer side. The service as a whole is divided into usage which is accessed by the role user and management which is accessed by the role customer.

usage and management

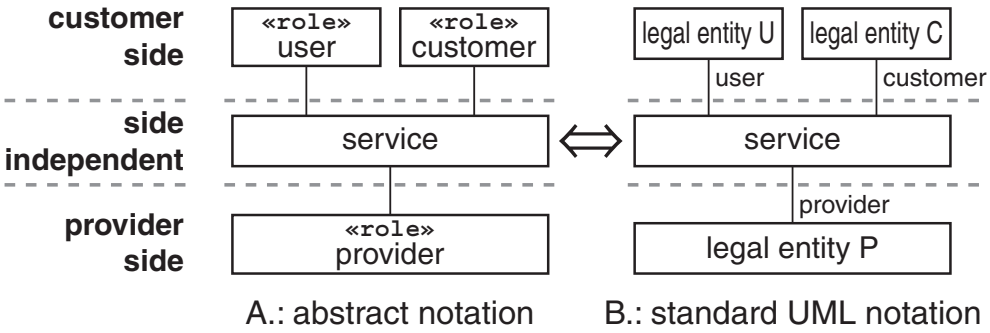

Figure 2.1: MNM Basic Model

overview of services

For each modeled service, the MNM Service Model defines a *Basic Model* to provide an overview of the *service* and its used *subservice*s as well as all participating roles. The Basic Model shows for the service and all subservices which entities (i.e., organizations or individuals) are acting in different roles concerned with each service. It can be defined in abstract notation containing only roles concerning each service, or in object-like notation where the roles are assigned to entities which act in these roles (see Fig. 2.1).

Apart from the Basic Model the MNM model comprises two main views. The *Service View* (see Fig. 2.2) gives a common perspective of the service agreed between the customer and the provider. The details of the service realization, that is provider-internal aspects, are not part of this view. For these details and their relationships another perspective, the *Realization View*, is defined (see Fig. 2.3).

common perspective between customer and provider

The Service View contains the service for which the functionality is defined for usage (*usage functionality*) as well as for management (*management functionality*). There are specifications for the *service access point* as well as the *CSM access point*, where user and customer can access the usage and management functionality, respectively. The service view also contains a list of *QoS parameters* and agreed value ranges for them which have to be met by the service. The *usage functionality* is realized by the *service implementation*

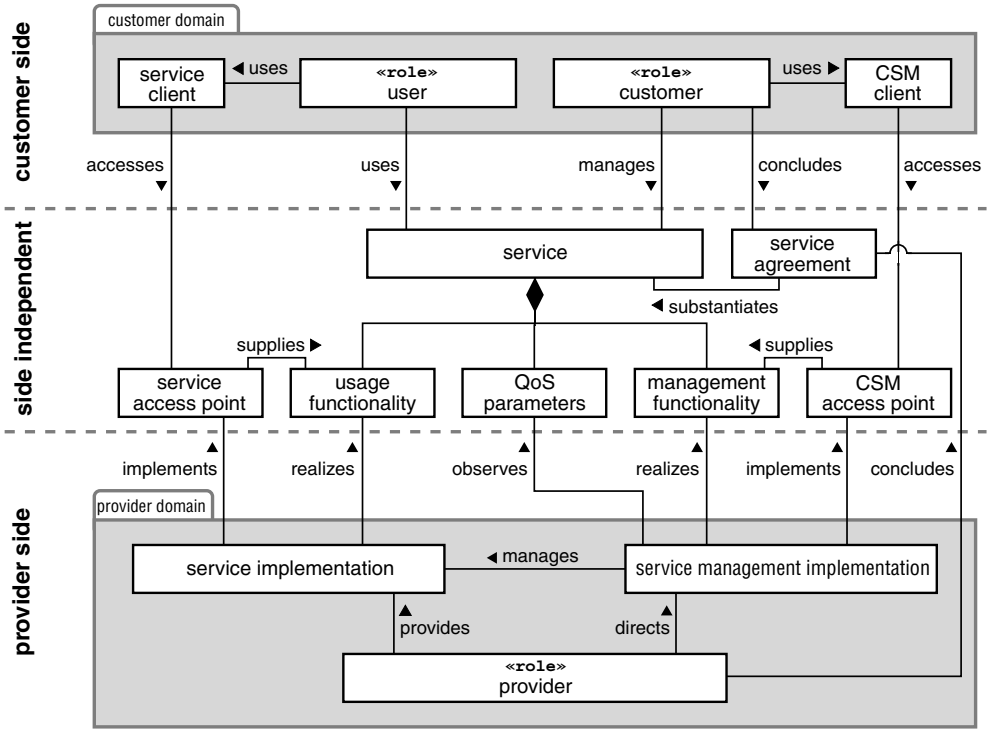

Figure 2.2: MNM Service View

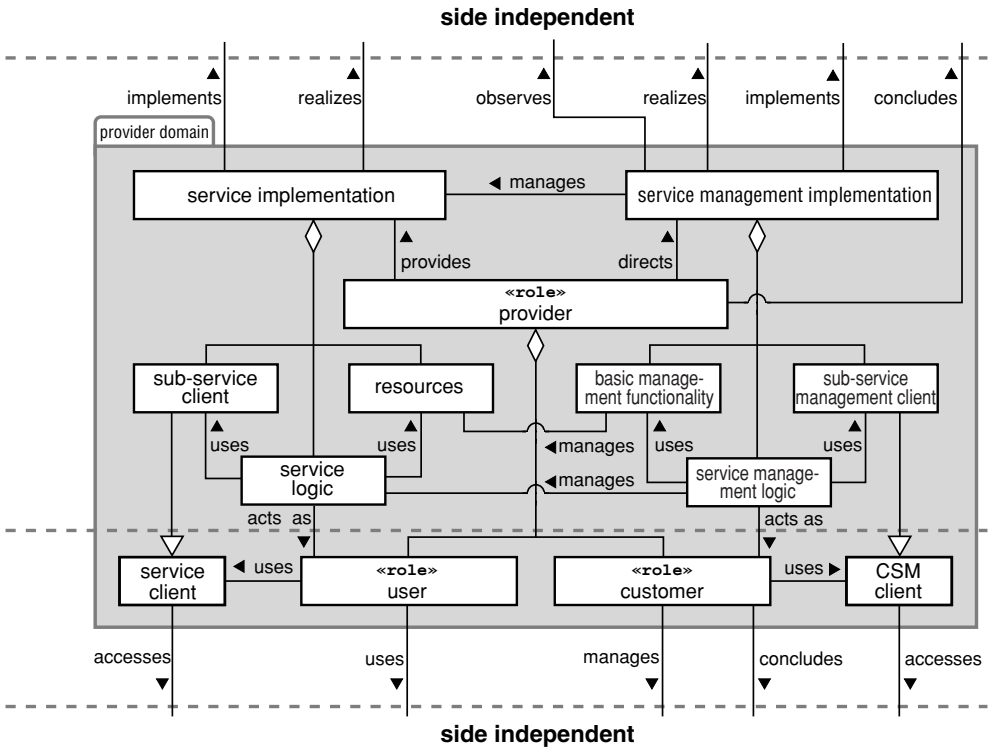

Figure 2.3: MNM Realization View

and the *management functionality* is realized by the *service management implementation*. Additionally, the service management implementation assures the adherence to the specified QoS parameter ranges.

details of the service realization

In the Realization View the service implementation and the service management implementation are specified in detail. Both are realized by using provider-internal components as well as subservices: For service implementation a *service logic* uses internal *resources* (devices, knowledge, staff) and external *subservices* in order to provide the service. Analogously, service management implementation includes a *service management logic* controlling *basic management functionalities* [HAN99] and the management of the external *subservices*. Subservices being external to the modeled service can be provided either by another provider (*provider-external subservices*) or provided by the provider itself (*provider-internal subservices*).

The MNM Service Model can also be used for a modeling of the used subservices, i.e., the model can be applied recursively for provider hierarchies.

2.3 Service Example Scenario at the Leibniz Supercomputing Center

In the following an example scenario comprising two different services is presented. It will serve as an illustrating example in the remainder of the whole thesis.

two typical example services

The scenario is based on the implementation of the e-mail service and the web hosting service at the *Leibniz Supercomputing Center (LRZ)* in Munich. Along with offering high performance computing facilities for Bavaria and Germany, the LRZ is also the common computing center for the Munich Universities. It operates the *Munich Scientific Network (MWN)*, which links universities and other research institutions to the global Internet. Moreover, in this network, which currently comprises more than 60,000 computers, the LRZ also provides high-level IT services such as the mentioned e-mail and web hosting services which are introduced in the following sections.

These two services have been chosen for two reasons: On the one hand, their internal structure is complex enough to illustrate different aspects of I/R analysis. On the other hand, they partially share common subservices as e.g., IP connectivity service, or storage service, which means that degradations in these common subservices might relate to the mail service as well as the web hosting service simultaneously.

In the following, both services will be described by using terms of the MNM Service Model (see Sect. 2.2) in Sect. 2.3.1 and Sect. 2.3.2, respectively. Afterwards, an example run of I/R analysis for this scenario is introduced in Sect. 2.3.4.

2.3.1 E-Mail Service

Here, the first example service, which is used for the requirements analysis as well as in the whole remainder of the thesis, is described. For this description, terms of the MNM Service Model are used, because these terms are generic, and not specific to a particular technology or type of scenario, so that later generalization is facilitated. First, the service scenario is shortly described in general. Second, functionalities and some of their possible degradations are given. Third, resources used for realization, some of their possible degradations, as well as the entailed degradations of functionalities are presented. In the following, all general dependencies of functionalities on resources are explained as a general overview. Finally, QoS parameters, SLA constraints, and SLA penalties for the service are also specified to allow for a first mapping from degradations of functionalities to business impact.

introduction of e-mail service scenario

The LRZ operates and provides the *e-mail service* (*mail service* in short). Customers of this service are the Munich universities, i.e., mainly the *TUM (Technical University of Munich)* and the *LMU (Ludwig-Maximilians Univer-*

sity), but additionally also various, smaller research institutions in Munich are customers. The users can be differentiated into students of the universities and university/research institution staff. Furthermore, the LRZ or more specifically its departments are customers of the service, too. In fact, the service is run by the LRZ user service department and the other departments of the LRZ access the service as additional users and customers.

In order to use the scenario of the service for illustrating examples of I/R analysis, it is described in more detail, especially concerning possible resource degradations and their impact on the service, on its customers, on its users, as well as on the corresponding SLAs (e.g., in form of SLA violations).

In the first place, impact of resource degradations on services here means degradations of functionalities of services which depend on the degraded resources, for all or a subset of the customers and corresponding users of these services.

different aspects of degradations of resources and functionalities

In general, it can be said that every degradation of a resource used for provisioning or management of the e-mail service might have an impact on all or some of the functionalities of the service. However, different resources can have different types of impact regarding aspects as the subset of affected functionalities, actually affected users and customers, the subset of affected QoS parameters: E.g., a resource degradation might affect all functionalities of a service, or it might only affect some functionality (e.g. only mail sending, not accessing of mail boxes). Furthermore, a resource degradation might affect the availability of a service functionality for all the corresponding users and customers of this functionality, or it might only affect the availability of a service functionality for a subset of its users (e.g. mailbox access unavailable only for students, not for staff of a university, because the respective mail boxes might be hosted on different mail incoming servers or might stay on different parts of an underlying filesystem structure). Finally, different resource degradation might affect different QoS parameters of a service functionality, e.g., high utilization of a network link might increase the e-mail sending delay, while a complete outage of the network link might cause the e-mail sending functionality to be completely unavailable.

Thus, regarding a service, it seems necessary to differentiate between different types of resources and their possible degradations depending on the different types of degradations of the service they entail, including aspects as the actual affected functionalities, actual affected QoS parameters, and subset of actual affected users and customers.

introduction of functionalities of the mail service

In the following, functionalities of the e-mail service are described in order to give a general overview and a basic understanding of the possible degradations of the service and its functionalities. The functionalities have been identified by a detailed analysis of the service scenario. In this way, the result of this analysis represents only one particular example for a subdivision of the particular mail service into functionalities, nevertheless one which is suitable for using it with respect to degradations and in general to I/R analysis. Moreover, it is explicitly mentioned that for this e-mail service scenario

issues which are totally specific to mail clients and go beyond the scope of the service as provided by the provider LRZ, such as security by client-side encryption, were not included in the analysis. Table 2.1 gives an overview of all functionalities, which are explained in the following. Afterwards, the resources used to realize the functionalities of the service, their possible degradations, as well as the degradations of functionalities they entail are introduced.

usage ($f_{\text{mail/use}}$)
sending mail ($f_{\text{mail/use/send}}$)
sending mail within inner domain ($f_{\text{mail/use/send/intra}}$)
sending mail to outer domain ($f_{\text{mail/use/send/extra}}$)
receiving mail ($f_{\text{mail/use/recv}}$)
receiving mail from inner domain ($f_{\text{mail/use/recv/intra}}$)
receiving mail from outer domain ($f_{\text{mail/use/recv/extra}}$)
accessing mail box ($f_{\text{mail/use/mbox\_access}}$)
customizing mail account ($f_{\text{mail/use/customize}}$)
web mail access ($f_{\text{mail/use/webmail}}$)
management ($f_{\text{mail/mgmt}}$)
inquiry and order management ($f_{\text{mail/mgmt/inq\_order}}$)
configuration management ($f_{\text{mail/mgmt/conf}}$)
problem and incident management ($f_{\text{mail/mgmt/prob\_inci}}$)
quality and security management ($f_{\text{mail/mgmt/qual\_sec}}$)
accounting management ($f_{\text{mail/mgmt/acc}}$)
change management ($f_{\text{mail/mgmt/change}}$)

Table 2.1: Overview of the functionalities of the e-mail service

As for any service, functionality of the e-mail service can generally be divided into the general classes *usage functionality* and *management functionality* (compare also Fig. 2.2 and Fig. 2.3). Here, the usage functionality can be further refined into the following functionalities: sending mail, accessing mail box, customizing mail account and mail box, and web mail access.

In order to be able to easily reference each specific functionality in the following, for each functionality an unique identifier (an f with a unique index) will be defined, often specified in parentheses after the first mentioning of the full name of the functionality. Using this notation, overall *usage functionality* of the mail service is denoted by $f_{\text{mail/use}}$, whereas *management functionality* is denoted by $f_{\text{mail/mgmt}}$.

Sending mail ($f_{\text{mail/use/send}}$) comprises sending e-mails via SMTP (Simple Mail Transfer Protocol) with or without authentication. The sending with authentication is necessary only if a sending user is accessing the service from outside the IP network of MWN in order to authenticate and authorize him. Sending mail with authentication is depending on an authentication service, while sending without authentication is not.

Depending on the destination mail domain, $f_{\text{mail/use/send}}$ can be further re-

fined into sending mail within the MWN mail domain ($f_{\text{mail/use/send/intra}}$), i.e., sending mail to another mail account handled by the mail service, or to the global Internet ($f_{\text{mail/use/send/extra}}$), i.e., sending mail to a mail account handled by a foreign mail server. It is useful to differentiate between both cases, because, $f_{\text{mail/use/send/intra}}$ only relies on network resources connecting provider-internal resources, while $f_{\text{mail/use/send/extra}}$ additionally has to use the up-link gateway to the global Internet. That is why degradations of the gateway to the global Internet are not affecting $f_{\text{mail/use/send/intra}}$, while $f_{\text{mail/use/send/extra}}$ might be affected by such degradations.

For *receiving mail* functionality ($f_{\text{mail/use/recv}}$) a similar refinement is possible as for $f_{\text{mail/use/send}}$, for the same reason: Mails received for a user of the mail service can be originated from another mail account handled by the mail service ($f_{\text{mail/use/recv/intra}}$), or from a mail account handled by a foreign mail server ($f_{\text{mail/use/recv/extra}}$).

Accessing mail box functionality ($f_{\text{mail/use/mbox\_access}}$) includes interactions such as checking for new mail, listing mail box content, accessing full or partial mail box content, via protocols such as POP (Post Office Protocol) or IMAP (Internet Message Access Protocol).

Functionality for *customizing mail account and mail box* ($f_{\text{mail/use/customize}}$) comprises all interactions of a user for setting parameters concerning aspects of his mail account or specifically his mail box. Examples of such user-controllable configuration parameters are mail sending priority, mail account passwords, spam auto detection/handling parameters, holiday auto reply settings, the mail forwarding configuration. As appropriate, this functionality could even be further refined.

In addition to accessing the before explained functionalities $f_{\text{mail/use/send}}$, $f_{\text{mail/use/recv}}$, $f_{\text{mail/use/mbox\_access}}$ directly (via SMTP and POP/IMAP), the LRZ allows to access them indirectly via a web mail interface, which is regarded as a further functionality ($f_{\text{mail/use/webmail}}$).

As a first step, management functionality of the mail service can be divided into different areas of management. Different subdivisions are possible, e.g., in general the FCAPS (fault, configuration, accounting, performance, security) classification, or being more service-oriented the particular classification developed for the CSM approach (Customer Service Management) which in a customer-oriented manner identified all classes of management interactions between customer and provider (compare Table 3.1 in Sect. 3.3.1): inquiry and order management ($f_{\text{mail/mgmt/inq\_order}}$), configuration management ($f_{\text{mail/mgmt/conf}}$), problem and incident management ($f_{\text{mail/mgmt/prob\_inci}}$), quality and security management ($f_{\text{mail/mgmt/qual\_sec}}$), accounting management ($f_{\text{mail/mgmt/acc}}$), and change management ($f_{\text{mail/mgmt/change}}$). In turn, the management functionality of a specific management area can be further refined, according to the specific scenario.

degradations of functionalities

The different functionalities described so far can be degraded in different ways. First of all, a functionality can become completely unavailable for all

its users and customers. Furthermore, depending on a given functionality, there also can be partial degradations of that functionality, i.e., degradations which do not entail complete unavailability of the functionality, but affect the functionality partially in a manner specific to that functionality: For example, sending mail functionality $f_{\mathrm{mail/use/send}}$ can be degraded by a large mail sending delay, i.e., the delay occurring during the delivery of an e-mail dispatched by a user to a destination mail domain. Another degradation of mail sending functionality might be a very long duration for mail dispatching, i.e., the duration for a user to dispatch an e-mail to the mail server. Moreover, another partial degradation of mail sending functionality could take place, if some but not all of the e-mails dispatched by users are silently lost instead of being delivered to the designated receiver.

Similar to the sending functionality, the mail receiving functionality $f_{\mathrm{mail/use/recv}}$ can be degraded by a high mail receiving delay, i.e., sending mails to a user of the mail service takes a relatively long time before the user can access the received e-mail in his mailbox (via the mail box accessing functionality). The mail box accessing functionality $f_{\mathrm{mail/use/mbox\_access}}$ could suffer from a low available bandwidth for the mail box access resulting in mail box access transactions taking a relatively long time.

degradations of management functionalities

Also the various management functionalities of the e-mail service can be degraded in different ways: For instance, configuration management ($f_{\mathrm{mail/mgmt/conf}}$), used e.g., for the update (adding, deleting, or changing) of mail users, can become either completely unavailable or its update request delay can increase to a high value making it at least difficult for the customer to update user configuration. Beyond this, the mail service itself is often used as a subservice for the management of other services, e.g., for the order and configuration management of the web hosting service presented in the next section. There, the mail service, especially its functionality $f_{\mathrm{mail/use/send/intra}}$, is used as one alternative for issuing change request. Consequently, if $f_{\mathrm{mail/use/send/intra}}$ is somehow degraded, at least this specific interface for ordering new services (new web sites to host) or requesting changes (to existing web sites) is degraded, too.

There are possibly further types of degradations for the functionalities of the mail service, but so far the ones mentioned above already give a general idea of how different functionalities can each be degraded in various specific ways.

The task of I/R analysis is to determine which degradations of functionalities are in which specific manner caused by the given degradations of resources. Therefore, in the following, the resources used for the realization and the management of the mail service are introduced. Next, the possible degradations of these resources as well as the consequences for functionalities are considered.

introduction of resources of the mail service

Similarly, as for the functionalities described above, for each introduced resource a unique identifier (an r with a unique index) is given in parentheses to allow for easy differentiation in the following. Fig. 2.4 basically gives an overview of these resource dependencies of the mail service of the LRZ, at

least as for its usage functionality $f_{\text{mail/use}}$. There $f_{\text{mail/use}}$ is illustrated by an ellipsis, and the resources its depends on are illustrated by specific resource icons. Dependencies are illustrated by dashed lines from the resource icons to the ellipsis illustrating $f_{\text{mail/use}}$.

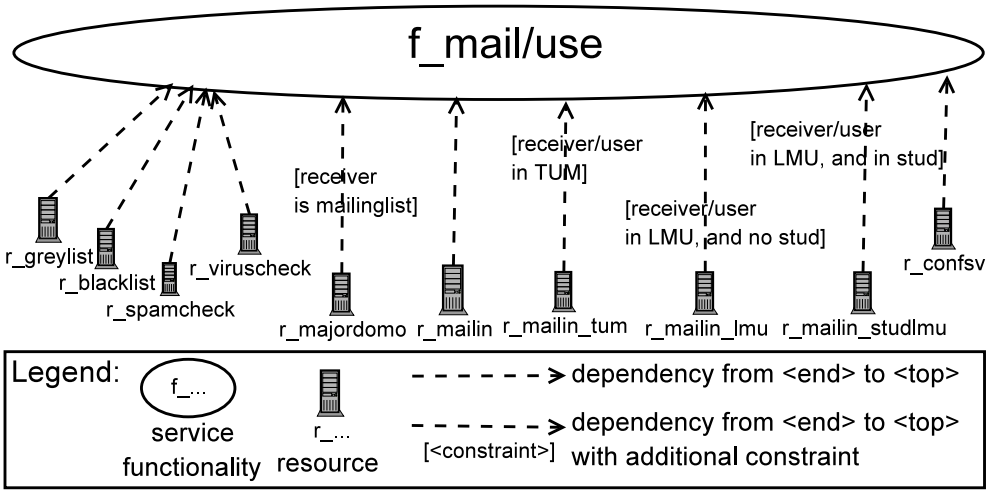

Figure 2.4: Dependencies from resources to the usage functionality of the e-mail service

For realization of the usage functionality of the e-mail service, which is essentially based on the protocols SMTP and POP/IMAP, the following resources are used: Different mail incoming servers for different user groups (r_{mailin}, $r_{\text{mailin\_tum}}$, $r_{\text{mailin\_lmu}}$, $r_{\text{mailin\_studlmu}}$) are used, which actually receive mails sent to users and which allow users to access the mails from their mail boxes. Actually, r_{mailin} is involved in the receiving of any mail, i.e., for any user, because each mail sent to some user of the service is first received by this mail incoming server. I.e., the mail receiving functionality for any user is dependent on this resource. But in case of users from TUM or LMU, e-mails received are relayed to the respective mail incoming servers, i.e., $r_{\text{mailin\_tum}}$ for all users of TUM, $r_{\text{mailin\_studlmu}}$ for student users of LMU, and $r_{\text{mailin\_lmu}}$ for non-student users of LMU. Therefore, for the users of TUM and LMU, the respective specific incoming servers are necessary for the mail receiving functionality, too. For users not pertaining to TUM and LMU, i.e., LRZ users or staff of other research institutions, the mails received by r_{mailin} are not sent further and are accessible by the users via r_{mailin} directly. That is why for such users, the mailbox access functionality is also depending on r_{mailin}. In contrast, for users of TUM and LMU, their mail boxes can be reached via the respective specific incoming mail server, to which r_{mailin} relays their mails. So, the mailbox access functionality for users of TUM depends on $r_{\text{mailin\_tum}}$, for non-student users of LMU on $r_{\text{mailin\_lmu}}$, and for student users of LMU on $r_{\text{mailin\_studlmu}}$.

Two mail relay servers ($r_{\text{mailrelay1}}$ and $r_{\text{mailrelay2}}$), which are load-balanced by using specific DNS records (Round-Robin DNS), are used for mail sending

purposes. Being combined by the load-balancing, they are uniformly accessible as a single virtual resource (r_{mailout}).

Additionally, various servers are used for checking of incoming mails: a spam checking server ($r_{\text{spamcheck}}$), a virus checking server ($r_{\text{viruscheck}}$), a blacklist filtering server ($r_{\text{blacklist}}$), and a server for performing mail graylisting (r_{greylist}). In order to store and to allow for customization of configuration data concerning user accounts and mail boxes, a dedicated configuration server is used ($r_{\text{conf\_sv}}$).

Furthermore, a majordomo server ($r_{\text{majordomo}}$) is used for realizing mailing lists, i.e., handling the distribution of mail sent to mailing list addresses. So, the mail sending functionality and the mail receiving functionality are both depending on $r_{\text{majordomo}}$, whenever the receiver address in an e-mail to be sent or received respectively is a mailing list address.

Moreover, the realization of management functionalities includes, in addition to the before described resources, the following: a trouble ticket system (r_{tts}) for incident and problem management purposes, an accounting database ($r_{\text{accounting\_db}}$) for accounting management purposes, and a phone system ($r_{\text{phone\_system}}$) in order to support the realization of most of the management functionalities.

degradations of resources and their entailed degradations of functionalities for the mail service

The introduced resources can be degraded in various ways entailing various degradations of functionalities. Some examples are discussed in the following.

E.g., the main incoming mail server r_{mailin} may suffer a complete outage for some period of time. In this case, the mail receiving functionality for all users is unavailable, as all incoming mails have to pass this server. Furthermore, the mailbox accessing functionality for user groups, which have no extra dedicated mail incoming server, i.e., users not from TUM or LMU, is also unavailable, since such users access their mailboxes directly via this server. Moreover, r_{mailin} may not be completely unavailable, but may only be degraded in some manner, e.g., by having a high CPU load or by its mail receiving queues being overloaded with too many mails waiting to be handled. In the first case, the high CPU load may affect both functionalities which depend on r_{mailin}, i.e., $f_{\text{mail/use/rcv}}$ for all users, as well as $f_{\text{mail/use/mailbox\_access}}$ for users not of LMU or TUM. Nevertheless, this would not cause the complete unavailability of these functionalities, but it would probably result in a high delay and low throughput for the transactions of these functionalities. For example, transfer of e-mails received from r_{mailin} to a mail client of a user of a research institution could be very slow. In the second case, overloaded mail receiving queues of r_{mailin}, the mail receiving functionality for all users will suffer from a high mail receiving delay.

As the handling of mails to mailing lists is realized by a majordomo server $r_{\text{majordomo}}$, a degradation of this server can affect the sending or the receiving of mail to such mailing lists. A complete outage of this server would at least delay the sending and the receiving of e-mails to and from a mailing list, until

the server is back to life. The server being unavailable for a longer period of time, e-mails sent to mailing lists might be lost completely if r_{mailin} finally gives up retrying to relay such e-mails to the unreachable majordomo server. Similar as for one of the normal mail incoming server, a high CPU load at least might cause bad performance in terms of delay and throughput for the handling of mail to mailing list.

A degradation of one of the mail relay servers $r_{\text{mailrelay1}}$ and $r_{\text{mailrelay2}}$, can have an impact on the mail sending functionality (for all users), because any mail sent from a user passes one of these servers. If both are completely unavailable, the whole mail sending functionality is unavailable. However, even if only one of them is unavailable, and the other one is working correctly, a user willing to dispatch an e-mail might only try to use the unavailable one of these servers. The reason for this is the use of Round-Robin DNS method for load-balancing of the two mail relay servers, in combination with the simple handling of DNS responses by some mail clients: Although, each response to a DNS request concerning a respective mail domain handled by the mail service contains the IP addresses of both relay servers, the order of them varies in a round-robin manner. Hence, one time $r_{\text{mailrelay1}}$ is stated to have the highest priority, and another time $r_{\text{mailrelay2}}$ is stated to have the highest priority. Unfortunately, some mail clients only try to use the first of these IP addresses and give up in case of failure, instead of further trying the second one. It depends on the mail client of a user not only to try the mail server with the highest priority. But unfortunately, most mail clients behave this way. So, one of the two servers being not available can cause an apparently random-like partial failing of the sending mail functionality (for all users depending on the used mail client program).

Furthermore, if only one of the servers is available, this one has to handle all mails to be sent, which might result quickly in additional performance degradation of this mail relay server left working. Moreover, if one or both of the mail relay servers are suffering a performance degradation, e.g., a high CPU load or an over-full mail sending queue, the mail sending functionality can at least be degraded in terms of throughput or delay of transactions.

introduction of subservices and their resources of the mail service

The resources discussed so far are used directly for the realization of the mail service. But, in addition to these resources, the realization of the mail service is based on various subservices, for which in turn resources are used to realize them. Degradations of a resource of such a subservice resulting in a degradation of some functionality of that subservice might also affect the mail service. In order to complement the introductions of resources used for realization of the mail service, these subservices together with their functionalities (as far as relevant for the mail service) as well as their dependence on resources are shortly introduced. To allow for an easy reference, each involved (sub)service is assigned a unique identifier (an s with a unique index).

To give an overview, Fig. 2.5 illustrates the dependencies of all subservices for the mail service. Actually, Fig. 2.5 is an instantiation of the MNM Service Model's basic view (Fig. 2.1), with the extensions that services are vi-

sualized as ellipses (as special UML class stereotype notation), and that the customer/user roles are left out. The e-mail service (denoted s_{mail}) is depend-

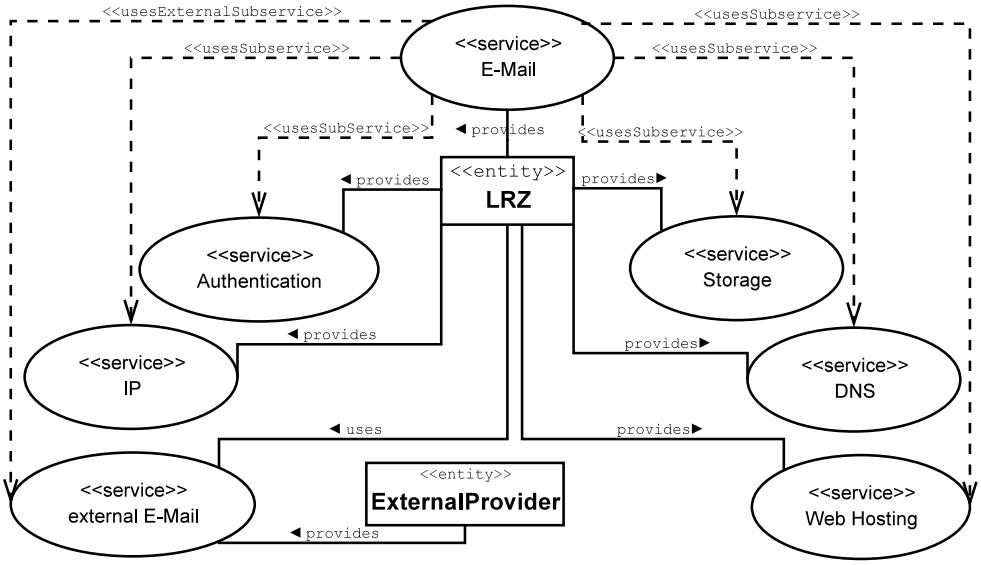

Figure 2.5: Dependencies of the e-mail service on subservices, illustrated as instantiation of the MNM Service model's basic view (see Fig. 2.1)

ing on the following provider-internal subservices: Generally, most of the components as well as the communication with other subservices is based on an IP connectivity service (s_{ip}). Additionally, as SMTP is relying on DNS, a DNS service (s_{dns}) is one of the subservices. Especially, as already stated above, the load-balancing of the two mail relay servers $r_{\mathrm{mailrelay1}}$ and $r_{\mathrm{mailrelay2}}$ is realized by Round-Robin DNS method, i.e., by two different possible orders of IP addresses in the responses to DNS requests concerning the name resolution of mail domains.

For authentication purposes an authentication service (s_{auth}) based on LDAP (Lightweight Directory Access Protocol) is used. Mail folders of users are stored using a storage service (s_{store}) based on AFS (Andrew Filesystem). In fact, the mail folders stored by s_{store} are accessed by the respective mail incoming servers of the respective users groups ($r_{\mathrm{mailin\_tum}}$ for users of TUM, $r_{\mathrm{mailin\_studlmu}}$ for students of LMU, $r_{\mathrm{mailin\_lmu}}$ for non-student users of LMU, and r_{mailin} for other users; compare above).

For sending and receiving of mail to or from external mail domains, the mail service provided by such an external domain (s_{extmail}) can be seen as a external subservice (the term subservice as in the MNM service model also covering peering-like service relationships). As the general e-mail functionality cannot only be accessed directly by using the protocols SMTP, POP/IMAP, but also by a web mail interface being part of the web hosting service, the other example service (compare Sect. 2.3.2), the web hosting service (s_{web}) is subservice of the e-mail service in this respect, too.

The functionality of the used subservices can be regarded as a single functionality or can be further refined, depending upon what is appropriate. So, there are the following usage functionalities of subservices: $f_{\text{auth/use}}$ of service s_{auth}, $f_{\text{store/use}}$ of s_{store}, $f_{\text{extmail/use}}$ of s_{extmail}, $f_{\text{ip/use}}$ of s_{ip}, and specifically $f_{\text{web/use/apage\_special/webmail}}$ of s_{web} (compare Sect. 2.3.2).

IP usage functionality $f_{\text{ip/use}}$ (of service s_{ip}) can be divided into $f_{\text{ip/use/con}}$ (normal ip connectivity between two end systems) and $f_{\text{ip/use/load\_balance}}$ (load-balancing by using an additional device such as a special purpose switch). The IP service, as being also an important subservice for other subservices of s_{mail}, as well as for the web hosting service, will be treated in more detail in Sect. 2.3.3.

Each of the subservices can be degraded in different way which can propagate to a degradation of the dependent mail service: For instance, an increase of the DNS request delay (of functionality $f_{\text{dns/use}}$) will also increase the delay of mail sending ($f_{\text{mail/use/send}}$). Moreover, as for sending a single e-mail multiple DNS requests have to be issued, one for each mail domain involved, i.e., normally at least two - domain of sender as well as domain of a single receiver - an increase of the DNS request delay will result in a multiplied (e.g., doubled) increase of the mail sending delay. Similarly, an increased IP path delay will (possibly multiplied) add to mail sending or mail receiving delay, depending on the actual IP path affected (see Sect. 2.3.3). Also management functionalities of the mail service can be indirectly degraded via degraded (management) functionalities of subservices. e.g., an unavailability of the configuration management of the DNS service ($f_{\text{dns/mgmt/conf}}$ as refinement of $f_{\text{dns/mgmt}}$) leads to an partial unavailability of the configuration management of the mail service, as no configuration update of mail domains is possible any more.

dependencies as abstraction of specific mappings from resource degradations to functionality degradations

As shown by the examples above, degradation of a specific resource can affect certain functionalities which depend on the degraded resource. Furthermore, the degradation of the resource can take place in a different manner and correspondingly can cause different types of degradations of the depending functionalities. That is, in general the mapping from degraded quality of resources (QoR) to degraded quality of (service) functionalities (QoS) has to be performed. But, as a first step, the knowledge that a functionality depends on a resource, not necessarily including the mappings of specific types of degradations of the resource to specific types of degradations of the functionality, gives nevertheless an overview and a first possibility to classify the degradations of resources and their entailed degradations of dependent functionality, simply by stating that a degraded resource might degrade the dependent functionalities in some way.

That is why in order to complement the introduction of this example scenario, all dependencies of functionalities on resources for the mail service as well as its sub services (as far as relevant) are shortly introduced. These dependencies of functionalities can in fact be regarded as a refinement of the dependencies of the mail service on its subservices (cf. Fig. 2.5). Here, a functionality may

depend directly on a resource or indirectly via another functionality, i.e., the functionality of a subservice).

dependencies of functionalities on resources

In general, all usage functionalities ($f_{\text{mail/use}}$) are depending on the IP connectivity functionality of the IP service ($f_{\text{ip/use/con}}$), because all used resource protocols, as SMTP, POP or IMAP, are IP based and all involved resources are inter-connected by the IP network provided by the IP service. Additionally, all usage functionalities are also depending on the DNS service's usage functionality ($f_{\text{dns/use}}$), since all used resource protocols also utilize to a large extent the DNS protocol for resolving hostnames of involved resources to IP addresses. So both for these two dependencies, the depending object (target of the dependency) comprises the full usage functionality of the e-mail service, whereas the object on which something depends (source of the dependency) is a single or the complete usage functionality of one subservice. But there are more refined functionality dependencies, too: The authentication service's usage functionality $f_{\text{auth/use}}$ is not always necessary for all usage functionalities of the e-mail service. In fact, authentication is necessary for sending mail in an authenticated way ($f_{\text{mail/use/send}}(\textit{authentication}\text{=yes})$ - necessary for sending mails from outside of MWN), accessing mail boxes ($f_{\text{mail/use/mbox\_access}}$), for customization ($f_{\text{mail/use/customize}}$) or for web mail access ($f_{\text{mail/use/webmail}}$). Authentication is not necessary for sending mail in an unauthenticated way, or for mere receiving of e-mails. So, for these dependencies, the target is more specific, in that it is only a specific usage functionality (e.g., $f_{\text{mail/use/mbox\_access}}$) which can even be more restricted further by a condition (e.g., $f_{\text{mail/use/send}}(\textit{authentication}\text{=yes})$).

Furthermore, receiving e-mails ($f_{\text{mail/use/recv}}$) as well as the mail box access ($f_{\text{mail/use/mbox\_access}}$) are both depending on the access to the storage where the mail boxes are stored, i.e., on the storage service's usage functionality ($f_{\text{store/use}}$). And in addition to that is sending or receiving mail to or from external domains of course using the usage functionality of an external mail service ($f_{\text{extmail/use}}$).

But not only as for the before mentioned dependencies, the source of a dependency has to be a functionality, it may also be one or more of the resources used for the service's provisioning: Therefore, the sending of mail ($f_{\text{mail/use/send}}$) is always depending on the virtual mail-out server (r_{mailout}) which itself is specifically depending on DNS usage functionality ($f_{\text{dns/use}}$), and on the Round-Robin DNS load-balanced mail relay servers ($r_{\text{mailrelay1}}$ and $r_{\text{mailrelay2}}$).

As DNS based load-balancing of the mail relay server is also used for receiving mails, also mail receiving functionality ($f_{\text{mail/use/recv}}$) has a dependency with the very same dependency targets ($f_{\text{dns/use}}$, $r_{\text{mailrelay1}}$, $r_{\text{mailrelay2}}$). This type of dependency has a dependency target including multiple objects.

Receiving mail in general ($f_{\text{mail/use/recv}}$) is further depending on the additional resources $r_{\text{greylist}}, r_{\text{spamcheck}}, r_{\text{viruscheck}}, r_{\text{blacklist}}$, and r_{mailin}. In addition to that, depending on the customer $f_{\text{mail/use/recv}}$ as well as $f_{\text{mail/use/mbox\_access}}$ are depending on $r_{\text{mailin\_lrz}}$ for LRZ users, on $r_{\text{mailin\_tum}}$ for TUM users, on

$f_{\text{dns/use}} \rightarrow f_{\text{mail/use}}$
$f_{\text{ip/use/con}} \rightarrow f_{\text{mail/use}}$

$r_{\text{mailout}} \rightarrow f_{\text{mail/use/send}}$
$f_{\text{auth/use}} \rightarrow f_{\text{mail/use/send}}(\textit{authentication} = \text{yes})$
$f_{\text{dns/use}}, r_{\text{mailrelay1}}, r_{\text{mailrelay2}} \rightarrow r_{\text{mailout}}$

$f_{\text{dns/use}}, r_{\text{mailrelay1}}, r_{\text{mailrelay2}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{greylist}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{spamcheck}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{viruscheck}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{blacklist}} \rightarrow f_{\text{mail/use/recv}}$
$f_{\text{store/use}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{mailin}} \rightarrow f_{\text{mail/use/recv}}$
$r_{\text{mailin\_lrz}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$
$r_{\text{mailin\_tum}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{TUM})$
$r_{\text{mailin\_lmu}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LMU}, \notin \text{stud})$
$r_{\text{mailin\_studlmu}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LMU} \cup \text{stud})$

$r_{\text{majordomo}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver}\ \text{is mailinglist})$

$f_{\text{extmail/use}} \rightarrow f_{\text{mail/use/recv/extra}}$
$f_{\text{extmail/use}} \rightarrow f_{\text{mail/use/send/extra}}$
$f_{\text{store/use}} \rightarrow f_{\text{mail/use/mbox\_access}}$
$f_{\text{auth/use}} \rightarrow f_{\text{mail/use/mbox\_access}}$
$r_{\text{mailin}} \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LRZ})$
$r_{\text{mailin\_tum}} \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{TUM})$
$r_{\text{mailin\_lmu}} \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LMU}, \notin \text{stud})$
$r_{\text{mailin\_studlmu}} \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LMU} \cup \text{stud})$

$f_{\text{auth/use}} \rightarrow f_{\text{mail/use/customize}}$
$r_{\text{conf\_sv}} \rightarrow f_{\text{mail/use/customize}}$
$f_{\text{mail/use/mbox\_access}} \rightarrow f_{\text{mail/use/customize}}$

$f_{\text{auth/use}} \rightarrow f_{\text{mail/use/webmail}}$
$f_{\text{web/use/apage\_special/webmail}} \rightarrow f_{\text{mail/use/webmail}}$
$f_{\text{mail/use/send}} \rightarrow f_{\text{mail/use/webmail}}$
$f_{\text{mail/use/recv}} \rightarrow f_{\text{mail/use/webmail}}$
$f_{\text{mail/use/mbox\_access}} \rightarrow f_{\text{mail/use/webmail}}$

Table 2.2: Dependencies for the usage functionalities of the e-mail service in $s \rightarrow t$ short notation

$r_{\text{mailin\_lmu}}$ for LMU staff, and on $r_{\text{mailin\_studlmu}}$ for LMU students. So here, the target of the dependency ($f_{\text{mail/use/recv}}$) is restricted by a condition concerning the customer, e.g., $f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$. Additionally, if the receiver is a mailing list, the majordomo server ($r_{\text{majordomo}}$) has to be used, too.

The web mail functionality ($f_{\text{mail/use/webmail}}$) is on the one hand depending on a web mail page access, which is actually provided by the web hosting service (see Sect. 2.3.2), and on the other hand it is depending on all other functionalities of the mail service to which it allows access to. And as already stated above, web mail access always requires authentication.

In Table 2.2 a summary of the dependencies of usage functionalities is given using the short notation $s \rightarrow t$, which means that t (target of the dependency) is depending on s (source of the dependency), where s and t are functionalities or resources potentially further restricted by a condition.

dependencies of management functionalities for the mail service

Not only usage functionalities of the mail service have dependencies, but also this holds for the management functionalities: In fact, some management functionalities (e.g., ordering configuration changes) are itself depending on sending and receiving mails, but in case of emergency can also be done by phone. A trouble ticket system is used for problem and incident management. For most management purposes the configuration server ($r_{\text{conf\_sv}}$) has to be accessed and valid authentication is always necessary for the customer.

But there are further dependencies, e.g., change management of the mail service depends on change of the DNS service, as mail customers can order new mail domains, which have to be installed and configured in the DNS. Table 2.3 gives a summary of these dependencies for management functionalities in short notation $s \rightarrow t$.

$f_{\text{mail/use/send}}, f_{\text{mail/use/recv}}, r_{\text{phone\_system}} \rightarrow f_{\text{mail/mgmt}}$
$r_{\text{tts}} \rightarrow f_{\text{mail/mgmt/prob\_inci}}$
$r_{\text{conf\_sv}} \rightarrow f_{\text{mail/mgmt}}$
$f_{\text{auth}} \rightarrow f_{\text{mail/mgmt}}$
$f_{\text{dns/mgmt/change}} \rightarrow f_{\text{mail/mgmt/change}}$

Table 2.3: Dependencies for the management functionalities of the e-mail service in $s \rightarrow t$ short notation

introduction of dependencies on subservices and their resources for the mail service

In the following, also important dependencies of the subservices from their resources as well as inter-dependencies between these resources (where it is appropriate) will be shortly explained: Similarly as the mail service's usage functionality itself is completely IP based, also the usage functionalities of the authentication service, of the storage service, the DNS service, and of the web hosting service are IP based and therefore highly depending on the IP service's normal connectivity functionality. Furthermore, the functionality of the authentication service is realized by two LDAP servers ($r_{\text{ldap\_sv1}}$ and $r_{\text{ldap\_sv2}}$), the functionality of the storage service is based on a AFS filesystem cluster (consisting of 3 servers: $r_{\text{afs\_sv1}}, r_{\text{afs\_sv2}}, r_{\text{afs\_sv3}}$), and the DNS service

actually utilizes 2 DNS servers ($r_{\text{dns\_sv1}}, r_{\text{dns\_sv2}}$). For the web mail page access functionality ($f_{\text{web/use/webmail}}$), provided by using a dedicated web mail server ($r_{\text{webmail\_sv}}$) as part of the web hosting service, refer to Sect. 2.3.2.

Table 2.4 summarizes in short notation the mentioned dependencies of subservices of the e-mail service, which are particularly of interest regarding the e-mail service. The dependencies of all above introduced resources on functionalities and resources of the IP service are covered in detail in Sect. 2.3.3.

$f_{\text{ip/use/con}} \rightarrow f_{\text{auth}}$
$r_{\text{ldap\_sv1}}, r_{\text{ldap\_sv2}} \rightarrow f_{\text{auth}}$

$f_{\text{ip/use/con}} \rightarrow f_{\text{store}}$
$r_{\text{afs\_sv1}}, r_{\text{afs\_sv2}}, r_{\text{afs\_sv3}} \rightarrow f_{\text{store}}$

$f_{\text{ip/use/con}} \rightarrow f_{\text{dns}}$
$r_{\text{dns\_sv1}}, r_{\text{dns\_sv2}} \rightarrow f_{\text{dns}}$

$f_{\text{ip/use/con}} \rightarrow f_{\text{web}}$
$f_{\text{dns/use}} \rightarrow f_{\text{web}}$
$r_{\text{webmail\_sv}} \rightarrow f_{\text{web/use/webmail}}$

Table 2.4: Dependencies for subservices of the e-mail service in $s \rightarrow t$ short notation

In general, there might be also dependencies between resources, e.g., a server is depending on his CPU and his storage devices. For the resources introduced above which are used for the mail service, such details are not considered here for simplicity. However, the detailed inter-dependencies among the network resources of the IP service are described in Sect. 2.3.3.

refined graphical illustration of dependencies

In Fig. 2.6 on page 36 the dependencies on resources as well as subservice functionalities for the usage functionalities of e-mail service are visualized in a graphical notation. This illustration gives an overview of all dependencies which have been discussed above. Actually, this own graphical notation for dependencies is introduced and used, as existing dependency notations usually lack an important feature, namely the possibility to express the inheritance hierarchy between functionalities, i.e., the relationship of a more general functionality and more refined subfunctionalities.

The following notation conventions are utilized:

- functionalities of the service s_{mail} are depicted as non-shaded, concentric ellipses, starting with a big ellipsis for $f_{\text{mail/use}}$, with the inclusion of smaller ellipses (e.g., $f_{\text{mail/use/recv}}$) within a bigger one (e.g., $f_{\text{mail/use}}$) denoting the inheritance relationship of the visualized functionalities.
- functionalities of subservices are depicted also as ellipses, having a shaded background, and being positioned around the ellipses of $f_{\text{mail/use}}$.

- resources of the mail service functionalities or resources of subservices are visualized by specific resource icons.

- dependencies between resources and functionalities (of the service or subservices) as shown as dashed lines.

This illustration of functionalities as ellipses is inspired by and borrowed from the UML notation of use cases, as in the MNM service model functionalities or at least their corresponding realizing processes are also considered as use cases. Similar, the notion of dependencies is borrowed from UML. Only inheritance of functionalities, i.e., inheritance of use cases, is - to give an simple overview - visualized by ellipses containment in contrast to UML inheritance notation. Alternatively, the inheritance relationship could be represented by normal UML inheritance notation.

Actually, Fig. 2.6 is a refinement of Fig. 2.4 on p. 26: It distinguishes different usage functionalities as refinement of $f_{\mathrm{mail/use}}$, and also takes into account dependencies on subservices. With respect to the latter one of these two refinements, Fig. 2.6 can also be regarded as a refinement of Fig. 2.5 on p. 29, because it also illustrates in a more detailed manner the dependencies on subservices for the mail service. Though, for readability, Fig. 2.6 as a refinement of Fig. 2.5, only covers usage functionalities of the mail service, not management functionalities.

consideration of business impact

So far, only resources, some of their possible degradations, and the mapping of these degradations to degradations of functionalities (for respective groups of users and customers) of the mail service have been treated. But I/R analysis has the final goal to decide how critical a given set of resource degradations is and what is the best way to recover from it. How critical a resource degradation is, depends on the severity of the degradations of functionalities it entails. Thus, after having identified the degradations of functionalities a resource degradation entails, the severity of these degradations of functionalities has to be evaluated. On the one hand, this severity is depending on the extent and order of magnitude of the entailed functionality degradations, but on the other hand it largely depends on the actual importance of the degraded functionalities.

Of course, the importance of a specific functionality seen from the provider's point of view might be different from the importance seen from a user's or customer's point of view. As I/R analysis is performed by the service provider, his point of view is used here for the evaluation of the importance of functionalities. But often the importance of a functionality or some specific aspect of it (e.g., specific QoS parameter ranges) for a customer or a user is expressed by constraints in the SLA between the customer and the provider. Violating these constraints, can cause high SLA violation penalties for the provider implying a high impact on his business. So, the provider, in the first place should have an interest in minimizing degradations of functionalities which cause high violation costs. Therefore, the importance of a functionality (and so of a degradation of a functionality in a certain extent) is evaluated on

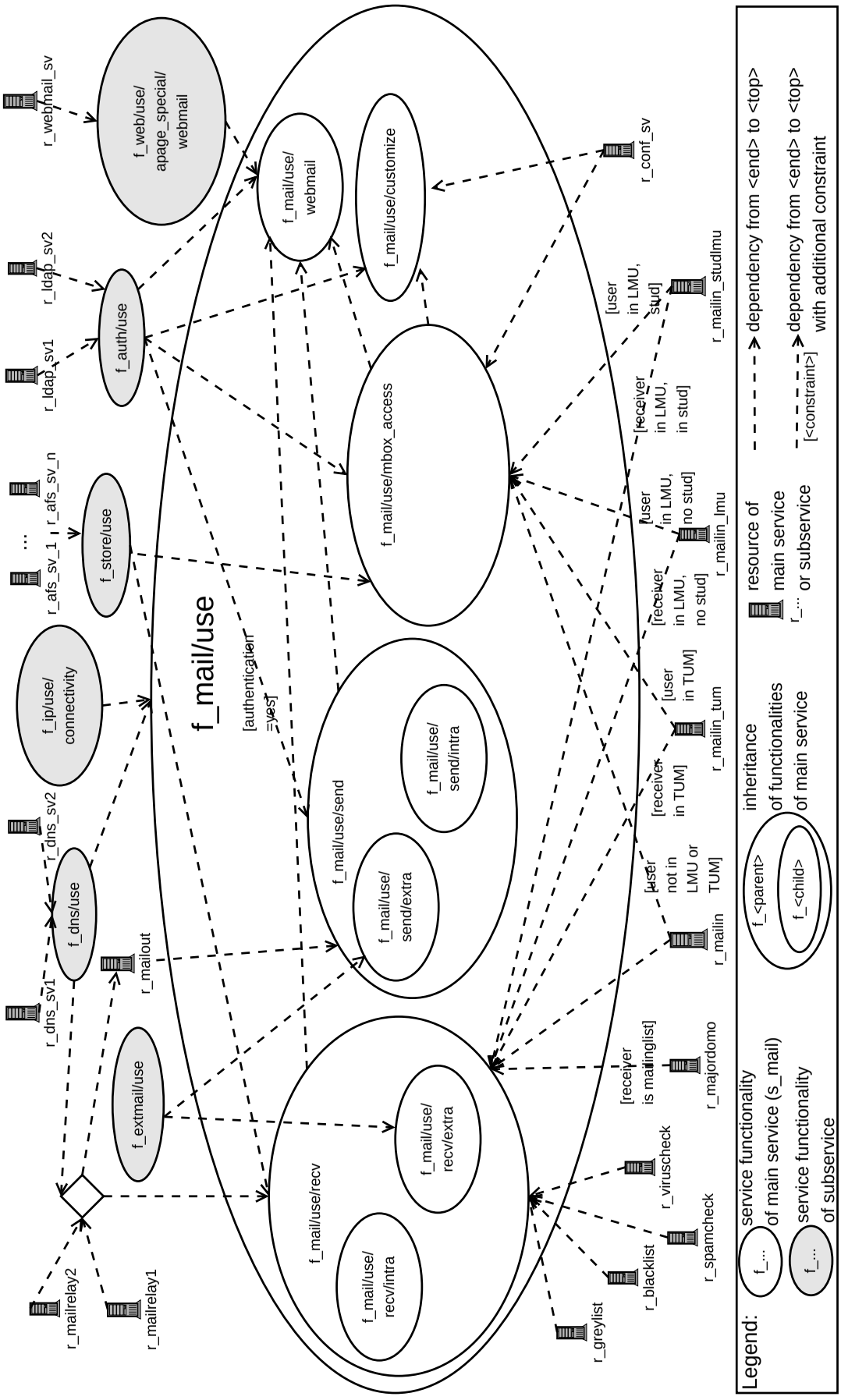

Figure 2.6: Dependencies on resources and subservice functionalities for the usage functionalities of the e-mail service

the basis of the specified SLA constraints and SLA penalty definitions. As an extension, later on, other possibilities for the definition and derivation of importance of certain functionalities and their degradations might also be used. This could include factors such as loss of future revenue because of customers canceling their contract or loss of public reputation in general.

As SLA constraints and SLA penalty definitions are assumed to be relevant for the evaluation of the importance of functionalities and their degradations, the introduction of the e-mail service scenario is completed in the following by discussing the specific SLA constraints and related penalty definitions for the mail service.

introduction of QoS parameters of the mail service

The SLA constraints between the LRZ and its customers are defined in terms of some QoS parameters agreed upon between LRZ and the customers. These QoS parameters will be shortly introduced and afterwards the SLA constraints and penalty definitions based on them are discussed.

In general, for any *QoS parameter* the used *QoS measurement metric* as well as the actual *QoS measurement methodology* is important. Moreover, concerning degradation dependencies, for a QoS parameter the *QoS parameter subject*, i.e., the set of related functionalities to which it is assigned, has to be specified: A degradation of a QoS parameter assigned to a functionality possibly causes this functionality to be degraded - be it partially or fully. A single QoS parameter may be related to a single functionality, to a set of functionalities or even to the whole service functionality. E.g., there might be a general QoS parameter availability covering the whole service, or a specific QoS parameter covering only the availability of a specific functionality. For a given QoS parameter, its subject is related to its measurement methodology somehow, as some (but not necessary all) of its assigned functionalities are actually measured with respect to the QoS parameter in order to determine the actual value of the QoS parameter. E.g., the actual measurement of a general availability of the whole service functionality might be measured by only monitoring one or two critical functionalities, though it may be assigned to all functionalities. Additionally to the list of associated functionalities, the subject might further be specifically restricted to a specific subset of customers. That is why a QoS parameter here is defined by a name identifying the QoS parameter in the context of the given service, as well as its QoS metric, its QoS measurement methodology, and its QoS parameter subject.

For the mail service, there are the following QoS parameters: general availability ($q_{\mathrm{mail/avail\_general}}$), availability per functionality ($q_{\mathrm{mail/avail\_specific}}$), general reliability ($q_{\mathrm{mail/reliab\_general}}$), reliability per specific functionality ($q_{\mathrm{mail/reliab\_specific}}$), intra mail sending delay ($q_{\mathrm{mail/delay\_send\_intra}}$), mail sending delay to outer domain ($q_{\mathrm{mail/delay\_send\_extra}}$), available bandwidth for mailbox access ($q_{\mathrm{mail/bandwidth\_mbox}}$), mailbox access delay ($q_{\mathrm{mail/delay\_mbox}}$), and request delay for account customization ($q_{\mathrm{mail/delay\_customize}}$). Each of these QoS parameters has a defined metric and corresponding measurement methodology whose specification is of the mail service SLA.

Availability per specific functionality is measured with a standard test interac-

tion specific for the respective functionality. The availability of intra and extra mail sending functionality as well as mail box access functionality are averaged to the general availability for the whole service. Reliability per functionality is defined as mean time to repair with respect to respective availability per functionality. Whereas reliability for the whole service is computed as the mean over all reliability per functionality values.

Intra sending delay, which is only assigned to intra mail domain sending functionality, is measured with standard test mail sending requests (mail size 100 Kb) to specifically defined test receiver mail sites within MWN. In a similar manner, the mail sending to the outside IP world (assigned to extra mail sending functionality) is tested with standard mail sending requests to some outside mail domains. All 15 minutes multiple test mail sending requests are performed and the average delay duration computed out of these as the specific delay QoS parameter.

Available bandwidth for mailbox access is assured by determining and recording the link utilization (peak value of 1-minute interval) of the network links connecting the mail incoming servers with the LRZ network's core backbone router ($r_{\mathrm{rt\_core}}$, see Fig. 2.9). In fact, this QoS measurement is done by the IP service as a subservice and the results are relayed to the mail service's management.

Also, mailbox access delay (assigned to mailbox access functionality) is tested and measured with a periodically executed and automated testing tool accessing test accounts on different mail incoming servers for the different research institutions. Different research institutions have different mail incoming servers for mail box access, and so mailbox delay is computed individually for each of the incoming servers. In a similar manner, customization request delay (assigned to account customization functionality) is measured by different test customization interactions with test accounts, also individually performed for each incoming server.

QoS parameters for management functionality are not covered here, but could also be defined. Table 2.5 provides a summary of all above discussed QoS parameters.

Each of the QoS parameters, e.g., $q_{\mathrm{mail/avail\_specific}}(f_{\mathrm{mail/use/mbox\_access}})$, is measured or calculated in general for all customers, e.g., $q_{\mathrm{mail/avail\_specific}}(f_{\mathrm{mail/use/mbox\_access}}(\textit{customer} = \mathrm{any}))$, as well as for a single customer, e.g., $q_{\mathrm{mail/avail\_specific}}(f_{\mathrm{mail/use/mbox\_access}}(\textit{customer} = \mathrm{TUM}))$ where the particular QoS parameter specification may differ for different customers (e.g., for availability of mail box access, as mail boxes of specific users can become inaccessible because of storage failures).

specification of SLA conditions for the mail service

In addition to the sole specification of QoS parameters, for each parameter specific value ranges or, in general specific *SLA constraints*, for defining particular service levels, as well as *SLA penalty costs* for not meeting these constraints have to be defined depending on the given SLAs agreed with the customers.

QoS parameters		
name:	**description:**	**subject:**
$q_{\text{mail/avail\_general}}$	availability of the whole usage functionality	$f_{\text{mail/use}}$
$q_{\text{mail/avail\_specific}}(fcty)$	availability specific to one of the subfunctionalities (fcty)	$f_{\text{mail/use/send}}$, $f_{\text{mail/use/recv}}$, $f_{\text{mail/use/mbox\_access}}$, $f_{\text{mail/use/customize}}$, $f_{\text{mail/use/web}}$
$q_{\text{mail/reliab\_general}}$	reliability of the whole usage functionality	$f_{\text{mail/use}}$
$q_{\text{mail/reliab\_specific}}(fcty)$	reliability specific to one of the subfunctionalities (fcty)	$f_{\text{mail/use/send}}$, $f_{\text{mail/use/recv}}$, $f_{\text{mail/use/mbox\_access}}$, $f_{\text{mail/use/customize}}$, $f_{\text{mail/use/web}}$
$q_{\text{mail/delay\_send\_intra}}$	intra-domain e-mail sending delay	$f_{\text{mail/use/send/intra}}$
$q_{\text{mail/delay\_send\_extra}}$	e-mail sending delay to next outer domain	$f_{\text{mail/use/send/extra}}$
$q_{\text{mail/bandwidth\_mbox}}$	available bandwidth of mail box access	$f_{\text{mail/use/mbox\_access}}$
$q_{\text{mail/delay\_mbox}}$	delay of mail box access	$f_{\text{mail/use/mbox\_access}}$
$q_{\text{mail/delay\_customize}}$	delay of mailbox/account customization request	$f_{\text{mail/use/customize}}$

Table 2.5: QoS Parameters of the e-mail service

For the e-mail service scenario some explicit value range requirements, i.e., SLA constraints, and associated *SLA penalty definitions* are specified between LRZ and its premium customers, namely the universities TUM and LMU. These constraints and penalty definitions are introduced in the following. The particular value (ranges) and their particular form in which they are presented is only to be meant as one representing example for specifying such constraints and penalties. Moreover, these definitions will be used for illustrating examples in the rest of the thesis:

The QoS parameter $q_{\text{mail/avail\_general}}$ (availability of whole $f_{\text{mail/use}}$) has to meet the requirement $q_{\text{mail/avail\_general}} > 99\%$ (per week-basis, regarding only business hours: Mon-Fri, 8:00 - 18:00). Each occurring 1% deviation below 99% leads to a penalty of 500 € for premium customers.

Furthermore, $q_{\text{mail/reliab\_general}}$ (reliability of whole $f_{\text{mail/use}}$) has the requirement that no outage of the most important functionalities ($f_{\text{mail/use/send}}$, $f_{\text{mail/use/recv}}$, $f_{\text{mail/use/mbox\_access}}$) takes longer than 30 minutes, i.e., $q_{\text{mail/reliab\_general}} < 30\ \text{min}$. For premium customers, the penalty for outages not meeting this, is 5000 € plus 10 € per additional minute above 30 per

outage.

Finally, the average delay for sending mails ($q_{\text{mail/delay\_send\_intra}}$, and $q_{\text{mail/delay\_send\_extra}}$ as far as its responsibility of the LRZ for sending mails to outside domains) has to be below 5 minutes. For each 1-hour interval in which a violation of this occurs, a penalty of 300 € has to be paid to premium customers.

Table 2.6 summarizes SLA constraints and SLA penalty definitions for premium customers as introduced above.

SLA regulations	
QoS/SLA constraint:	**SLA penalty:**
general availability $q_{\text{mail/avail\_general}} > 99\%$ (per week, only during business hours) $[sla\_cnstr_{\text{mail1}}]$	for each 1% deviation below 99%: 500 € $[sla\_pnlty_{\text{mail1}}]$
general reliability: $q_{\text{mail/reliab\_general}} < 30$ min $[sla\_cnstr_{\text{mail2}}]$	penalty per outage > 30 min: 5000 € $+ 10$ € $* duration$ /min $[sla\_pnlty_{\text{mail2}}]$
mail sending delay intra-domain $q_{\text{mail/delay\_send\_intra}}$ or extra-domain $q_{\text{mail/delay\_send\_extra}} < 5$ min $[sla\_cnstr_{\text{mail3}}]$	each hour interval in which violation occurs: 300 € $[sla\_pnlty_{\text{mail3}}]$

Table 2.6: QoS constraints and associated SLA violation penalties for the e-mail service

This section introduced the mail service scenario of the LRZ, serving as the first example scenario in the remainder of the thesis. All functionalities as well as some possible types of degradations of them were explained. Moreover, resources, some of their degradations, as well as the general dependence of functionalities on them were presented. Combining every given piece of information given, it is possible to derive specific degradations on functionalities entailed by specific degradations of resources. Finally, defined QoS parameters and SLA conditions were given to allow for further determination of business impact.

Sect. 2.3.2 introduces the web hosting service in a similar manner. In Sect. 2.3.4, an example run of I/R analysis covering both introduced example services is given, starting from two resource degradations and determining the entailed degradations of dependent functionalities for respective user groups as well as the caused business impact by SLA penalties.

2.3.2 Web Hosting Service

In this section, the second example service, which is also used throughout the thesis, is introduced. This is done in similar terms as in Sect. 2.3.1 for the first example service, and it will be referred to Sect. 2.3.1 as far as it is appropriate.

First, the scenario will be briefly explained. Second, its functionalities, its subservices and used resources as well as the dependence of functionalities on resources and on subservices are given. Last, QoS parameters and respective SLA constraints as well as SLA penalties are defined.

introduction of web hosting service scenario

In addition to the e-mail service, the LRZ operates and provides a web hosting service based on HTTP (Hypertext Transfer Protocol). Basically, this service provides virtual web servers to its customers. Users and customers are very similar to the ones of the e-mail service. But on the customer side a user can also be any person using a web browser to access the web pages, not being restricted to students or research institution staff. Therefore, an additional type of user, "anonymous user", exists.

usage ($f_{\text{web/use}}$)
authenticate ($f_{\text{web/use/auth}}$)
access web page ($f_{\text{web/use/apage}}$)
access static web page ($f_{\text{web/use/apage/static}}$)
access dynamic web page ($f_{\text{web/use/apage/dynamic}}$)
access cgi web page ($f_{\text{web/use/apage/dynamic/cgi}}$)
access php web page ($f_{\text{web/use/apage/dynamic/php}}$)
access special web page ($f_{\text{web/use/apage\_special}}$)
web mail access ($f_{\text{web/use/apage\_special/webmail}}$)
trouble ticket system web access ($f_{\text{web/use/apage\_special/tts}}$)
mysql web configuration access ($f_{\text{web/use/apage\_special/mysqlconf}}$)
general configuration web access ($f_{\text{web/use/apage\_special/conf}}$)
management ($f_{\text{web/mgmt}}$)
inquiry and order management ($f_{\text{web/mgmt/inq\_order}}$)
configuration management ($f_{\text{web/mgmt/conf}}$)
problem and incident management ($f_{\text{web/mgmt/prob\_inci}}$)
quality and security management ($f_{\text{web/mgmt/qual\_sec}}$)
accounting management ($f_{\text{web/mgmt/acc}}$)
change management ($f_{\text{web/mgmt/change}}$)

Table 2.7: Overview of the functionalities of the web hosting service

introduction of functionalities

The usage functionality of the web hosting service can be divided into various single functionalities: Mainly, the usage functionality includes access to web pages of virtual web servers for the different research institutions. But there, access to static and dynamic web pages can be differentiated. Even more, dynamic pages can be realized with CGI scripts or with the PHP programming language. As each of these types of web pages has different dependencies on resources, it is useful to differentiate between them even if a normal user, i.e., a person accessing the web page with a web browser, will not realize the difference in functionality itself. For web pages which are restricted for specific user groups, a HTTP authentication functionality is provided. Additionally, to the before-mentioned access to normal web pages of virtual web servers, there are some special web pages accessible, partly for the management of

the web hosting service or even other services. Table 2.7 gives an overview of functionalities of the web hosting service.

introduction of subservices

The web hosting service (s_{web}) uses the following subservices, which are partly the same subservices as for the e-mail service: IP service (s_{ip}), authentication service (s_{auth}), DNS service (s_{dns}), AFS storage service (s_{store}), NFS storage service ($s_{\text{nfs\_store}}$), and database service (s_{db}). In Fig. 2.7 the dependencies on subservices for the web hosting service are illustrated, as instantiation of the MNM Service Model's basic view (see Fig. 2.1 in general, and also compare Fig. 2.5 of the mail service example).

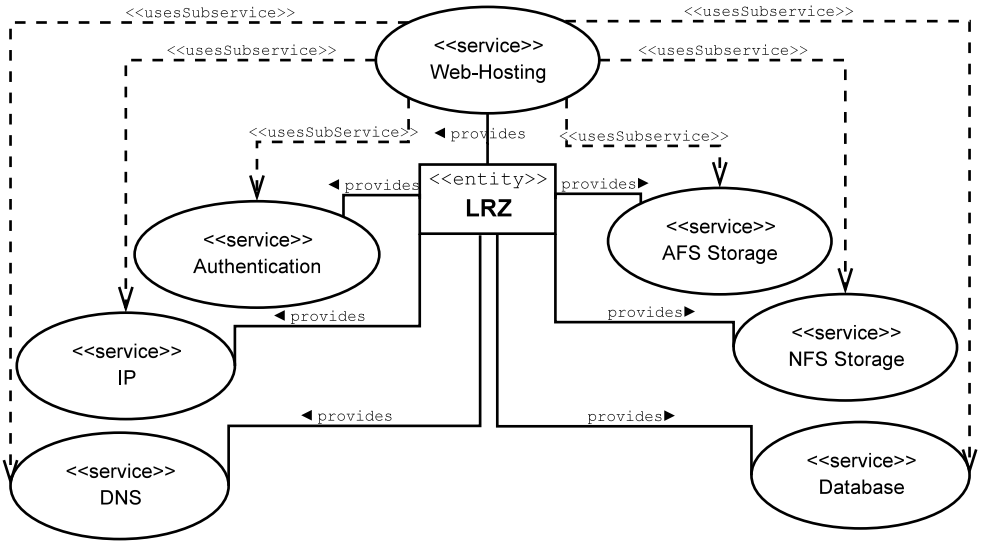

Figure 2.7: Dependencies on subservices of the web hosting service, as instantiation of the MNM Service Model's basic view (Fig. 2.1)

refined illustration of dependencies

In Fig. 2.8 on the dependencies on resources as well as subservice functionalities for the usage functionalities of web hosting service are illustrated. The notation conventions used are the same as in Fig. 2.6 on page 36 for the mail service, which are explained on page 34. Similarly, Fig. 2.8 can also be regarded as a refinement of Fig. 2.7, though being restricted to usage functionalities due to the purpose of readability.

introduction of resources

Resources utilized directly by the web hosting service and corresponding dependencies of the functionalities are the following: 10 web servers ($r_{\text{websv(x)}}, x \in \{1 \ldots 10\}$), each with a specific server configuration, an emergency server ($r_{\text{web\_emerg}}$), 2 caching servers ($r_{\text{webcache(x)}}, x \in \{1, 2\}$), and a dedicated web mail server ($r_{\text{webmail\_sv}}$).

Each of the 10 apache web servers can be in one of the following configurations: *normal* (supports normal virtual web servers of most customers), *lrz* (for lrz web pages), *special* (allows most special web pages, except from web mail and trouble ticket system web access), and *tts* (for trouble ticket system web access). Web servers with the same configuration are load-balanced be the an extra load-balancing switch located in the IP service.

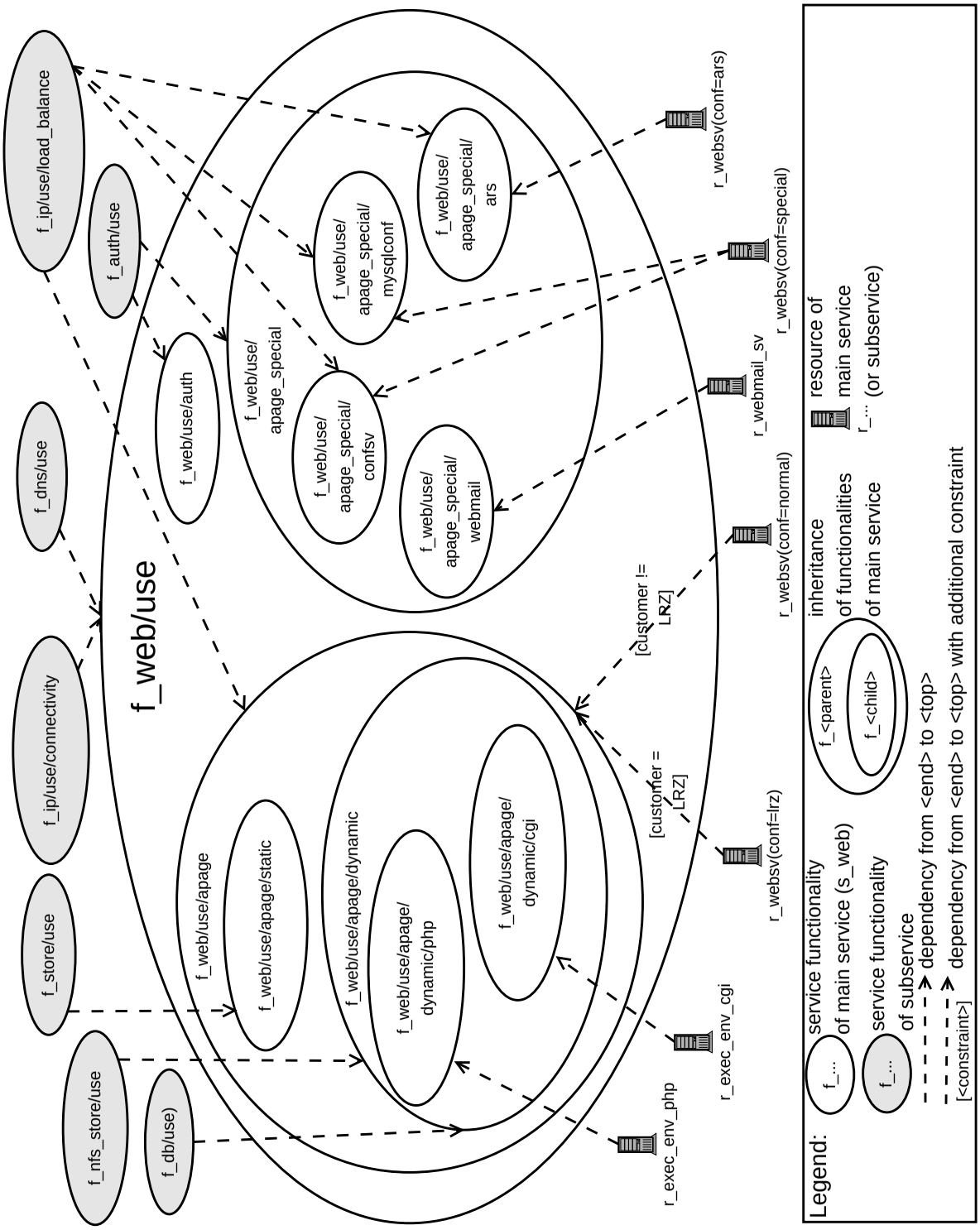

Figure 2.8: Dependencies on resources and subservice functionality for the usage functionalities of the web hosting service

General dependencies for the web service:

$f_{\text{ip/use/con}} \rightarrow f_{\text{web/use}}$
$f_{\text{dns/use}} \rightarrow f_{\text{web/use}}$
$f_{\text{auth/use}} \rightarrow f_{\text{web/use/auth}}$

Dependencies for the load-balancing of the 10 web servers:

$f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}}$ and for $x \in \{1 \ldots 10\}$:
$r_{\text{websv(x)}}(\textit{configuration} = \text{normal}) \rightarrow f_{\text{web/use/apage}}(\textit{customer} = \text{TUM})$
$r_{\text{websv(x)}}(\textit{configuration} = \text{normal}) \rightarrow f_{\text{web/use/apage}}(\textit{customer} = \text{LMU})$
$r_{\text{websv(x)}}(\textit{configuration} = \text{normal}) \rightarrow f_{\text{web/use/apage}}(\textit{customer} = \text{research institute})$

More specifically expressing the load-balancing by
the coordinating ip service functionality $f_{\text{ip/use/load\_balance}}$:

$r_{\text{websv(x)}} \left(\begin{smallmatrix} \textit{configuration} \\ = \text{normal} \end{smallmatrix} \right)_{\substack{x= \\ 1,\ldots,10}}, f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}} \left(\begin{smallmatrix} \textit{customer} \\ = \text{TUM} \end{smallmatrix} \right)$
$r_{\text{websv(x)}} \left(\begin{smallmatrix} \textit{configuration} \\ = \text{normal} \end{smallmatrix} \right)_{\substack{x= \\ 1,\ldots,10}}, f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}} \left(\begin{smallmatrix} \textit{customer} \\ = \text{LMU} \end{smallmatrix} \right)$
$r_{\text{websv(x)}} \left(\begin{smallmatrix} \textit{configuration} \\ = \text{normal} \end{smallmatrix} \right)_{\substack{x= \\ 1,\ldots,10}}, f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}} \left(\begin{smallmatrix} \textit{customer} \\ = \text{research} \\ \text{institute} \end{smallmatrix} \right)$

Further dependencies for the web service:

$f_{\text{store/use}} \rightarrow f_{\text{web/use/apage/static}}$
$r_{\text{execenv\_php(x)}} \rightarrow f_{\text{web/use/apage/dynamic/php}}$
$r_{\text{execenv\_cgi(x)}} \rightarrow f_{\text{web/use/apage/dynamic/cgi}}$
$f_{\text{nfs\_store/use}} \rightarrow f_{\text{web/use/dynamic/cgi}}$
$f_{\text{db/use}} \rightarrow f_{\text{web/use/dynamic}}$
$r_{\text{websv(x)}}(\textit{configuration} = \text{LRZ}) \rightarrow f_{\text{web/use/apage}}(\textit{customer} = \text{LRZ})$

$f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage\_special/mysqlconf}}$
$f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage\_special/conf}}$
$f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage\_special/tts}}$
$r_{\text{websv(x)}}(\textit{configuration} = \text{special}) \rightarrow f_{\text{web/use/apage\_special/mysqconf}}$
$r_{\text{websv(x)}}(\textit{configuration} = \text{special}) \rightarrow f_{\text{web/use/apage\_special/conf}}$
$r_{\text{websv(x)}}(\textit{configuration} = \text{tts}) \rightarrow f_{\text{web/use/apage\_special/tts}}$
$f_{\text{web/use/auth}} \rightarrow f_{\text{web/use/apage/special}}$
$r_{\text{webmail\_sv}} \rightarrow f_{\text{web/use/apage\_special/webmail}}$

Table 2.8: Dependencies of the web hosting service in $s \rightarrow t$ notation

Static web pages ($f_{\text{web/use/apage/static}}$) are located in an AFS filesystem provided by the storage service (s_{store}). For allowing to run dynamic web pages, each web server has the ability to execute either CGI scripts or PHP scripts (functionalities $f_{\text{web/use/apage/dynamic/cgi}}$ and $f_{\text{web/use/apage/dynamic/php}}$). This requires working installations of appropriate execution environments of CGI and PHP on each web server which are represented as extra resources $r_{\text{execenv\_cgi(x)}}$ and $r_{\text{execenv\_php(x)}}$ for $x \in \{1 \ldots 10\}$. Moreover, the CGI scripting environments use the NFS storage service ($s_{\text{nfs\_store}}$), and both of the execution environments use the database service (s_{db}) for storing their data. The web mail server is specifically utilized for the web mail access, which is actually used as a subservice for the mail service (compare with Sect. 2.3.1).

Additionally a trouble ticket system (r_{tts}) and a phone system ($r_{\text{phone\_system}}$) are used for management purposes in a similar manner as for the e-mail service.

Usage functionalities of subservices are (compare with e-mail service): $f_{\text{ip/use/con}}$, $f_{\text{ip/use/load\_balance}}$, $f_{\text{dns/use}}$, $f_{\text{auth/use}}$, $f_{\text{store/use}}$, $f_{\text{nfs\_store/use}}$, $f_{\text{db/use/oracle}}$, $f_{\text{db/use/mysql}}$. The database service (s_{db}) provides 2 different types of databases, Oracle Database and Mysql Database represented as different functionalities.

dependencies of functionalities on resources and subservices for the web hosting service

Similarly to the mail service, all usage functionalities of the web hosting service depend in general on the connectivity functionality of the IP service and on the usage functionality of the DNS service.

The HTTP authentication functionality ($f_{\text{web/use/auth}}$) in fact uses the authentication usage functionality ($f_{\text{auth/use}}$), which is based on LDAP. This authentication functionality is necessary for restricted customer web pages, as well as all special pages, such as trouble ticket system access, mysql database configuration interface, and web mail access.

In Table 2.8 a list of the functionality dependencies for the web hosting service is given, in the short notation $s \rightarrow t$, which denotes that t (target of the dependency) is depending on s (source of the dependency), where s and t are functionalities or resources.

$r_{\text{nfssv}} \rightarrow f_{\text{nfs\_store/use}}$
$r_{\text{mysqlsv}} \rightarrow f_{\text{db/use/mysql}}$
$r_{\text{orcalesv1}} \rightarrow f_{\text{db/use/mysql}}$
$r_{\text{orcalesv2}} \rightarrow f_{\text{db/use/mysql}}$

Table 2.9: Dependencies of the subservices for the web service in $s \rightarrow t$ notation (in addition to the dependencies given in Table 2.4)

The functional dependencies for subservices of the web hosting service are partly similar as in the case of the mail service (compare with Sect. 2.3.1). In addition to this it can be said that the NFS storage service is based on an NFS filesystem server (r_{nfssv}), and the database service realizes its both

different usage functionalities with a mysql server (r_{mysqlsv}) and two Oracle database servers ($r_{\text{oraclesv1}}$ and $r_{\text{oraclesv2}}$). These additional dependencies for subservice are illustrated in Table 2.9. The specific resources and interdependencies of the IP service concerning the web hosting service can be found in the following Sect. 2.3.3.

QoS parameters		
name:	**description:**	**subject:**
$q_{\text{web/avail\_general}}$	availability of the whole usage functionality	$f_{\text{web/use}}$
$q_{\text{web/avail\_specific}}(fcty)$	availability specific to one of the subfunctionalities (fcty)	$f_{\text{web/use/apage/static}}$, $f_{\text{web/use/apage/dynamic/cgi}}$, $f_{\text{web/use/apage/dynamic/php}}$
$q_{\text{web/reliab\_general}}$	reliability of the whole usage functionality	$f_{\text{web/use}}$
$q_{\text{web/reliab\_specific}}(fcty)$	reliability specific to one of the subfunctionalities (fcty)	$f_{\text{web/use/apage/static}}$, $f_{\text{web/use/apage/dynamic/cgi}}$, $f_{\text{web/use/apage/dynamic/php}}$
$q_{\text{web/apage\_delay}}$	average web page access delay (for static web pages)	$f_{\text{web/use/apage/static}}$
$q_{\text{web/access\_bandwidth}}$	bandwidth for web page access	$f_{\text{web/use/apage}}$
$q_{\text{web/mgmt\_bandwidth}}$	bandwidth for web page management	$f_{\text{web/mgmt/change}}$
$q_{\text{web/resolution\_time}}$	average help desk problem resolution time (per week)	$f_{\text{web/mgtm/prob\_inci}}$

Table 2.10: QoS parameters of the web hosting service

QoS parameters, SLA constraints, and penalties for the web hosting service

Similarly to the e-mail service, for the web hosting service QoS parameters together with corresponding SLA constraints and SLA penalties are defined for the web hosting service. The following QoS parameters, being summarized in Table 2.10, are defined for the web hosting service: Similar as for the e-mail service, there is a general availability ($q_{\text{web/avail\_general}}$) for $f_{\text{web/use}}$, as well as a functionality-specific availability ($q_{\text{web/avail\_specific}}(fctq)$). Also, general reliability ($q_{\text{web/reliab\_general}}$) for $f_{\text{web/use}}$, as well as a functionality-specific reliability ($q_{\text{web/reliab\_specific}}(fcty)$) is defined as a QoS parameter. The average delay of web page access ($q_{\text{web/apage\_delay}}$) is measured by some defined regularly scheduled test transactions. In addition to this, the available bandwidth for web page access ($q_{\text{web/access\_bandwidth}}$) as well as web page management ($q_{\text{web/mgmt\_bandwidth}}$) is ensured. For incident and problem management, the average help-desk problem resolution time (per week) $q_{\text{web/resolution\_time}}$ is ensured. Again, for each of these QoS parameters, specific QoS metrics and corresponding QoS measurement methodologies are defined. Moreover, Ta-

ble 2.11 lists respective SLA constraints and SLA penalties concerning these QoS parameters. Again, their actual values and their representation is meant only as an example, which will mainly serve for illustration purpose.

SLA regulations	
QoS/SLA constraint:	**SLA penalty:**
general part:	
average help-desk problem resolution time (per week) < 3 h: $q_{\text{web/resolution\_time}} < 3$ h	
guaranteed bandwidth for web page management: 1Mb/s	
premium customers:	
general availability $q_{\text{web/avail\_general}} > 95\%$ (per week, only during business hours) $[sla\_prem\_cnstr_{\text{web1}}]$	for each 0.1% deviation below 95%: 100 € $[sla\_prem\_pnlty_{\text{web1}}]$
general reliability: $q_{\text{web/reliab\_general}} < 30$ min $[sla\_prem\_cnstr_{\text{web2}}]$	penalty of outage for each hour: 7000 € $[sla\_prem\_pnlty_{\text{web2}}]$
average web page access delay < 20 s (up to 100kB): $q_{\text{web/apage\_delay}} < 20$ s $[sla\_prem\_cnstr_{\text{web3}}]$	each 15-min interval in which violation occurs: 100 € $[sla\_prem\_pnlty_{\text{web3}}]$
normal customers:	
general availability $q_{\text{web/avail\_general}} > 90\%$ (per day, only during business hours) $[sla\_norm\_cnstr_{\text{web1}}]$	for each 5% deviation below 90%: 100 € $[sla\_norm\_pnlty_{\text{web1}}]$
general reliability: $q_{\text{web/reliab\_general}} < 1$ h $[sla\_norm\_cnstr_{\text{web2}}]$	penalty of outage for each hour: 1000 € $[sla\_norm\_pnlty_{\text{web2}}]$
average web page access delay < 30 s (up to 100kB): $q_{\text{web/apage\_delay}} < 30$ s $[sla\_norm\_cnstr_{\text{web3}}]$	each 1-hour interval in which violation occurs: 10 € $[sla\_norm\_pnlty_{\text{web3}}]$
students (student web server):	
best-effort, no guarantees	no penalties

Table 2.11: QoS constraints and associated SLA violation penalties for the web hosting service

Here, the web hosting service of the LRZ, which serves as a second example service in this thesis, was presented. Functionalities, their general dependence on resources and subservices, as well as SLA conditions were introduced in a similar manner as for the mail service in Sect. 2.3.1.

In Sect. 2.3.4, an example run of I/R analysis covering both introduced example services is given, starting from two resource degradations and determining the entailed degradations of dependent functionalities for respective user groups as well as the caused business impact by SLA penalties.

2.3.3 IP Service

In addition to the before described web hosting service (s_{web}) and e-mail service (s_{mail}), the LRZ's IP service (s_{ip}) is now described in more detail, as this service is an important subservice for both services as well as most subservices of both, and its resources have important inter-relationships concerning degradations.

functionality of the IP service

Basically, as stated previously the IP service (s_{ip}) provides two usage functionalities: generic IP connectivity $f_{\text{ip/use/con}}$ (between connected resources or to the outside IP world) as well as load balancing of a given list of resources ($f_{\text{ip/use/load\_balance}}$). Resources inter-connected by the IP service, i.e., mainly all resources used for realizing the e-mail service, the web hosting service, and most of their subservices - all high-level services (with respect to IP), e.g., $r_{\text{mailrelay1}}$, are subsumed under the term *IP-using resources* in the following. Additionally, also the most client end systems for the high level services use IP to communicate with their respective high-level service access points, e.g., clients for the mail service located within the MWN ($c_{\text{mailclient\_mwn}}$) and web service clients in the MWN ($c_{\text{webclient\_mwn}}$). The IP-using resources as well as these clients are subsumed under the term *IP endpoints* in the following. Having introduced these conventions, the general IP connectivity $f_{\text{ip/use/con}}$ between two IP endpoints can be in a more refined manner explicitly specified as $f_{\text{ip/use/con}}(\textit{path}= endpoint_1, \ldots, endpoint_2)$, e.g., $f_{\text{ip/use/con}}(\textit{path}=r_{\text{mailin\_tu}}, \ldots, c_{\text{webclient\_mwn}})$, expressing the IP connectivity along the path between the two particular IP endpoints.

resources of the IP service

Fig. 2.9 illustrates the resources and their inter-relationships for the IP service. The inter-connected resources, i.e., the IP endpoints, use network access points identified by network end point addressing parameters. In general, e.g., switch ports, MAC addresses, IP addresses, or hostnames of the connected resource can be used. But here, for reason of the purpose of simplicity, the IP endpoint name, i.e., resource name, itself (e.g., $r_{\text{mailrelay1}}$) is used throughout the thesis. So, network access points as a resource of the IP service are named $r_{\text{ipaccess}}(endpoint)$, e.g., $r_{\text{ipaccess}}(r_{\text{mailrelay1}})$.

For providing the IP functionality among the network access points and the outside IP world, various dedicated devices are used as resources of the IP service: various network routers ($r_{\text{rt\_xxx}}$), network switches ($r_{\text{sw\_xxx}}$), the specific load balancing switch ($r_{\text{lb\_sw}}$), and some devices for functionalities of the IP service used internally, i.e., a DHCP server (r_{dhcpsv}), a NAT server (r_{natsv}), a VPN server (r_{vpnsv}). The router $r_{\text{rt\_wan}}$ provides access to the IP world outside the MWN.

In addition to network access points and the dedicated network devices, network links are resources of the IP service. A network link between two network devices, or between a network device and an network access point is identified by both inter-connected resources: For instance, $r_{\text{iplink}}(r_{\text{rt\_core}}, r_{\text{rt\_lrz}})$ or $r_{\text{iplink}}(r_{\text{rt\_sw21}}, r_{\text{ipaccess}}(r_{\text{mailrelay1}}))$

Summarizing, the resources of the IP service are basically the IP access points

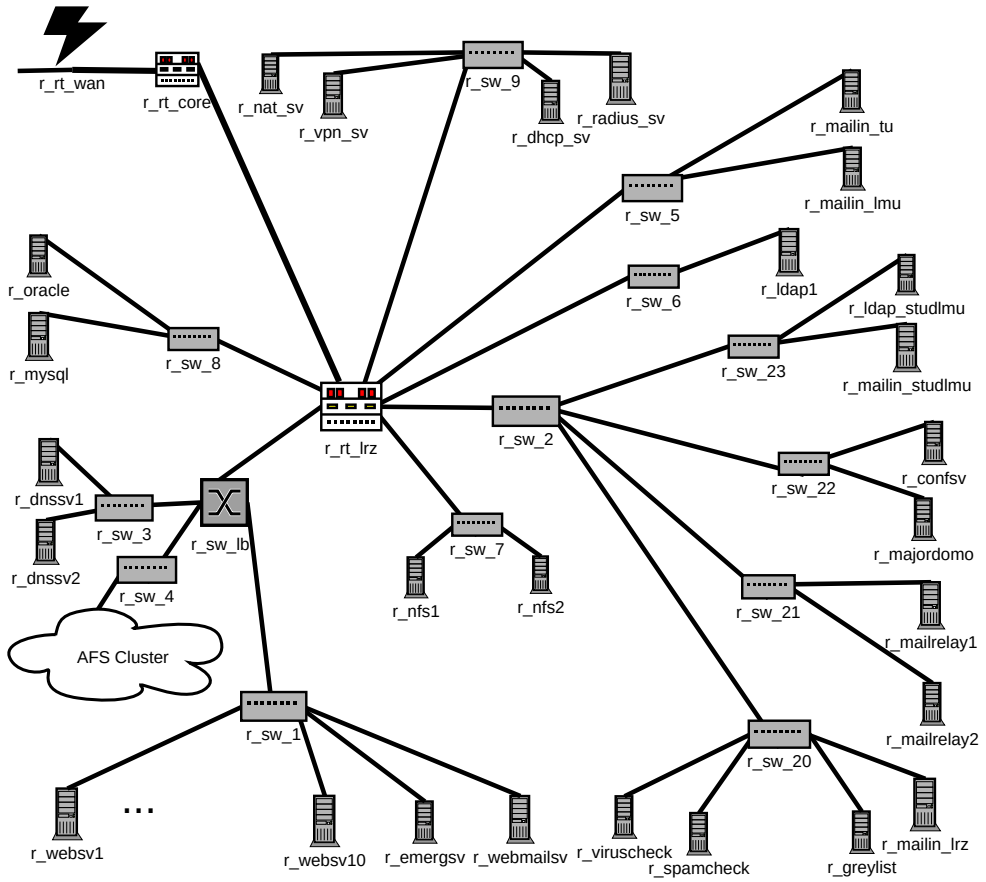

Figure 2.9: Resources and their inter-dependencies of the IP service

for IP endpoints, i.e., the resources of the other services, ($r_{\mathrm{ipaccess}}(endpoint)$), the network devices ($r_{\mathrm{rt\_...}}$), and the network links ($r_{\mathrm{iplink}}(from, to)$) interconnecting the former two ones.

inner dependencies and degradations for the IP service

In general, all these IP resources contribute for realizing, especially the generic IP connectivity $f_{\mathrm{ip/use/con}}$ already mentioned above. But having a more closer look, actually only a particular subset of them realizes the connectivity between specific two particular endpoints, namely the access points, routers, and links along the particular path between the IP endpoints, depending on the currently used specific routing. This means e.g., that the IP connectivity between the mail incoming server $r_{\mathrm{mailin\_tu}}$ and any normal mail client within the MWN ($c_{\mathrm{mailclient\_mwn}}$), $f_{\mathrm{ip/use/con}}(\textit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}})$, is depending on e.g., on IP link $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}})$, i.e., $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}}) \rightarrow f_{\mathrm{ip/use/con}}(\textit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}})$. Concerning Fig. 2.9, it is to note here that normal (mail) clients are located (with respect to IP) in IP subnets which are connected via the router $r_{\mathrm{rt\_core}}$ to the IP-using resources.

This means also that a degradation of $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}})$ leads (potentially) to a degradation of $f_{\mathrm{ip/use/con}}(\textit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}})$. For example can the complete IP link may become unavailable, which at least makes a re-routing necessary or leads also to total unavailability of the con-

nectivity functionality along that path. Moreover, the IP link may only have some partial degradation, e.g., suffering from a current high link utilization, caused by a high amount of foreign (with respect to the supported services, e.g., mail service) IP traffic. This resource degradation of $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}})$ will (potentially) lead to a low remaining IP throughput rate for $f_{\mathrm{ip/use/con}}(\mathit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}})$.

dependencies on high-level services and degradations thereof

Moreover, as the affected path between $r_{\mathrm{mailin\_tu}}$ and a potential mail client $c_{\mathrm{mailclient\_mwn}}$ is used for communication between them, respective mail service functionalities (for such a mail client) will be degraded, e.g., the mail box access (delay $q_{\mathrm{mail/delay\_mbox}}$ very high) and sending of mail (specifically dispatching the mail from client to server) (delay $q_{\mathrm{mail/delay\_send\_intra}}$, $q_{\mathrm{mail/delay\_send\_extra}}$) is slowed down. That is a degradation of an IP resource also causes a degradation a high-level service (s_{mail}) which uses the IP service as subservice, specifically the mail service functionalities $f_{\mathrm{mail/use/mbox\_access}}$ and $f_{\mathrm{mail/use/send}}$ (mail receiving is also affected but not considered here). The particular dependency covering this as introduced in Sect. 2.3.1 is simply the general one, $f_{\mathrm{ip/use/con}} \rightarrow f_{\mathrm{mail/use}}$. For this specific degradation example here, this dependency can be refined into $f_{\mathrm{ip/use/con}}(\mathit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}}) \rightarrow f_{\mathrm{mail/use/mbox\_access}}$ as well as $f_{\mathrm{ip/use/con}}(\mathit{path}{=}r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{mailclient\_mwn}}) \rightarrow f_{\mathrm{mail/use/send}}$. Concluding, this degradation example, illustrates how degradation of resources in a subservice can lead to degradations of the dependent service. Moreover, this example will be used further investigated as part of the next section.

further degradations of IP resources

There are many different possibilities for degradations of IP resources/IP connectivity functionality, e.g., high packet loss on a path because of a high error rate on an IP link of the path, or over-running packet queues on a network router on the IP path. Furthermore, the various degradations could be differentiated with respect to different protocols, e.g., low TCP throughput because of high amount of packet drops, UDP (User Datagram Protocol) failures due to high amount of packet drops.

2.3.4 Example run of an I/R analysis

Here, an example of a complete run of an I/R analysis will be presented. The example run is based on the LRZ scenario presented in the previous sections 2.3.1 and 2.3.2.

two example resource degradations

It is assumed that two independent resource degradations g_{r1} and g_{r2} are taking place simultaneously: an important network link ($r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}})$) connecting all mail servers to the outside world is suffering a high utilization (first degradation g_{r1}) and a complete outage of one of the storage devices (r_{afssv3}) is occurring (second degradation g_{r2}).

first resource degradation and impact on functionalities

The highly utilized network link's utilization has risen above 60%. This is slowing down e-mail sending and e-mail folder access. So the e-mail service s_{mail} in general, i.e., every customer of it, is affected. Nevertheless, this does not cause a complete outage, but only a severe performance impact, i.e., a high

network transport delay resulting in slow e-mail transfer ($q_{\text{mail/delay\_send\_intra}}$, $q_{\text{mail/delay\_send\_extra}}$, and $q_{\text{mail/delay\_mbox}}$). The functionality of the e-mail service in general is still available to users and customers.

second resource degradation and impact on functionalities

The storage outage g_{r2} (affecting parts of the AFS filesystem) is affecting parts of mail incoming folders ($f_{\text{mail/use/mbox\_access}}$) and a part of the hosted web sites ($f_{\text{web/use/apage}}$). So by this degradation not all service instances, i.e., customers are affected, but only a subset. However, for the affected subset of users and customers access to incoming mail functionality is completely impossible. One third of the mail accounts (2000) of LMU and one tenth (100) of the mail accounts of TUM (only staff mail boxes, not the students' ones) as well as all mail accounts for 3 of the 10 additional research institutions using the e-mail service (non-premium customers) become inaccessible through g_{r2}. Moreover, 10 of 300 web pages of LMU and TUM as well as the web pages for 5 from 20 other research institutions with hosted web pages become inaccessible.

impact on QoS

After identifying which services and which specific service instances (i.e., customers and users) are affected by the current resource degradations, the actual current values of QoS parameters for these services or services instances have to be checked. If these values are already available from some repository this one can be used. Otherwise measurements have to be done first. The resource degradation g_{r1} (link utilization $> 60\%$) is affecting the network delay resulting in an average e-mail sending delay which is about $5.5\ \text{min}$ (for all e-mail service instances). The second degradation g_{r2} will decrease availability and reliability of e-mail and web hosting as long as the degradation is not resolved (but only for affected service instances).

SLA violations

The current QoS values are compared to defined QoS parameter ranges and in doing so current and potential future QoS violations for affected service instances are found (compare SLA constraints and SLA penalty definitions in Table 2.6 and Table 2.11)

Resource degradation g_{r1} is affecting SLA constraints sla_cnstr_{mail3}: mail transfer delay too high (SLA states $> 5\ \text{min}$ as an SLA violation causing 300 € penalty costs per hour).

Whereas resource degradation g_{r2} is affecting SLA constraints sla_cnstr_{mail1} (concerning availability of e-mail service), sla_cnstr_{mail2} (concerning reliability of e-mail service), $sla\_prem\_cnstr_{\text{web1}}$, $sla\_norm\_cnstr_{\text{web1}}$ (concerning availability of web hosting service), and $sla\_prem\_cnstr_{\text{web2}}$, $sla\_norm\_cnstr_{\text{web2}}$ (concerning reliability of web hosting service), but only for the above mentioned subset of customers and users. The list given in Table 2.12 summarizes the identified SLA violations of g_{r2} (compare Table 2.6 and Table 2.11).

Afterwards, the identified QoS and SLA violations have to be mapped to financial impact development over time. In the first instance, this relates mainly to SLA violation costs directly related to these QoS violations. These impact will be the higher the longer the degradations are not resolved. So, the

mail service:

- **availability (**sla_cnstr_{mail1}**):** $q_{\text{mail/avail\_general}} > 99\%$; each 1% deviation below is causing a penalty of 500 € per affected premium customer.
- **reliability: (**sla_cnstr_{mail2}**):** $q_{\text{mail/reliab\_general}} < 30\ \text{min}$; each longer outage causes $(5000\ € + 10\ € \cdot duration\ /\text{min})$ as penalty per affected premium customer.

web hosting service:

- **premium customers:**
  - **availability (**$sla\_prem\_cnstr_{\text{web1}}$**):** $q_{\text{web/avail\_general}} > 95\%$; for each $0.1\ \%$ deviation a penalty of 100 € is required per customer.
  - **reliability (**$sla\_prem\_cnstr_{\text{web2}}$**):** $q_{\text{web/reliab\_general}} < 30\ \text{min}$; each longer outage requires to pay a penalty of 7000 € per affected customer.
- **normal customers:**
  - **availability (**$sla\_norm\_cnstr_{\text{web1}}$**):** $q_{\text{web/avail\_general}} > 90\%$; for each 5% deviation a penalty of 100 € is required per customer.
  - **reliability (**$sla\_norm\_cnstr_{\text{web2}}$**):** $q_{\text{web/reliab\_general}} < 1\ \text{h}$; each longer outage causing a penalty of 1000 €.
- **students:** best-effort

Table 2.12: SLA violations for the example I/RA run, caused by the resource degradation g_{r2}

$\lceil . \rceil : real \rightarrow int$, $\lceil x \rceil :=$ least integer which is equal or greater than x,
(for rounding purposes)

$(.)^+ : real \rightarrow real^+$, $x^+ := \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$
(to ignore negative values)

$\text{gt}(.,.) : real \times real \rightarrow \{0,1\}$, $\text{gt}(x,y) := \begin{cases} 1, & \text{if } x > y \\ 0, & \text{else} \end{cases}$
(to allow for comparison)

Table 2.13: Helper functions for specifying business impact of the example I/RA run

SLA penalty (slp) function (of duration of business time t) for business impact caused by g_{r1} (**mail sending delay**, sla_pnlty_{mail3}):

$slp_{\text{mail3}} : biz\_duration \rightarrow cost$
$slp_{\text{mail3}}(t) = 2 \cdot \lceil (t - 5\ \text{min})/1\ \text{h} \rceil^{+} \cdot 300\ €$
(the factor 2 takes into account both premium customers, LMU and TUM)

Table 2.14: SLA violation costs caused by degradation g_{r1}

financial business impact - SLA violation penalties

SLA penalty costs basically are represented as functions of duration of business hours (Mo-Fri, 8:00 - 18:00). For this purpose, the helper functions, defined in Table 2.13 are used. Based on this, Table 2.14 and Table 2.15 specify the financial business impact of g_{r1} and g_{r2}, respectively, in the form of SLA penalty (slp) functions of duration of business time t ($biz\_duration \rightarrow cost$) per SLA violation (slv): Corresponding to the involved SLA violation costs definitions, for g_{r1} one slp slp_{mail3} and for g_{r2} multiple slps slp_{mail1}, slp_{mail2}, slp_prem_{web1}, slp_prem_{web2}, slp_norm_{web1}, and slp_norm_{web1} are specified.

Finally, there is a function which computes the duration of business hours from an actual start time in a specific duration of real time: $biz\_duration(start, real\_duration) : time \times duration \rightarrow biz\_duration$. This can be composed out of individual SLA penalty functions mentioned above in order to compute the costs depending on actual start time and real time duration.

Putting these single SLA penalty cost functions altogether leads for both of the given resource degradation to an overall SLA penalty cost function (it is assumed that the current start time is here Tuesday, 10:00 o'clock, so that the next 8 hours are business hours):

$slp_{\text{gr1}}(.) = slp_{\text{mail3}}(.)$,

$slp_{\text{gr2}}(.) = slp_{\text{mail1}}(.) + slp_{\text{mail2}}(.) + slp\_prem_{\text{web1}}(.) + slp\_prem_{\text{web2}}(.) + slp\_norm_{\text{web1}}(.) + slp\_norm_{\text{web2}}(.)$.

evaluation and comparison of financial impact

So, it has to be determined whether, in what order, with how much effort, and in what time scale to resolve both resource degradations given. This relates directly to the development of repair costs over time. And the determined repair costs have to be compared to the financial impact development identified before.

Fig. 2.10 shows the graphs of slp_{gr1} and slp_{gr2} over time. By comparing both graphs it can be concluded, that the SLA penalties for g_{r2} after 20 minutes will start to grow quickly, whereas g_{r1} is causing penalties from the very beginning after 5 minutes but not so much increasing later on. So, if it is possible to fix g_{r1} very quickly (within 1-3 minutes), it would be desirable to do this first and fix g_{r2} afterwards as fast as possible. Otherwise, g_{r2} will have to be fixed with high priority first. In this case it is assumed that the high network link load (g_{r1}) cannot be fixed in a very short interval of only 3 minutes. Therefore,

SLA penalty (slp) functions (of duration of business time t) for business impact caused by g_{r2}:

$ratio_{index}$ is used per SLA violation as a factor to take into account partial unavailability in the sense, that a functionality is not functioning for all possible customers or users for which it should work.

mail availability (sla_pnlty_{mail1}):
$slp_{\text{mail1}} : biz\_duration \rightarrow cost$
$slp_{\text{mail1}}(t) = ratio_{mail} \cdot \lceil (t - 30\ \text{min})/30\ \text{min} \rceil^{+} \cdot 500\ €$
(in a normal business week (Mo-Fri, 8:00-18:00) there are $5 \cdot 10 = 50$ business hours, so 1% availability represents 30 minutes of business hours).

mail reliability (sla_pnlty_{mail2}):
$slp_{\text{mail2}} : biz\_duration \rightarrow cost$
$slp_{\text{mail2}}(t) = ratio_{mail} \cdot (\text{gt}(t, 15\ \text{min}) \cdot 5000\ € +$
$\lceil (t - 15\ \text{min})/1\ \text{min} \rceil^{+} \cdot 100\ €)$

web hosting availability for premium customers ($sla\_prem\_pnlty_{\text{web1}}$):
$slp\_prem_{\text{web1}} : biz\_duration \rightarrow cost$
$slp\_prem_{\text{web1}}(t) = ratio\_prem_{web} \cdot \lceil (t - 30\ \text{min})/30\ \text{min} \rceil^{+} \cdot 500\ €$
(in a normal business week (Mo-Fri, 8:00-18:00) there are $5 \cdot 10 = 50$ business hours, so 1% availability represents 30 minutes of business hours).

web hosting reliability for premium customers ($sla\_prem\_pnlty_{\text{web2}}$):
$slp\_prem_{\text{web2}} : biz\_duration \rightarrow cost$
$slp\_prem_{\text{web2}}(t) = ratio\_prem_{web} \cdot \text{gt}(t, 5\ \text{min}) \cdot 7000\ €$

web hosting availability for normal customers ($sla\_norm\_pnlty_{\text{web1}}$):
$slp\_norm_{\text{web1}} : biz\_duration \rightarrow cost$
$slp\_norm_{\text{web1}}(t) = ratio\_norm_{web} \cdot \lceil (t - 5\ \text{h})/2.5\ \text{h} \rceil^{+} \cdot 100\ €$
(in a normal business week (Mo-Fri, 8:00-18:00) there are $5 \cdot 10 = 50$ business hours, so 10% availability represent 5 hours and 5% represent 2.5 hours of business hours).

web hosting reliability for normal customers ($sla\_norm\_pnlty_{\text{web2}}$):
$slp\_norm_{\text{web2}} : biz\_duration \rightarrow cost$
$slp\_norm_{\text{web2}}(t) = ratio\_norm_{web} \cdot \text{gt}(t, 1\ \text{h}) \cdot 1000\ €$

$ratio_{index}$ has the following values (according to the before mentioned numbers of affected users):
$ratio_{mail,LMU} = 1/3$, $ratio_{mail,TUM} = 1/10$, i.e., $ratio_{mail} = 1/3 + 1/10$,
$ratio\_prem_{web} = 10/300 \cdot 2$, $ratio\_norm_{web} = 5$.

Table 2.15: SLA violation costs caused by degradation g_{r2}

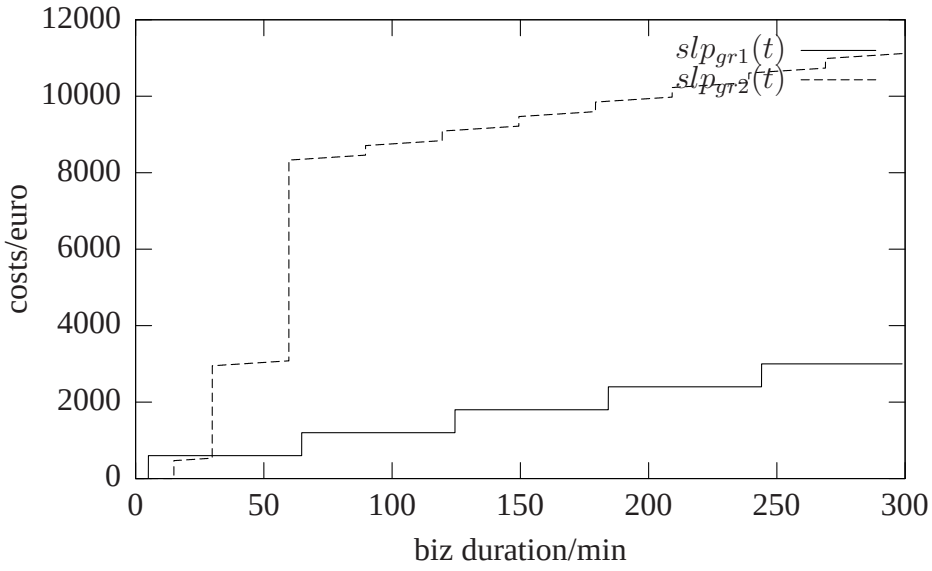

Figure 2.10: SLA penalty functions for resource degradations g_{r1} and g_{r2}

it is decided to fix degradation g_{r2} first with as much effort as possible and afterwards to fix degradation g_{r1}.

prioritization of resource degradations

As a result, it can be concluded, that the bad reliability of the web hosting service and the e-mail service is more important than the impact caused by the high link utilization. That is to say, resource degradation g_{r2} is more severe and more critical than resource degradation g_{r1}.

recovery options

Thus, it is recommended to repair the broken storage device first, immediately, with as much staff and expertise as available. One first recovery options for this is e.g., to try to reboot it. This may be combined or followed (in cause of failure to reboot) by checking whether a recent configuration change may have caused the problem. Another option is to install a backup device which will take much longer, tough.

Afterwards, it should be tried to cope with the high link utilization by e.g., re-routing particular IP traffic, prioritizing particular IP traffic on this link, or replacing the network link by one with more capacity.

For each of these different options for handling g_r1 and g_r2 an appropriate scheduling as well as a sufficient specification of all necessary parameters to determine each option in enough level of detail, concerning e.g., estimated duration and necessary effort, have to be identified and specified.

2.4 Requirements

In this section all relevant requirements are identified which a comprehensive framework for I/R analysis should fulfill. Very first, the generic requirements for I/R analysis already introduced and discussed in Sect. 1.2 are summarized in Sect. 2.4.1. Following, a general classification of more specific requirements is discussed in 2.4.2. Afterwards, these specific requirements are identified and presented in 2.4.3 to 2.4.7. Finally 2.4.8 provides a summary of the requirements.

2.4.1 Generic requirements

In the introduction, in Sect. 1.2, the discussion of deficiencies of today's approaches for I/R analysis was already concerned with generic requirements for I/R analysis. Actually these are: integration into existing service provisioning and service management environment, genericity concerning the service scenario, and manageability. They are listed and summarized in the following:

- (R0.1) smooth integration into the service provisioning and service management environment of the service provider: support I/RA by integrated access to all necessary exiting information
- (R0.2) genericity: applicability for any given IT service scenario, i.e., not depending on specific service technologies, service management technologies, or management architectures.
- (R0.3) manageability concerning changing service provisioning and service management environment: allowing for up-to-dateness and synchronization with a changing service infrastructure/service management infrastructure with a modest level of effort.

2.4.2 Specific requirements

In addition to the generic requirements of the previous section, more specific requirements can be identified which a framework for I/R analysis has to cover. By looking at the example run of an I/R analysis (see Sect. 2.3.4), larger different classes can be identified, in which these specific requirements for the I/R framework can be grouped. Fig. 2.11 illustrates a corresponding classification of specific requirements, which is identified and discussed in the following.

First, appropriate resource and service modeling including QoS parameter modeling is necessary. This includes proper modeling of all appearing dependencies, e.g., between services and resources or between services and subservices.

Second, the different types of degradations of resources, services and QoS parameters have to be modeled and appropriate mapping of these has to be possible. This is necessary for the notion of occurring resource degradations itself and to determine their impact on specific service instances (for a specific customer) or - more precisely - on specific service functionalities and QoS parameters of a service instance. This type of impact will be called service impact which is constituted by one or more service degradations.

Third, the service degradations have to be mapped on impact for the business of the provider, which comprises aspects such as financial and reputational impact. This mainly depends on the SLA violation costs which can be derived from the QoS degradations. But also the actual current usage of the degraded services might be considered here. Even further, customer satisfaction and public reputation can be considered here as influencing factors. All these impact factors will be subsumed under the term *business impact*.

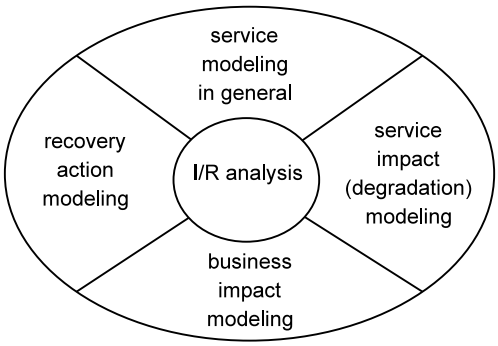

Figure 2.11: Classification of requirements

Fourth, for completeness, also the repair costs for the actual recovery realization have to be considered, and have to be combined with the previously determined impact. It is necessary to define the notion of a recovery action and to differentiate the various possible choices of recovery actions together with their associated costs, time range and actual effects on the service modeling (changes of the service modeling, e.g., making a resource redundant). This allows the selection of an appropriate costly, timely, and efficient recovery alternative.

Fifth, workflow modeling for the whole I/R analysis (as basically defined in Sect. 2.1) is needed, to define the steps, input and output artifacts, and conditions when to apply the analysis in which manner.

Summing up, all requirements can be grouped and classified into these areas: service modeling, service degradation modeling, business impact modeling, recovery action modeling, and workflow modeling and execution.

In the following the requirements of each area are analyzed in detail (see Sect. 2.4.3 to Sect. 2.4.7). Afterwards (Sect. 2.4.8) it is described how all the requirements are fulfilled by the framework.

2.4.3 Requirements on service modeling in general

Requirements of this class consider only the service modeling in general whereby no notion of resource/service degradations is introduced. This includes (static) aspects of the service concerning design, provisioning, and operating such as the description of functionality as well as dependencies from resources and subservices.

An important aspect is the number of different domains involved in the service provisioning. If multiple domains are involved, i.e., some services of a provider are based on subservices provided by another provider, it is necessary for complete, efficient and effective I/R analysis to be performed to have appropriate inter-domain management interfaces for exchange of impact and QoS degradation information for the regarded subservices. Otherwise, it would not be possible to include and map degradations of the subservices to degradations of services (or more specifically service instances) which are based on these subservices. A basis for the required inter-domain management interfaces is of course a common inter-domain service description for the subservices which effectively and efficiently allows to be combined with service descriptions of the dependent services.

Another relevant aspect of service modeling is the used granularity of the functionality definition: If the service description does not distinguish between separate service functionalities, e.g., sending e-mail, receiving e-mail, and various management functions for the e-mail service, the mapping of resource degradations to affected service instances and customers will be in many cases very general and unspecific, i.e., only stating that the whole service is affected, instead of determining which specific service functionalities are affected. Possibly one aspect of the functionality of a service is not as important or not as frequently used as another one. Consequently the decision to recover a service completely in case of an outage should be different depending on the actually affected functionalities.

The level of detail for the regarded service's description also plays an important role. The service description can focus only on the common perspective of provider and customer (cf. MNM service view, 2.2), it can focus only on the provider-internal realization of the service (cf. MNM realization view, 2.3), or it may include both of these perspectives. Of course, for a complete impact analysis starting with (low-level) resource degradations the realization view is necessary, because only here resources and their relationship amongst each other and to the services are visible. And to really map these resources degradations on specific (high-level) services, service instances and customers the service view is necessary, because only here the services/customers and especially their inter-relationships are visible. Moreover, in the realization view, different levels of abstraction for the realization of the functionality specified in the MNM service view can be distinguished, i.e., similar as specified in the instantiation methodology for MNM Service model [GHH+02] resources itself can be distinguished from the abstract functionality they provide within

the realization view: In contrast to the functionality in the service view agreed upon with the customer, the functionality in the realization view represents a (only provider-internally known) refinement of the former functionality, but is still more abstract than the specification of actual resources to lastly realize this refined functionality. For example, in the example scenario (Sect. 2.3.1) the mail receiving functionality $f_{\mathrm{mail/use/rcv}}$ (visible to customers, i.e., in service view) could be split into multiple such provider-internal functionalities, e.g., for blacklist checking, for spam checking, for IP address checking, for the proper mail receiving, the mail storing in the mailbox. Each of these provider-internal functionalities would be realized by the respective resources as described in Sect. 2.3.1, such as $r_{\mathrm{blacklist}}$, $r_{\mathrm{spamcheck}}$, r_{mailin}. Service scenarios are often (e.g., with MNM model's instantiation methodology) developed in the way described above, i.e., common design with customer (functionalities in service view), provider-internal design (refined functionalities in realization view), actual implementation (resources). This differentiation results in correspondingly refined dependencies for degraded resources/functionalities, and is therefore relevant for I/R analysis. Concluding, for a complete impact analysis, both views, namely realization view and service view, in appropriate levels of detail, have to be taken into account

Moreover, it can also be considered that the impact analysis is only done partially, that is only one/some of the abstraction levels discussed above:

- only for the service view: in case if not starting with known resource degradations, but instead directly with known degradations in one or more services.
- potentially only in different level of detail on the resource layer (realization view), as discussed above.

Another aspect are inter-relationships between specific dependencies: A service might independently be based on two different resources as an e-mail incoming server and an outgoing e-mail server which would be two separate dependencies without any special relationship between them. In case of a specific resource degradation of one of the two mentioned servers the I/R analysis could regard these two resources independently. A degradation of the mail incoming server will affect the e-mail receiving functionality, whereas a degradation of the other server will affect only the sending functionality. Often a service does depend on completely isolated different resources, but the dependencies might be (possibly mutually) related to each other: For example, each service instance of the web hosting service introduced above is based on two equally configured web servers on the one hand for performance reasons and on the other hand for reliability. Even more if both servers for a specific service instance, i.e., hosted web site, fail, one or two other of altogether 20 available web servers can quickly be configured to also take over the provisioning of the service for the affected service instances. All this is achieved by using two (also redundant) load-balancers which forward web site requests to the right web server. So for the web hosting example, the dependencies of the service from the two load balancers and the 20 web servers are all related

in some way. Such a complex situation as described above is often found in today's service realizations, such as redundancy for performance, reliability, or other purposes. In order to allow a combined processing during I/R analysis, a set of interrelated dependencies, each from a single resource/functionality (source) to a single resource/functionality (target), is more appropriately regarded as a single, more complex dependency with multiple dependent resources/functionalities as sources or as targets of the dependency. In UML (Unified Modeling Language) terminology this means that both multiplicities (for source or target of the dependency) can be greater than one. Concluding, I/R analysis should also be able to efficiently and effectively cope with dependencies between degraded resources/functionalities with multiplicities greater one.

The following list summarizes all identified requirements concerning service modeling in general, i.e., all necessary aspects which should be covered:

- (R1.1) number of domains:
  - single-domain (i.e., no consideration of provider-external subservices)
  - multi-domain (e.g., provider LRZ and external mail provider in the scenario of Sect. 2.3.1, i.e., consideration of provider-external subservices)
- (R1.2) granularity of functionality definition:
  - whole service (no distinction of separate service functionalities, e.g., as f_{mail} covering all functionality of the mail service in Sect. 2.3.1)
  - service functionalities of the whole service (e.g., separate functionalities $f_{\text{mail/use/send}}$, $f_{\text{mail/use/recv}}$, $f_{\text{mail/mgmt}}$ as a specialization of f_{mail} for the mail service in Sect. 2.3.1)
- (R1.3) level of detail regarding realization:
  - only service view (consider only functionalities and QoS parameters of a service)
  - only realization view (consider only resources and QoR parameters)
  - both views (consider functionalities and resources together with their QoR/QoS parameters, e.g., as for e-mail service in Sect. 2.3.1)
  - also differentiation of multiple levels of (resource) abstraction in realization view (e.g., resources differentiated from their (only provider-internally known) functionality inside of realization view, which is more refined than functionality (known to users/customers) in service view; compare above)
- (R1.4) dependencies with multiplicities ≥ 1 (describing the interrelationship of multiple dependencies between single resources/functionalities)

- 1 (single dependencies without specific inter-relationships between some of them)
- n (related dependencies: e.g., for explicit modeling/evaluation of redundancy/load-balancing distribution, such as the dependency $f_{\mathrm{dns/use}}, r_{\mathrm{mailrelay1}}, r_{\mathrm{mailrelay2}} \rightarrow f_{\mathrm{mail/use/recv}}$ of the e-mail service in Sect. 2.3.1)

2.4.4 Requirements on service degradation and quality modeling

This class contains all modeling requirements concerning degradations of resources and services. All different kind of service degradations are subsumed under the term *service impact*. In contrast to the previously described class, i.e., service modeling in general, it is concerned with the description and distinction of different degradations of resources and the actual mapping of these to services and service instances. Therefore it is concerned with the dynamic aspects of the occurring degradations, especially relevant to service fault management as well as service quality management, whereas the previous requirement class was concerned with mostly static aspects of service modeling relevant in general for service management.

An important issue are the dynamics of dependencies during the execution of an I/R analysis run: Some characteristics of the (statically) defined dependencies might change while an I/R analysis is performed. Therefore, I/R analysis has to consider possible changes of characteristics of the dependencies over time to determine the possible future impact. Such characteristics include validity of dependencies (e.g., resources might be replaced (temporarily) by others) or relationships between some dependencies (e.g., redundancy) might change.

Another important aspect to consider is the differentiation of separate types of resource degradations: The simplest solution is to distinguish only between “up” and “down” state of a resource or a service functionality. But, as the example I/R analysis reveals, also a more detailed (performance-like) point-of-view is necessary: The utilization of the highly utilized link cannot be handled in this simple way. One has to distinguish between complete outage of this link and (possibly) multiple utilization levels for the link, because different service (instances) might depend on this resource in different ways: i.e., different QoS parameters of different service (instances) might be depending on this resource, and therefore require to distinguish already on the resource level between different degradation steps to allow detailed and effective I/R analysis. Consequently, service impact comprises first (pure) functional impact, i.e., whether a service functionality is working and accessible or not, as well as QoS-specific impact, i.e., a more refined view taking into account different types of degradations of a functionality with specific QoS reduction, each with different potential QoS values/value ranges Similarly, for the degradations of resources, different types, i.e., different QoR/QoD parameters, with

different QoR/QoD values/value ranges, have to be considered in order to allow actually to derive functionality degradations in a such fine-grained level.

Of course the number of simultaneously regarded occurring or assumed resource degradations is an issue, too. If, for example, only one degradation is taking place, the recovery step of the I/R analysis has only to decide what specific effort in which time-range is necessary to recover from this single degradation. But if multiple current degradations are regarded, the recovery decision has first to decide which degradation to handle first or even to determine an order of recovery for all regarded degradations.

Following, a list summarizes the most important aspects which have to be covered by an effective I/R analysis:

- (R2.1) dynamics of dependencies per I/R analysis run:
  - static
  - dynamic: some dependencies might change during an analysis run; the change may be determined either according to an explicitly modeled pattern (e.g., dependent on the specific time of day additional servers are added in a redundancy cluster to handle an expected larger amount of requests) or determined by active measurement/active probing (e.g., determine which of two redundant servers is/was actually used at a specific point in time)
- (R2.2) number of degradation types per resource/functionality:
  - 1 (no particular QoR/QoS parameters distinguished) (e.g., in Sect. 2.3.1: complete unavailability of $f_{\mathrm{mail/use/rcv}}$)
  - n (different QoR/QoS parameters) (e.g., in Sect. 2.3.1: low availability, or high mail sending delay for $f_{\mathrm{mail/use/send}}$; low throughput, or high sending delay for $f_{\mathrm{mail/use/send}}$; random partial unavailability, or low bandwidth for $f_{\mathrm{mail/use/rcv}}$)
- (R2.3) number of different value levels/value ranges per degradation type of a resource/functionality:
  - 1 (“ somehow degraded or not”) (e.g., in Sect. 2.3.1: mail sending delay of $f_{\mathrm{mail/use/send}}$ somehow (unspecifically) degraded)
  - n (multiple QoR/QoS levels per degradation type) (e.g., in Sect. 2.3.1: mail sending delay of $f_{\mathrm{mail/use/send}} > 5\ \mathrm{min}$ which is more than $2\ \mathrm{min}$ above normal average)
- (R2.4) number of simultaneously occurring degradations:
  - 1 (here only determine impact and how fast, by what actions to recover from it)
  - n (e.g., as resource degradations g_{r1} and g_{r2} in the example run in Sect. 2.3.4; here additionally determine detailed importance/order/scheduling of every necessary recovery actions)

2.4.5 Requirements concerning the modeling of business impact

This class of requirements is concerned with factors which influence the financial and reputational impact of resource degradations on the business of the provider, altogether subsumed under the term business impact. No recovery or repair costs are considered here, but the impact over time is determined only based on the results of the occurring degradations.

In contrast to costs including repair costs for performing a selected recovery action the business impact considered here is more specifically called *pre-recovery business impact* in short.

First, regarding financial impact, SLA penalties have especially to be considered as the costs being directly caused by service degradations.

Second, the determined SLA costs should be combined with the actual current and expected future service usage to more accurately estimate all resulting costs.

Third, other resulting costs have to be considered (compare with Business Impact Analysis). Examples are:

- Revenue loss while a degradation takes place, e.g., because dynamically used/subscribed service is not used/subscribed by customers any more, i.e., depending on the expected future service usage.
- Customer satisfaction/public image (future financial impact because of customers terminating contracts or lacking of new customers).
- Public image in general (considering not only current, but potential customers in the future)
- (Regulatory costs, e.g., for not conforming to some legal obligations).

Each of them has to be quantifiable in an appropriate manner, or at least a clearly specified mapping from a qualitative specification to a quantifiable one has to be provided. These quantification specifications have to be defined according to the needs of the provider's business policies.

Following, a list summarizes the most impact aspects regarding business impact (by not performing any recovery) which have to be covered by an effective impact analysis: service usage to one item: other costs, because too complicated to

- (R3.1) SLA penalties, e.g., as the SLA penalties defined for the mail service in Table 2.6 or the web hosting service in Table 2.11, and their specific value development in the example run in Sect. 2.3.4.
- (R3.2) Actual current/future service usage (by users/customers).
- (R3.3) Further resulting (indirectly or directly) financial/reputational impact (if no recovery is performed), see above.

2.4.6 Requirements for recovery action modeling

Here, the recovery actions and associated costs are defined to correlate them with the business (financial/reputational) impact (over time; as a function of time or duration) identified in the previous phase.

First, the notion of recovery action has to be defined: This comprises the specification of count and granularity of different recovery actions to choose from and the association of each possible recovery action with related parameters to lastly determine the complete costs of the action. These related parameters comprise the following: Priorities, i.e., order of recovery for current existing degradations have to be identified, or more precisely a specific time plan for all recoveries to be set-up, personal and other extra effort of an action, and altogether the complete costs for the recovery action.

Second, combine with the previously determined business impact, i.e., for each current degradation find recovery alternative which is most efficient and costly regarding to needed time, (staff) effort, repair costs, i.e., resulting in minimum actual business impact being called *reduced business impact*.

- (R4.1) notion of recovery action alternatives:
  - (R4.1.1) granularity of recovery action notion (using only an enumeration of various alternatives, e.g., “do nothing”, “do normal (as regular) repair”, “perform repair with extraordinary effort”, “outsource the repair”, in contrast to also allowing an explicit parameterization for each of these alternatives allowing for a more fine-grained modeling)
  - (R4.1.2) determination of priority/order/scheduling (or more precisely appropriate point in time) of recovery for all current degradations (e.g., as in the example run of Sect. 2.3.4, handle degradation g_{r2} before g_{r1})
  - (R4.1.3) duration needed for each handling of a degradation
  - (R4.1.4) specific effort necessary per action (staff, extra resources)
  - (R4.1.5) repair costs determination per action
- (R4.2) combination with the previously determined business impact to determine reduced business impact

2.4.7 Requirements concerning the course of I/R analysis

This class of requirements is concerned with the course of action for I/R analysis, i.e., with workflow and process-related aspects of the I/R analysis.

In general, existing today’s IT management functional area classifications (e.g., FCAPS) as well as particular IT process frameworks (such as ITIL, eTOM) should be considered here to allow easy integration of the framework

with existing service provisioning and service management. Specifically, this comprises the following aspects:

First, which steps and tasks of I/R analysis should be regarded: only (pre-recovery) impact analysis (no recovery considered), recovery analysis, i.e., the identification and recommendation of an appropriate recovery action, and while tracking the actual recovery performed the customer should be kept informed as well as the modeling should be adapted if needed (feedback on modeling).

Second, the depth of the impact and recovery performed has to be considered. Roughly speaking, the targeted time-range of the recovery has to be determined: short-term (ITIL incident management) or long term (ITIL problem management).

Third, the actual urgency level has to be considered, too: The framework could be applied to real currently occurring degradations which have to be handled as fast and efficient as possible because they really cause impact now. Otherwise, the framework can also be applied in a simulation-like manner for assumed degradations to find out potential problems in the realization of the service and to check if for every considerable case an appropriate repair alternative can be found. Both types of application (for current or for assumed degradations) can be done with varying depth (or range) of analysis as described previously (incident or problem management).

Following, a list summarizes the most important aspects regarding the workflow of an I/R analysis in general which have to be covered by an effective I/R analysis:

- (R5.1) tasks of I/R analysis:
  - impact analysis
  - recovery analysis recommendation
  - recovery tracking
  - customer notification
  - modeling adaptation/feedback on modeling
- (R5.2) range/depth of I/R analysis (type of degradations covered)
  - actual occurring, current degradations: covering failure management, and performance management as far as concerned with current performance degradations
  - only assumed (possible future) degradations: simulation, covering further management areas, mainly maintenance action planning and availability management
- (R5.3) urgency level of I/R analysis
  - short-term: incident management (current degradations) and scheduling of forthcoming maintenance actions (assumed degradations)

– long-term: problem management (current degradations) and availability management (assumed degradations)

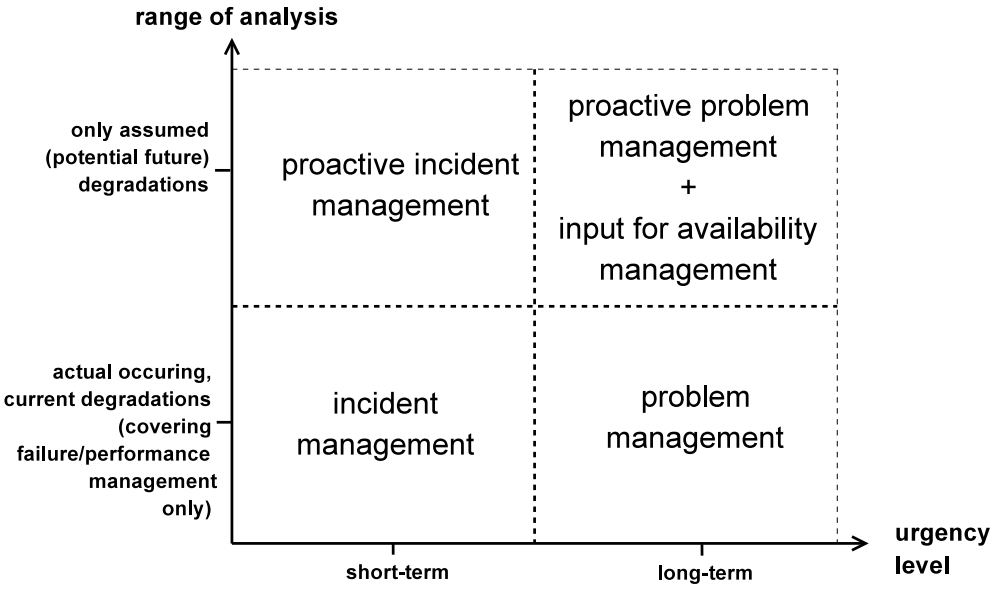

Figure 2.12: Different types of application of I/R analysis

While requirement R5.1 determines the different steps an I/R analysis run is consisting of, requirements R5.2 and R5.3 together distinguish between the various situations (in terms of management areas) which I/R analysis can be applied to. Fig. 2.12 illustrates this relationship between R5.2 and R5.3 and the different types of situations where I/R analysis can be applied to.

2.4.8 Summary of the requirements

This chapter was concerned with the identification of all necessary requirements for the framework. At first important terms were defined and the MNM Service Model was introduced. An example scenario comprising two services was presented by using the MNM Service Model and an example run of an I/R analysis for this scenario was shown. Afterwards, a general classification of requirements for the framework was given. Following, requirements were identified for each identified requirement class. Last, it was described how the iteratively developed framework will meet the requirements. Fig. 2.13 shows as overview an illustration for all these requirements (the specific ones, not the generic ones of Sect. 2.4.1) as dimensions of a star.

In Chapter 4 a generic framework for I/R analysis will be developed and discussed, which addresses all these requirements. Based on this, an instantiation methodology will be introduced in Chapter 5, which allows to instantiate this generic framework to a concrete service scenario concerning all requirements.

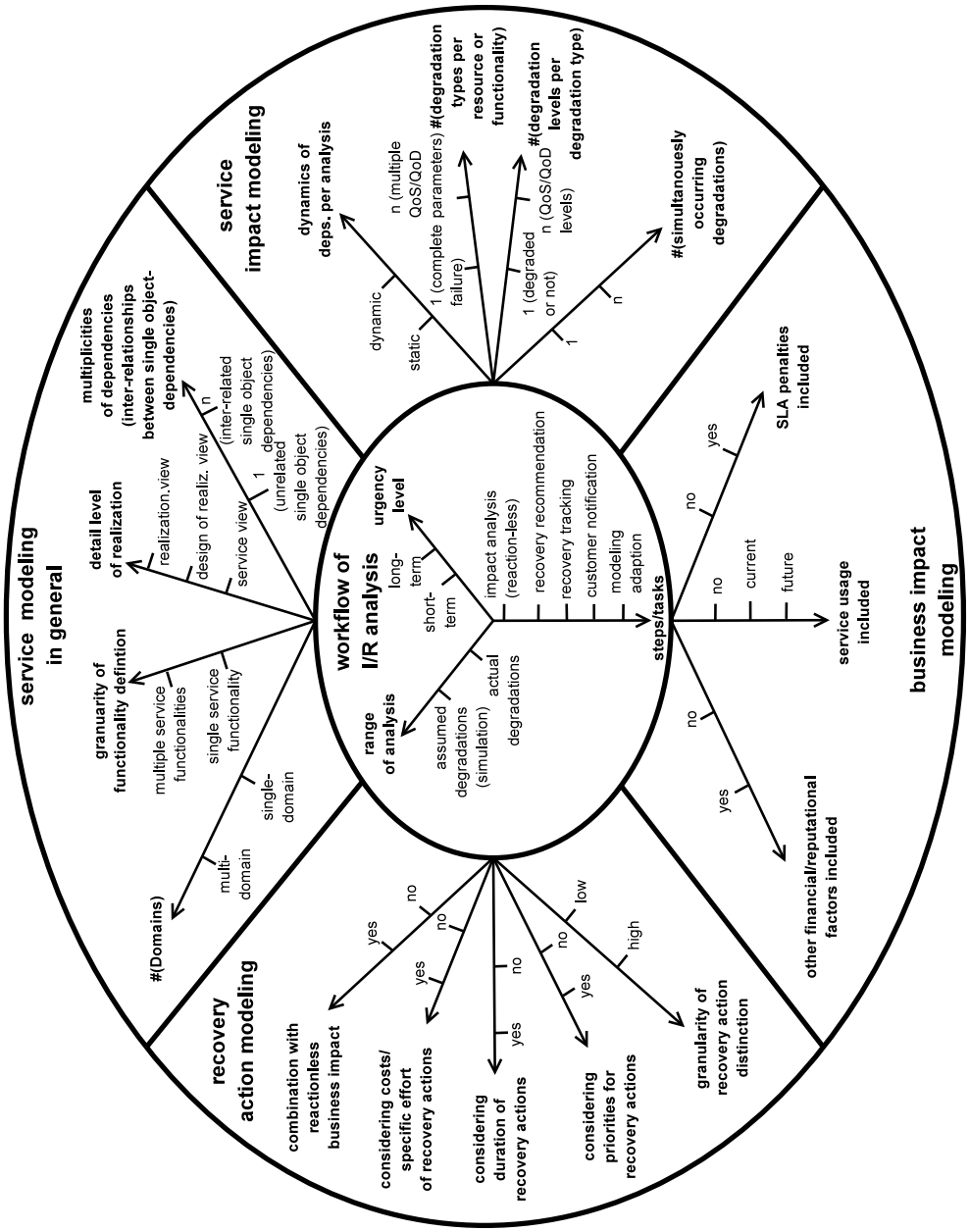

Figure 2.13: Requirements (refinement of Fig. 2.11)

Chapter 3

Related Work

Contents

In this chapter related work which is relevant for I/R analysis according to the requirements identified in Sect. 2 is discussed and evaluated. This encompasses products, standards, and approaches in research, each where appropriate.

existing approaches for I/RA

In Sect. 3.1 existing approaches, in this case mostly commercial products, which are concerned directly with I/R analysis today are introduced and their limitations shown.

related work for course of I/RA

Following, for each requirement class identified in Sect. 2.4 related work relevant for the requirements of that class is introduced and evaluated. First, concerning the requirement class R5 (course of I/R analysis) from a conceptual point of view, existing IT management process standards and their coverage of I/R analysis are analyzed in Sect. 3.2. In contrast, the implementation level of R5, i.e., techniques for the realization of the tasks of I/RA, is covered later in Sect. 3.6. But, in advance to this, from Sect. 3.3 to Sect. 3.5, for the four requirements classes R1 to R4, concerned with the modeling of resources and service functionalities and their dependencies (R1), of particular (quality) degradations and their dependencies (R2), of the business based on this (R3), and of the relationship to recovery actions (R4), subsequently related work is treated:

related work for service modeling

In Sect. 3.3 approaches for the modeling of services, their functionalities, and resources (R1) are treated. This further includes specifically approaches for the modeling of dependencies between resources and functionalities.

related work for degradation modeling

Based on this, in Sect. 3.4, approaches for the classification and modeling the (quality) degradations (and their dependencies) of functionalities and resources (R2) are treated next. This includes especially the specification, measurement and dependencies of quality parameters (QoR/QoS mapping).

related work for business impact and recovery modeling

In Sect. 3.5 approaches regarding the relationship of degradations of service functionalities and resources to the business (R3) and recovery selection (R4) are discussed. This includes approaches for SLA modeling, as well as methods for the prediction of future service usage by extrapolation techniques.

techniques for reasoning about dependencies

As already mentioned above, R5 from implementation point of view, i.e., techniques for the realization of the tasks of I/RA (R5.1), are discussed in Sect. 3.6. There, specifically potential techniques for reasoning about the modeled data in general (R1 to R4), i.e., mainly about degradations and their dependencies, are treated.

assessment

At last, in Sect. 3.7 an overall assessment of the related work introduced before is performed, specifically according to the requirements of the respective requirement class.

3.1 Existing Approaches for I/R Analysis

Following, existing approaches, mainly commercial products, targeting towards impact and/or recovery analysis are discussed and their limitations regarding the requirements of Sect. 2.4 are analyzed.

Current Practice Usually, as already discussed in Sect. 1.2, the current practice in today's service fault management is that impact and recovery analysis is mostly done by hand using detailed, but undocumented expert knowledge and corresponding best practices. Especially an automated and integrated decision support, as well as an automated method for the tracking of an on-going or completed recovery with respect to the reached reduction of business impact, is missing.

Research There are some research approaches for I/RA, like [SS06], but mostly they are limited to specific types of scenarios or technologies used, For example, the approach of [SS06] is limited to IP telephony.

Commercial Products Also commercial products have been developed in order to be used for impact/recovery analysis. Examples of them are presented in the following.

specifically concerned with recovery recommendation only

The first commercial product, SpectroRX [Apr, Apr04], being part of the Spectrum Suite from Computer Associates (formerly from Concord, Aprisma, and originally Cabletron System), is specifically concerned only with recovery recommendation, therefore not explicitly with the impact analysis. It uses a Case Based Reasoning (CBR, see Sect. 3.6.3) approach for the resolution of problems. It can be used within the Spectrum suite or standalone for enhancing an existing 3rd-party trouble ticket system for problem resolution.

Its problem resolution method using CBR might be interesting, but unfortunately its is proprietary and details about it are not publicly available. Impact analysis is only covered implicitly by performing the problem resolution based on past problem solutions (CBR), i.e., by reusing a past recommended resolution of a similar solution and thereby also reusing implicitly the past impact analysis done to identify this resolution. Above all, SpectroRX seems not to have an explicit instantiation methodology for the application to concrete scenarios.

non-open, proprietary service dependency models

There are also examples of commercial tools which are concerned with both impact analysis and recovery recommendation. Examples are Netcool/Impact [Net, Net04] being now part of the Tivoli Suite from IBM (formerly from Micromuse), Service Navigator from HP [HP], and Smarts Business Impact Manager from EMC (formerly InCharge from Smarts) [Sma, Sma04].

They all have some modeling of dependencies between resources and services, but these models and how they are actually used for I/RA are proprietary. As far as public information is available, the following can be said. Generally these model all share the same limitations: Concerning the granularity of functionality definition (requirement R1.2), their models are based only on dependencies between services as a whole - as far as information is publicly available. Refinement and decomposition into different functionalities of a service, potentially also to particular sets of instantiations of such functionalities (such as $f_{\text{auth/use}}$, and $f_{\text{mail/use/send}}(\textit{authentication}=\text{yes})$ in the example scenario of Sect. 2.3.1) and correspondingly refined dependencies (such as $f_{\text{auth/use}} \rightarrow f_{\text{mail/use/send}}(\textit{authentication}=\text{yes})$ in Sect. 2.3.1) are not covered. Aspects of $m : n$ dependencies and related aspects as redundancy and load-balancing are partially covered by the possibility to somehow aggregate status information along the service dependency models. But further temporal aspects, i.e., dynamics over time (requirement R2.1), seem not to be covered. Whether multiple QoR/QoS parameters per resource/service, i.e., also along the respective dependencies with a corresponding mapping of them, are supported is not clear from what is publicly available. What notions of SLA, of business degradations (R3) and furthermore of recovery actions (R4) and of corresponding recovery plans are utilized is not publicly available. Above all, also none of these products - as far as public information is available - has a top-down-oriented instantiation methodology for application to a concrete scenario, and if so it is obviously proprietary. Furthermore, the coverage of the generic requirement R0.3, manageability in case of updates, may be insufficient: It is unclear how difficult it is for experts to modify the modeling in case of changes on the service environment, or how far is the degree of automation concerning this.

potential basis of an implementation

Nonetheless, each of these tools may be adapted to serve as an implementation platform for I/RA after an appropriate, consistent and open modeling concerning R1 to R4 has been established, as part of the development of an I/RA framework. For instance, IBM Netcool/Impact provides access to many other management data sources and to many other management tools. Moreover, it includes a policy language similar to an imperative programming language in order to process events and combine them with other data sources and trigger appropriate actions in the environment.

Assessment Concluding, it can be said that all discussed existing approaches, in general share the following limitations concerning I/RA:

- lack of genericity, or top-down-orientation; above all for most products even no open specification exists about how impact/recovery analysis is actually performed, or
- insufficient (or at least non-open, proprietary only) possibility to specify dependencies between service functionalities, their degradations, and potential recovery alternatives, in appropriate granularity and accuracy (R1 to R4), e.g., not explicity supporting detailed $m : n$ dependencies,

modeling of redundancy, dependencies of quality parameters, time dependencies, or

- lack of a top-down oriented instantiation methodology (covering also the instantiation of the preceding item), which allows a service provider to apply the particular approach to his concrete service scenario in a consistent way.

Consequently, it is really necessary to develop an integrated framework, including an instantiation methodology, which consistently meets all requirements of the classes R1 to R5.

3.2 Related Work Concerning the Course of the I/R Analysis Conceptually

In the following, approaches are discussed which are relevant to the course of I/RA (requirement class R5) from a conceptual point of view. In contrast, related work for the implementation level of the course of I/RA will be treated in Sect. 3.6 instead.

basic environment for I/RA

Integrated Service Management A framework for integrated service management was proposed in [DR02, DR03]. As not being specifically concerned with I/RA, it naturally cannot provide a detailed description of the tasks of I/RA (R5.1). However, it provides a generic, overall framework, in which the I/RA framework can be embedded: A component architecture being developed for the tasks of I/RA (R5.1) may be integrated in this generic service management framework, which would facilitate e.g., the access to the various pieces of management information being necessary for I/RA. That is why, the integration concerning the modeling requirement classes, mainly R1 and R2, is shortly discussed here. Concerning the first modeling requirement class, R1 (service modeling), the following can be said: The granularity of functionality definition and corresponding dependencies thereof (requirement R1.2) is not fully covered, e.g., with respect to dependencies like $f_{\mathrm{auth/use}} \rightarrow f_{\mathrm{mail/use/send}}(\textit{authentication}\text{=yes})$, $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw\_2}}) \rightarrow f_{\mathrm{ip/use/con}}(\textit{path} = r_{\mathrm{mailin\_tu}}, \ldots, c_{\mathrm{webclient\_mwn}})$, $r_{\mathrm{websv(x)}}(\textit{configuration} = \text{special}) \rightarrow f_{\mathrm{web/use/apage\_special/mysqconf}}$, and $r_{\mathrm{mailin\_lmu}} \rightarrow f_{\mathrm{mail/use/mbox\_access}}(\textit{user} \in \text{LMU}, \notin \text{stud})$, in the example scenario of Sect. 2.3. $m : n$ dependencies and their relationship to details of redundancy/load-balancing is to some degree approached by so-called OR dependencies within the QoS specification language Qual. The extension towards the second modeling requirement class, R2 (quality modeling), is partially covered: On the one hand, with the QoS specification language Qual QoR/QoS parameters of services can be specified as well as the detailed dependencies between them and between their values (R2.2, R2.3). However, on the other hand, the specification and processing of dynamics in/over time of dependencies (R2.1) regarding dependencies of services or in more detail of their QoS parameters and their values, is not addressed explicitly. Concluding, the general modeling of the integrated service modeling itself cannot fully provide a modeling concerning all requirements of R1 and R2. But, parts of its modeling can be reused for I/RA, especially the generic QoS language Qual for the specification of the mapping between QoS parameters can be used for I/RA to describe dependencies between QoS parameter (values) in general (compare Sect. 3.4).

specification for dependencies of quality parameters

Customer Service Management The *Customer Service Management (CSM)* approach [LLN98, Lan01, Ner01], being related to the MNM ser-

vice model (Sect. 2.2), proposes an integrated and unique interface between a service provider and its customers for all management related tasks. This interface, the CSM access point (compare definitions in Sect. 2.1), allows the customer, regarding all management areas (FCAPS), to fetch management information about his subscribed services in an integrated and consistent manner, as well as to control and manage the subscribed services in an integrated way, to the extent as it is agreed with the provider.

assessment

Consequently, the CSM approach may also be used in order to realize the interface to the customer for the I/RA task *customer notification* of requirement R5.1. Indeed, [Ner01], already comprises an extensive classification and subdivision of management interactions between customer and provider for the CSM access point. Specifically, the interaction class *Problem Management: Status inquiry of a problem report* in [Ner01] (compare Table 3.1) is the one which has to be refined in order to specify the detail of the I/RA task customer notification. This interaction class also comprises the aspect of exchanging degradation information of provider-external subservices in order to use them for I/RA regarding services based on these subservices.

In addition to providing a basis for the I/RA task customer notification, especially the above mentioned CSM interaction classification of [Ner01] (Table 3.1), which represents a refinement of the management functionality classification of the MNM service model, can be used as a basis for modeling of service (management) functionalities of a concrete service scenario. That is why this classification is specifically treated in Sect. 3.3.1 as an extension to the MNM service model.

IT Process Management Frameworks In the following two sections, the established IT process management frameworks IT Infrastructure Library (ITIL) and Enhanced Telecom Operations Map (eTOM) are examined, regarding the coverage of the requirement class R5, especially R5.1 concerning the tasks of I/RA.

Further standards, but not being specifically concerned with the details of IT management, instead being more concerned with high-level financial aspects, are *Common Objectives for Information and related Technology (Cobit)* [COB] and Balanced Scorecard [KN96].

3.2.1 IT Infrastructure Library

The *IT Infrastructure Library (ITIL)* is a collection of best practices for IT processes in the area of IT service management. It is provided by the *British Office of Government Commerce (OGC)* and the *IT Service Management Forum (itSMF)* [itS].

In ITIL version 2 (ITIL v2) [ITI, Off99, Off00, Off01, Off02b, Off02a, Off02c, Off03, Off04], service management is subdivided into 11 modules. These modules are grouped into Service Support Set, which comprises provi-

der internal processes, and Service Delivery Set, which comprises processes at the customer-provider interface. In each module processes, functions, roles, and responsibilities as well as necessary databases and interfaces are specified. Generally, ITIL specifies contents and processes with the aim of high level of abstraction. Consequently, it is not concerned with particular management architectures or tools.

Particularly, service fault management is specified by the modules Incident Management process and Problem Management process within the Service Support. Moreover, there is the Availability Management process being part of the Service Delivery. The *Incident Management* is concerned with the fast, short term solution of newly occurring failures and degradations in order to ensure a continuity of the services. It uses the *service desk* as interface to customers, e.g., for receiving reports about new incidents. Severe failures and degradations which cannot be solved or only temporarily are transferred as structured queries to the Problem Management for finding a final, long-term solution. The purpose of the *Problem Management* is to solve given, known problems. This further includes taking care of keeping priorities, minimizing the reoccurrence of problems, and the provisioning of corresponding management information. After receiving requests from the Incident Management, problems have to be really identified (complete root cause analysis) and found information about final recovery actions is transferred to the Change Management, which is another module of ITIL. The *Availability Management* has the goal to sustain and ensure the availability of all provided IT services in order to support the business at justifiable costs, over a period of time.

Incident Management

Problem Management

Availability Management

only abstract recommendations

Regarding the processes, ITIL specifies only what has to be done in a general manner. Specifically, concerning I/RA or above all its particular tasks (requirement 5.1), ITIL does not provide in detail descriptions of steps or activities which could be done in order to perform I/RA or one of its particular tasks. Above all ITIL generally includes no information how the ITIL processes can actually be performed, i.e., by which actual techniques or tools. It includes only no particular definition of degradation and impact, which can be used for a detailed, recursive calculating of business impact, by using particular dependencies between resources/service functionalities and their degradations: It is only recommended that the important business impact has to be identified according to the needs of the business of the organization, and that an impact analysis (e.g., entailed from resource degradations) has to be accordingly oriented towards a minimizing of such business impact in order to support the business as most as possible. Concluding, an impact analysis according to ITIL should be business-oriented, which may include customer-orientation depending on the definition of the business goals.

In 2007 a new, extensively revised version of the ITIL standard, ITIL version 3 [Off07d, Off07b, Off07e, Off07c, Off07a], has been published. This version contains much more processes than earlier ones aiming at the coverage of the whole service life cycle. Nevertheless, the processes related to I/RA (see above) are still in place and their recommendations concerning I/RA, in

terms of workflow/process descriptions as well as of service models, and impact/degradation definition, are still as vague as before. Consequently, the new version does not provide additional benefit compared with the previous version for the development of the I/RA framework in this thesis.

assessment

Consequently, the recommendations of ITIL can only give some hints about what information has to be considered for the derivation of business impact from resource degradations: Especially that an impact analysis has to be oriented towards business-orientation and customer-orientation. Neither a detailed and explicit description of I/RA and its tasks can be derived from the ITIL specification, nor can the abstract modeling of the ITIL specification be reused as a basis for an integrated modeling of I/RA with respect to the requirement classes R1 to R4. Nevertheless, after having been developed a detailed specification of the tasks of I/RA as part of the development of the I/RA framework, and having a corresponding detailed modeling regarding R1 to R4 in place, this may be integrated with a concrete existing ITIL-conforming infrastructure, being utilized in a concrete service scenario.

3.2.2 Enhanced Telecom Operations Map

The *Enhanced Telecom Operations Map (eTOM)* [GB 05] is maintained and standardized by the *TeleManagement Forum (TMF)* [eTO]. The latter is an international organization of service providers and suppliers in the area of telecommunications services. Similar to ITIL (Sect. 3.2.1), eTOM is a process-oriented framework. But originally it was designed for a narrower focus, i.e., the market of information and communications service providers.

Basically, eTOM is a process classification including a description for each process contained. This classification is based on a horizontal grouping into processes for customer care, service development & operations, network & systems management, and partner/supplier. In addition, there is also a vertical grouping, distinguishing fulfillment, assurance, billing, in order to reflect the service life cycle.

Problem Handling

Concerning fault management three processes have been defined along the horizontal process grouping: Problem Handling, Service Problem Management, and Resource Trouble Management. The process *Problem Handling* is concerned with receiving trouble reports from customers and with solving them by using the Service Problem Management. Moreover, it provides information to the respective customer about the current status of the trouble report processing as well as about the general network status, including planned maintenance. In addition to that, Problem Handling has to inform the QoS/SLA management about the impact of current errors on the SLAs.

Service Problem Management

In the process *Service Problem Management* reports about service failures affecting customers are received from Problem Handling and appropriately transformed. A task is further to identify the root causes and to identify and realize a problem solution or a temporary workaround.

Resource Trouble Management

The process *Resource Trouble Management* is concerned with related tasks on the resource layer: It is responsible for resource failure event analysis, alarm correlation & filtering, and failure event detection & reporting. Moreover it has to execute different tests to identify resource failures. Furthermore, it is tracking the status of the processing of the trouble reports, similarly to the functionality of a trouble ticket system.

process descriptions very rough

Although the process classification in eTOM distinguishes a large number of processes, the description provided for each process is not very detailed. Generally, eTOM is useful as a check list summarizing what aspects for the management processes have to be taken into account. Moreover, there is no methodology for applying it to a concrete scenario.

assessment

Concluding, eTOM provides the possibility to basically classify I/R analysis and its task (R5.1). It can give hints about what to perform in general in these tasks. But, as a detailed description for these tasks cannot be derived from eTOM, similarly as with ITIL (see Sect. 3.2.1), a detailed, integrated workflow modeling for the tasks of I/RA has to be performed as part of the development of the I/RA framework. This will be actually based on some previous work already described in [HSS05a, HSS05b], which can be used as a basic starting point.

3.3 Related Work for the Service Modeling in General

Here approaches are discussed being relevant to the requirement class R1, i.e., concerning the modeling of service in general. Generally, R1 is concerned with the modeling of resources and service functionalities, as well as with the modeling of dependencies between resources and service functionalities, in appropriate granularity.

terms used

In Sect. 2.1 particular terms, concerning also the modeling of resources and service functionalities in general, have been introduced. Specifically, for the term *service* which is defined in various ways today, the particular definition of Sect. 2.1, being consistent with the MNM service model (Sect. 2.2), is used in the following.

Modeling of redundancy and load-balancing The requirement R1.4, being concerned with $m : n$ dependencies between resources and service functionalities, is related to aspects like *redundancy* and *load-balancing*. For accurate calculation of impact by I/RA, a detailed modeling of the connected $m : n$ dependency and the details of the related redundancy/load-balancing will be necessary.

Both, redundancy and load-balancing are mainly applied to resources, while they in principle can also be applied to functionalities. Normally, both may be applied for the purpose of performance enhancement and/or reliability enhancement.

Load-balancing can be classified according to the distribution method it uses. Typical example of the distribution method for load-balancing are:

- round-robin (e.g., DNS round robin)
- random
- hash-based (i.e., distinguished by client address)
- least resource usage

Load-balancing is widely applied for performance enhancement, but depending on the distribution method can be also be used for reliability enhancement.

A modeling of dependencies suitable for I/RA will have to include all aspects of $m : n$ dependencies and the related redundancy/load-balancing as necessary for I/RA: For example, in case a round-robin distribution method is used for two load-balanced resources, and one of two resources is suffering a complete outage, on average, only each second resource access will fail, as the other resource is still left working. That is, without a coverage of these details of a $m : n$ dependency by the service modeling, I/RA will fail to accurately

derive the resulting impact of resource degradations in appropriate granularity. The above introduction about redundancy and specifically load-balancing provides a rough basis of what has to be supported for by an appropriate modeling, e.g., in providing a potential, basic classification of load-balancing distribution methods.

UML The *Unified Modeling Language (UML)* [RJB98, OMG07a, OMG07b] is an object-oriented, generic, universal modeling language, which is widely used today for Software development as well as for specification tasks in general. It is also used as basis for various approaches for service modeling including some of them discussed in the following.

As UML is a totally generic, object-oriented modeling language, it can in any case be a suitable basis for the modeling of services regarding the requirement class R1, possibly also for the modeling of the requirement classes R2 to R4.

To cover different aspects of modeling, there are different types of UML diagrams, e.g., class and object diagrams, use case diagrams, activity diagrams, interaction diagrams. Particularly activity diagrams provide a convenient way for workflow modeling. So, this type of diagram can be used for describing all tasks of I/RA as appropriately refined workflows.

dependencies in UML

The modeling in UML is based on a meta-model [OMG07a, OMG07b]. In this UML meta model, the concept of a dependency is basically introduced as a *generic relationship* between its *related (meta) objects*. Furthermore, the UML meta model explicitly introduces a refined kind of relationship, the *directed relationship*, which subdivides its *related (meta) objects* specifically into *sources* and *targets*, the latter ones being dependent on the former ones. This notion of a of directed relationship is in UML mainly used as a general meta model concept, generalizing abstract modeling relationships like e.g., the refinement from an interface to its implementation. Nonetheless, in this thesis similar terms concerned with directed relationships as the ones introduced above will be used throughout this thesis: *related objects* or more precisely *dependent objects* of a *dependency*, and more specifically the *sources* or *source objects*, as well as *targets* or *target objects* of a *dependency*. Instead of describing generic meta model concepts as in UML, they will being used to describe dependencies between resources, service functionalities, their degradations, and their potential recovery, that is for modeling dependencies of objects concerning R1 to R4.

Internet Information Model The *Internet Information Model* was designed by the *Internet Engineering Task Force (IETF)* [CMRW96] for the specification of management information in the IP world, and is widely used today. The management information is stored in a so-called *Management Information Base (MIB)*. For the exchange of the management information usually the *Simple Network Management Protocol (SNMP)* also proposed by the IETF is used. A MIB comprises a set of MIB variables. Such MIB vari-

ables are used to describe particular aspects of the management for IP devices and IP networks. Moreover these MIB variables are hierarchically organized in a common Internet registration tree. Different extension parts of this tree are administered by different organizations, such as basically by the IETF itself or by other standardization organization for integrating new standards, or by commercial vendors. Concerning the latter, a large set of vendor-specific MIB variables exists in so-called *enterprise MIBs*.

simplistic modeling approach

The modeling paradigm of the Internet Information Model is quite simplistic as it supports no object-orientation (especially no inheritance) or above all not the definition of complex containment structures. The only organization of the model is provided by the Internet registration tree which is usually not modified by normal customer/users but only by the above mentioned administrating organizations. Moreover, the simplistic approach entails that related information parts with a common aspect, e.g., all information concerning the interfaces of a IP router, is scattered over the whole registration tree.

no real extension for modeling of service dependencies

Above all, most extensions for the registration tree are concerned with technology specific or at least resource-oriented aspects. Despite of a few efforts to integrate service-related information such as [HKS99], the model is only focused on resource management. Therefore, the management information made available by the Internet Management Model today is not providing any real basis for the modeling dependencies of service functionalities, their degradations, and their potential recovery, i.e., concerning R1, and the extension towards R2 to R4.

The *Open Systems Interconnection (OSI)* [OSI92] was a standardization effort for networking in general. It comprises an object-oriented management information model which is highly superior to the Internet Information Model treated above, e.g., by including modeling features such as inheritance and complex containment relationships. However, the OSI approach and especially its information model was never widely accepted and used, especially not for the purpose of service-oriented management as this term is regarded today.

Particular approaches for service modeling The approaches for IT Service Modeling which are specifically investigated concerning service modeling are the MNM service model, CIM, ITIL CMDB, SID, and the Service MIB approach. These are treated in detail from Sect. 3.3.1 to Sect. 3.3.5 respectively. The MNM service model was already basically introduced in Sect. 2.2, but in Sect. 3.3.1 further aspects and extensions are treated. As the analysis will show, none of these approaches fully and explicitly covers all requirements of R1. But, an adaption of one of them or a combination of multiple ones may be designed to fulfill all the requirements of R1. For such an adaption, an integrated modeling for all aspects of the requirements of R1, being compatible with the other modeling requirement classes R2 to R4 will have to be developed as part of the I/RA framework.

In the following, research approaches for the specification, and, as it is often

related to the former, the finding of dependencies between resources, service functionalities, and their degradations are presented.

Research concerning dependency modeling The specification of dependencies, namely of resources and service functionalities, their particular degradations and their potential recovery, is crucial for I/RA (R1 to R4).

But, despite of the importance of dependencies for fault diagnosis in general and other management tasks, dependencies and their features are most often described only superficially by the various existing approaches and standards concerned with service modeling.

dependencies in CIM

The CIM model (Common Information Model), being treated in greater detail in Sect. 3.3.2, is one of the few existing standards which is explicitly concerned with the modeling of dependencies, yet without covering the details. The CIM core model [GRM97] comprises a generic dependency class, similar as the UML meta model (see p. 80), from which other classes can inherit. However, the dependencies include nearly no particular attributes and are above all not concerned with services in the meaning of I/RA framework. Dependency graphs based on CIM dependencies are used in [AAG+04a] for problem determination. A particular extension of CIM for the purpose of fault diagnosis and impact analysis of telecommunication services is addressed in [SS06].

dependency graphs

In general, dependency graphs are a common concept to model dependencies for the purpose of fault diagnosis. Particularly, [Gru98, Gru99] presents a generic approach for such graphs. But, the dependencies themselves and their particular features are not further addressed. In [KK01] dependency models are basically categorized into *functional*, *structural*, and *operational* models. Furthermore, [KK01] discusses the acyclic nature of dependency graphs as one of their important aspects: Mutual dependencies on the service level usually indicate a bad design.

using XML

[EK02] proposes an approach to manage service dependencies particularly with *XML (eXtensible Markup Language)*, defining a resource description framework. Moreover, a further approach using XML based on World Wide Web Consortium's (W3C) Resource Description Framework [RDF] is concerned with the modeling of dependencies. This approach includes only examples of potential attributes for dependencies, such as the strength (likelihood that a target component is affected if source component fails), the criticality regarding the goals of the organization, and the degree of formalization (how difficult it is to determine the dependency). A subsequent paper [EK02] discusses challenges of dependency graphs, e.g., the distribution of the the graphs, missing information, and efficient queries.

dependencies for ISPs

In [CR99] the dependencies for services offered by *Internet Service Providers (ISPs)* are classified in the following way: An *execution dependency* denotes the relationship of the performance of an application server process and the status of the host. A *link dependency* denotes the relationship of the service

performance and the status of network links. Moreover, a *component dependency* describes the fact, that, in case of an Internet service that is provided on different load-balanced servers, e.g., using round-robin DNS scheduling, the performance depends on the currently selected server. An *inter-service dependency* denotes the relationship between services, e.g., an e-mail service depending on an authentication service and on a storage service. *Organization dependencies* denote the relationship among multiple domains, i.e., if dependent services and/or servers belong to different domains. A methodology to discover these dependencies was addressed in [RCN99]. IP hosts, IP links, and components in general are resources used for services. But in the MNM service model there are other, more general types of resources, e.g., staff or used processes, which are obviously not covered by [CR99]. Anyway, [CR99] is restricted to ISP scenarios. In this thesis, the types execution dependency, link dependency, and component dependency fall under the general category of dependencies among resources or dependencies from resources to services/service functionalities. The inter-service dependency and the organization dependency, corresponds in this thesis to dependencies between services or service functionalities, either provider-internally or among multiple providers (subservice usage), respectively.

A strength attribute for dependencies is defined in [BKH01] having the values strong, medium, weak, or absent. Further dependency models are presented in [CJ05] based on multi-layered Petri nets, as well as in [Has01] being concerned with dependencies of software classes.

assessment

Regarding I/R analysis, especially requirements class R1, but also the possible integration and extension for the further modeling requirement classes R2 to R4 the following can be said: The generic idea of dependencies between objects (for R1 and R2) is addressed by most approaches, but none of them includes the detail in an integrated way as demanded by the requirements: no consideration of time-dependence, QoS mapping or other parameterization of dependency instances, no details concerning dependency covered, no consideration of redundancy modeling or generic combination of multiple dependent dependencies (composite dependencies) with the exception of AND/OR-dependencies which are too simplistic. Concluding, the existing approaches for modeling of dependencies for services cannot be regarded as satisfactory concerning the requirement class R1, because they are not targeted towards the needs of service-orientation in general, and specifically not towards the needs of I/RA.

In [Mar06, HMSS06] a modeling particular for dependencies in service modeling was approached. So-called *composite dependencies* are introduced for modeling specifically $1 : n$ dependencies. However, the approach is not yet specific enough to cover all details necessary for I/RA in appropriate granularity and accuracy. Concerning R1, the details of redundancy and load-balancing have not explicitly been covered, although the composite dependency approach might provide a general basis for modeling of $m : n$ dependencies. Moreover, with respect to the other modeling requirement classes,

R2 to R4, no explicit extension for time relationships (dynamics) as well as dependencies of QoR/QoS parameter (values) were considered yet.

Research concerning dependency finding For the particular instantiation of the I/RA framework to a concrete service scenario, especially dependencies in the given service scenario, concerning basically R1, but also for extensions towards the other modeling requirement classes R2 to R4, will have to be identified. That is why, in the following research approaches concerning the discovery and identification of dependencies with respect to service modeling are treated.

querying existing dependency knowledge

[SS04] describes various methods for obtaining fault localization models. Often the discovery of dependencies is possible only in a technology dependent manner. For example, DNS-related dependencies (on hosts) are usually not be stored explicitly in configuration databases, but have to be determined from files like "resolve.conf". The *Physical topology MIB* [BJ00] provides an information source for IP networks. In [BC03] service information is auto-discovered from the configuration of network elements. In [KKC00] the querying of system configuration repositories is approached.

using passive monitoring

In [GNAK03, AGK+04] a method using passive monitoring is utilized to discover dependencies analyzing observed interactions. Message traces are used in [AMW+03] for the discovery of dependencies. The number of interactions is used as indicator of the dependency strength. A subsequent publication [AAG+04b] treats the effect of inaccurate modeling for problem determination.

using instrumented code

Basically, such inaccuracies originate from missing or false dependencies which are can be mostly avoided using instrumented code. *Application Response Measurement (ARM)* [ARM98] provides the possibility to instrument applications in order to collect additional monitoring information in general. A set of libraries for code instrumentation is also presented in [KHL+99]. A particular approach that uses instrumented code for dependency determination is given in [BKK01, BKH01]: Nodes of interest are identified and their code instrumented for monitoring. Afterwards, the effects of perturbating and injecting faults into the nodes are monitored. Moreover, the changed system behavior is used to determine the strength of dependencies which are grouped in four levels. Obviously, it is necessary to carefully apply such a method in production environments. Generally, the effort for instrumenting code may be high, and above all instrumentation is often not possible in all situations.

using AI methods

There are also approaches to use particular methods from the area of AI (Artifical Intelligence). [Ens01b, Ens01a] presents a neural network based approach. For each pair of related resources in a network the activity is monitored, using indicators like CPU load (for the whole device or per application), bandwidth utilization or combinations of the former ones. The respective activity curves are used as input to a neural network which decides whether relationship between these activities exists. Furthermore, approaches which utilize data mining techniques to event log files in order to identify patterns

have been proposed [TJ01, BHM+01, HMP02]. Identified patterns are used as indicators of dependencies.

assessment

As the above analysis has shown, the automated discovery of dependencies especially on the service level is still subject to ongoing research. So, even if some approaches for the automated discovery of dependencies exist, which may support the instantiation of the I/RA framework to a concrete service scenario, the documentation of dependencies with respect to service modeling is a great issue: The documentation of services which is required for change management anyway should therefore be combined with the parallel documentation of dependencies among functionalities and between functionalities and resources. For dynamics of dependencies, e.g., which client request makes use of which load-balanced resources, specifically instrumented code in connection with the documentation can provide a solution. On the resource level, the discovery of dependencies is usually technology specific. For instance, IBM Tivoli Application Dependency Discovery Manager [IBM] provides a set of 250 product specific sensors for such a dependency discovery.

3.3.1 MNM Service Model

The MNM service model [GHH+01, GHK+01, GHH+02] basically has been presented in Sect. 2.2, and was particularly used there for the description of the example scenario in Sect. 2.3. In the following further aspects and extensions are added to this basic introduction.

classification of management functionalities

One important aspect for I/RA is the granularity of functionality definition (requirement R1.2). The subdivision of usage functionality is usually depending in the concrete service, but concerning the management more generic subdivisions are already in place with the MNM service model: As introduced in Sect. 2.2, the MNM service model makes a basic distinction between usage and management of a service. Consequently, the overall functionality of a service is basically divided into usage functionality and management functionality. Moreover, particularly, for the management functionalities, a basic (interaction process) classification was introduced. Moreover, a generic service life cycle comprising the phases negotiation, provisioning, usage, and deinstallation were identified for use with the model. Fig. 3.1 shows this management interaction classification with the specific phases of the service life cycle where they occur.

In [Ner01] all management interactions necessary between customer and provider of service have been investigated in detail with respect to the CSM approach (compare p. 74), which lead to a refinement the basic classification of management functionalities. Table 3.1 shows an overview of this refined classification of interactions between customer and provider for management purposes, at the Customer Service Access Point, according to [Ner01].

Either the basic or the refined classification of management functionalities can be used as a starting point for modeling the management functionality

Interaction Class:	*Interactions:*
Inquiry Management	List all services in service catalog
	Inquiry of service specification of a service
	Search of a service with special characteristics
	Determination of customer profile
	Input of standardized customer's invitation
	Input of individual customer's invitation
Order Management	Input of customer's order
	Information inquiry about acceptance of order
	List all orders of a customer
	Information inquiry about a specific order
	Status inquiry about a specific order
	Modification of an order
	Cancellation of an order
	Interaction concerning delays of order realization
Configuration Management	Overview of all configured services
	Status inquiry about configuration of a service
	Notification about change of service configuration (by provider)
Problem Management	List all maintenance notifications of a customer
	Get contents of a maintenance notification
	List all problem reports for a customer
	Get contents of a problem report
	Input of new problem report
	Status inquiry of a problem report
	Modification of a problem report
	Cancellation of a problem report
	Checking of problem resolution by customer
Quality Management	Status inquiry about service access point
	Overview of status of QoS parameters
	Status inquiry of a specific QoS parameter
	Notification about SLA violation (by provider)
Accounting Management	Information about current service usage
	Sending of invoice (by Provider)
	Information about accounting-relevant aspects
Change Management	Request for change
	Information about acceptance of change
	List all change requests
	Get contents of a specific change request
	Status inquiry of a specific change request
	Modification of a change request
	Cancellation of a change request
	Interaction concerning delays of change realization

Table 3.1: CSM interaction classification according to [Ner01] (interactions are customer-initiated if not indicated otherwise)

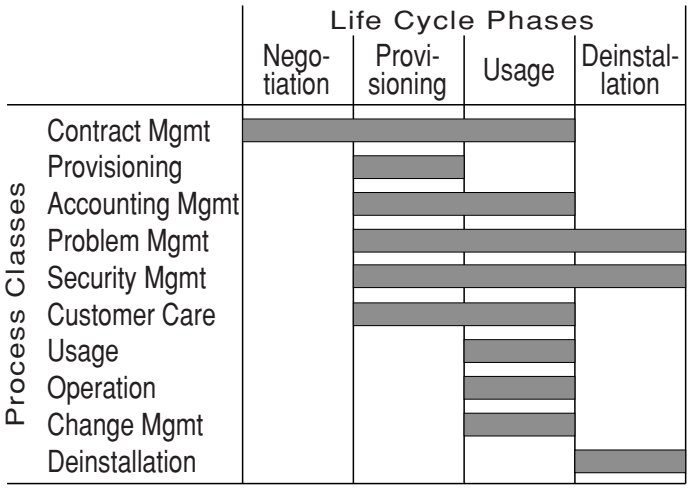

Figure 3.1: Service life cycle and interaction classification of MNM Service Model

of any given service. Concerning I/RA, such a subdivision of management functionalities may be necessary, as each management functionality may have different particular dependencies concerning their degradations, and so the knowledge of these particular dependencies may be necessary for determining entailed business impact in appropriate detail (compare p. 25 in Sect. 2.3.1).

extension of MNM service model for specification of interaction process of a functionality

Particularly in [GHH+02] it was described how the different (usage and management) functionalities of a service can be modeled in detail with UML diagrams (p. 80):

Basically, on the one hand, the different functionalities correspond to use cases in a UML use case diagram giving an overview of the functionalities as well as their basic relationships. The different relationships which can be modeled already roughly (no additional dependency attributes, only $1:1$ relationships) on this level, are basically inheritance of use cases (functionalities) and the dependency between use cases (functionalities). Inheritance of use cases can be represented explictly in UML, whereas the latter can be represented by *include* or *extend* relationships between use cases.

On the other hand, for each functionality, the detailed interaction process (between user/customer and provider) may be specified by separate UML diagrams. There are actually three specification levels, each with a different level of detail: First, there is the modeling of the interaction process with respect to the service view, i.e., the common perspective of customer and provider. This is done by UML activity diagrams, one for each functionality. Second, for approaching the modeling of the provider-internal realization of a functionality, the interaction process of each functionality may be represented by a refined activity diagram, which includes provider-internal aspects not visible and known to the user/customer. Third, as a further refinement, the interaction process for the realization of a functionality, as part of the service logic, can be specified in detail, including access point to resources and usage of subservice (functionalities) by subservice clients. For this [GHH+02] proposed

so-called UML collaboration diagrams, which nowadays, in the current UML standard, are called UML communication diagrams. Concluding, the extension of the MNM service model described in [GHH+02] provides a detailed way to describe all functionalities of a service. The specification by the first level, i.e., using activity diagrams for specification of the interaction process of a functionality from perspective of the service-view, has been particularly used by [Sch00, Sch01] for the specification of SLAs (compare Sect. 3.5.1).

instantiation methodology

In addition to providing a way how to use UML diagrams for the specification of functionalities, [GHH+02] presents a complete instantiation methodology, which allows to apply the MNM service model (with the extension of using particular UML diagrams) to any given, concrete service scenario. For this actually two different methodologies are specified, one top-down oriented and a bottom-up oriented one. Basically, the former one is customer-oriented, starting from the requirements of the customer side, and targeting towards an appropriate design and implementation of the service. In contrast, the latter one is more provider-oriented, starting from a (given) implementation and targeting towards an appropriate common definition of the service between customer and provider (service-view). The former one can be used for service offerings, or for designing a service from scratch, the latter one can be used for reverse-engineering of existing services.

assessment

Concluding, the MNM Service Model, together with its instantiation methodology, allows to model any given service scenario. But, even if it allows to specify functionalities in detail, it is not yet concerned explicitly with the dependencies of the functionalities and their particular details. Nevertheless, the specifications of functionalities by UML diagrams may be used as an overview to identify all dependencies. Moreover, in general, the MNM model provides a generic, sound, consistent basis for the modeling of service functionalities. Therefore, it might be used as a basis for a modeling to develop as part of the framework, which eventually fulfills all requirements of R1, and also provides the possibility to integrate all further aspects of the other modeling requirement classes, R2 to R4. For example, its functionality classification, basically distinguishing usage and management, and further the subclassification of management functionalities may be used as starting point to fulfill the requirement R1.2 (granularity of functionality definition).

3.3.2 CIM

The Common Information Model (CIM, [CIM]) is developed by the industry organization *Distributed Management Task Force (DMTF)* which is the successor of the *Desktop Management Task Force*. Originally, the goal of this standardization effort was the detailed modeling of a computer system. This has been extended to also address network related issues. The modeling comprises a large and detailed set of modeled entities, such as physical network connection equipment, complete hosts, applications, or even user passwords.

The aim of CIM has been to completely replace the existing Internet Information Model (p. 80) due its limitations.

detailed modeling of resources

CIM, being based on UML (see p. 80), is divided into a *Core Model* comprising basic classes and various extensions, the *Workgroup Models*, which share model parts via the Core Model. It provides class diagrams with a large number of attributes and methods which are also specified in the machine-readable *Managed Object Format (MOF)* files. The overall management approach with CIM is called *Web Based Enterprise Management (WBEM)*. It is proposed to access CIM data with a *CIM Object Manager (CIMOM)* module for which a set of implementations exists [Hei04]. Naming conventions can vary between organizations so that CIM implementations are usually not directly interoperable. CIM mostly deals with network and systems management. The standardization of service-oriented information is limited to definition of service attributes being directly associated with device attributes.

assessment

Concluding, CIM provides many classes (more than 1000) with attributes being useful for network and systems management. But, due to the insufficient coverage of service management information, it is not suitable for service management purposes, especially regarding the requirement class R1. Nevertheless it may be used for modeling dependencies between resources or at least can be a source of such dependency information.

3.3.3 ITIL CMDB

abstract recommendation for a configuration database

ITIL (Sect. 3.2.1) recommends to use a so-called *Configuration Management Database (CMDB)* [Off00] to serve as a common information source. The CMDB primarily stores information for the Configuration Management, but other parts of ITIL propose to extend the use of the CMDB for their purpose. The CMDB as common information source should contain the relationships between all system components, including incidents, problems, known errors, changes, releases, as well as reference copies (or respective references) of software and documentation. Furthermore, the information about IT users, staff, and business units can be included. Beyond this, the information about SLAs and their relationship to other information parts in the CMDB can be added. The particular pieces of information stored in the CMDB are the so-called *Configuration Items*(CIs). ITIL does not specify how these CIs have to be modeled and above all not how the CMDB can be implemented. This is entailed by the high-level nature of ITIL which allows organizations to implement the framework according to their specific requirements.

assessment

Because of this fact, the CMDB cannot be used as a basis for service modeling concerning the modeling requirement class R1. Nonetheless, having developed an appropriate modeling for I/RA framework, it may be combined with a particular implementation of the CMDB in a concrete scenario.

3.3.4 SID

part of NGOSS

The TeleManagement Forum is developing a framework for operation support systems which is called *Next Generation Operation Support and Software (NGOSS)* [TMF04b]. The goal of this framework is to create a vendor independent architecture for operation support systems for which a complete management solution can be built from independent modules. Actually, eTOM (see Section 3.2.2) has been incorporated as part of NGOSS. Moreover, another part of NGOSS is the *Shared Information/Data Model (SID)* [TMF04a] aiming at the standardization of IT asset management information as required for telecommunication service management. So, SID as the common information source for eTOM corresponds to the CMDB as the information source of ITIL.

separated modeling of services

Even though some basic concepts of CIM (Sect. 3.3.2) are used, the work is not entirely based on CIM. The model is object-oriented and structured into a hierarchy of levels according to a top-down approach. Aggregated System Entities and Aggregated Business Entities are used for a differentiated view on resource-related and business-related information [BGSS06]. While the two top layers of the hierarchy already are in relatively mature state, much work has to be done for the lower layers, e.g., concerning the definition of necessary attributes. Services are separated into two different views which are basically modeled independently from each other. The CustomerFacing Services are modeling information with relation to service management at the customer provider interface, whereas the ResourceFacing Services are modeling the utilization of resources for the realization of services. This separation has the advantage of allowing an easier modeling of the information needed for a particular purpose in the first place. However, this results also in the requirement of adding additional pieces of information to reflect the relationships across the CustomerFacing and ResourceFacing Services: For example, resources are used to provide a certain quality of service. A customer-oriented fault management (CustomerFacing Service) therefore has to access information from the ResourceFacing View to investigate the resources which are entailing certain degradations of quality of service.

use of design patterns

Generally, a big challenge in real world scenarios is the unification of information about particular devices. The information is usually not only vendor-dependent, but may also differ among different releases of the same vendor. SID applies *design patterns*, a general technique in software engineering, such as the composite pattern, to address this issue.

assessment

The development of SID is still in progress, but generally the approach is very promising to address the needs of service-orientation in the future. Thus, for I/RA it can only be partially considered as basis for a modeling concerning the requirements of R1 and potential extensions towards R2 to R4.

3.3.5 Service MIB approach (SMIB)

adequate modeling of service management information

The approach of Martin Sailer [Sai05, DHHS06, DgFS07] is based on the MNM service model (Sect. 3.3.1). It aims at addressing the issues and deficits of the existing standards regarding service-orientation. This approach is called *Service MIB (SMIB)* targeting to build a common repository of all information required for service management.

based on service attributes

A basic role for the description of a service have the so-called *service attributes*. In [DgFS07] a specification methodology for them has been devised, which is illustrated in Fig. 3.2.

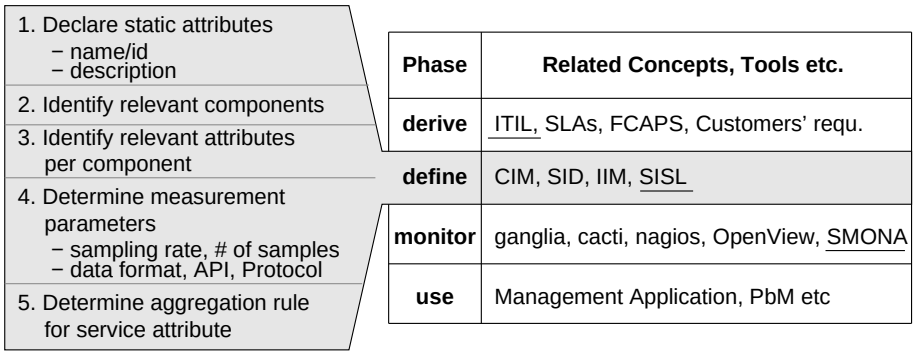

Figure 3.2: Methodology for the specification of service attributes [DgFS07]

attribute derivation methodology

This methodology is divided into the separate phases *derive*, *define*, *monitor* and *use*. In the *derive* phase the requirements for service management information are derived from the requirements of customers as well as from management frameworks (in particular ITIL, Sect. 3.2.1). The focus of the work is on the *define* phase which is subdivided into five subphases: First, information of an attribute which is regarded as static, such as name and description, is specified. Second, dependencies on other services or resources are identified, for which methods such as the ones treated on p. 84 can be applied. Following, these dependencies are refined by identifying the parameters of the related services or resources which are relevant for the corresponding service attribute. A measurement methodology is specified for these parameters, and afterwards a set of aggregation rules is determined. In the *monitor* phase the defined parameters are continuously monitored as having been specified. For this the *Service Monitoring Architecture (SMONA)* architecture has been developed. At last, the measurement results are reported to management applications in the *use* phase.

The service attributes are denoted in a declarative XML-based language called *Service Information Specification Language (SISL)*. Doing so they are independent of a specific implementation. The term service attribute used here includes QoS parameters, but can also comprise other features of a service which are not directly related to QoS, such as the use of storage space by the service. This language will be discussed in more detail in Sect. 3.4.2 concerning the specification of mapping between QoR/QoS parameters.

assessment

Basically, the Service MIB approach seems to be a promising basis for the modeling requirements of R1 (and extensions towards R2 to R4). However, a particular modeling covering really all these requirements in detail, specifically dependencies between particular functionalities and their instantiations, as well as dynamics of dependencies are not covered yet. Nonetheless, the service MIB can be used as consistent repository for most information needed by I/RA regarding R1 to R4. Particularly, the SISL language may be used to specify degradation dependencies between QoR/QoS parameters (see Sect. 3.4.2). Moreover, its SMONA architecture provides a way to measure and to access all additional information necessary for calculating such dependencies.

3.4 Related Works for Degradation and Quality Modeling

In the following, related work is discussed which is relevant to requirement class R2, that is for degradation and quality modeling. This comprises related work for the classification of degradations in general, as well as the modeling, measurement, and mapping of QoR/QoS parameters for an accurate specification of the degradations.

Research concerning degradation classification In the literature, various classifications for degradations according to different aspects have been introduced. These classifications are most often using the term *failure* instead of *degradation*.

different classifications according to different aspects

One potential, basic classification [Guo04] is taking into account the particular, negative influence on the communication of the respective, failing/degraded request/response pair:

- omission failure
  - send omission failure
  - receive omission failure
- response failure: incorrect response to a request
  - value failure: wrong value returned
  - state transition failure: expected state change not performed, wrong effects
- timing failure: failure to obey specified bounds of timing constraints
- arbitrary failure: creating arbitrary responses at arbitrary times
- crash: repeated, continuous omission failure

Furthermore, failures can be classified by the structure of their appearance as *random failures* or *systematic failures*.

In [Can03], regarding the evolution in time, failures are classified as permanent (completely), intermittent, or transient. Moreover, in [Can03], failures are classified by their *failure semantics* (mainly of interest when the failure is externally detectable):

- fail-silent: resource/service stops after failure completely without responding anymore.
- fail-stop: resource/service stops after failure only returning constant value.

- byzantine failure: resource/service fails in an arbitrary or malicious manner.
- fail-fast: in the beginning for a short period of time byzantine behavior, but afterwards fail-stop.

To this classification of failure semantics of [Can03], as inter-mediate level *fail-stutter*, as introduced in [ADAD01], can be added. Fail-stutter is a generalization of fail-stop considering multiple levels of performance degradations.

Any of the classifications of failures or degradations given above provides aspects which might have to be taken into account when appropriately modeling degradations and their dependencies for I/RA (R2), that is if it is necessary for determining business impact in appropriate granularity or accuracy.

QoR/QoS management Instead of only classifying, QoR/QoS management in general tries to specify, measure, and potentially map the concrete value (ranges) of degradations by referring to quality parameters. Quality parameters for resources are usually called *quality of resource parameters (QoR)*, sometimes also *quality of device parameters (QoD)*. Correspondingly, Quality parameters for services (or service functionalities in detail) are called *quality of service parameters (QoS)*. That is, degradations of resources or service functionalities can be in detail described by the concrete values of one or multiple affected QoR/QoS parameters of the resources or service functionality.

QoR as well as QoS parameters have to be uniquely defined: A clear specification of the *QoR/QoS parameter metric*, as well as a specification of the correspondingly used *QoR/QoS measurement methodology* is important (compare introduction of QoS parameters in the example scenario on p. 37 in Sect. 2.3.1). For QoS parameters such clear specification is normally agreed between the customer and the provider, whereas the QoR is defined and known only provider-internally.

Examples in general for QoR/QoS parameters are availability or reliability. However, in order to be really usable, e.g., for I/RA, the concrete definition (in terms of metric and measurement methodology) used for each of them has to be known. An overview and classification of QoS in general, is approached in [HSS04].

Especially for I/RA with respect to requirement class R2, three aspects of QoR/QoS parameters are important: the specification of QoR/QoS parameters and their values, the measurement of their actual values, and the mapping of QoR/QoS parameters and their values. The last aspect is specifically concerned with dependencies between QoR/QoS parameters and their values. Concerning I/RA, for each individual dependency of resources or service functionalities, this last aspect may be important as a refinement or not, depending on whether this information is needed for accurate determination of business impact.

Two particular approaches from QoR/QoS management concerned with the modeling of quality as necessary for I/RA, are treated in detail. These are the approach by Garschhammer for customer-oriented QoS measurement (Sect. 3.4.1) and the SISL language used for the specification of the mapping of QoS parameters and their respective values (Sect. 3.4.2).

A further, generic approach for the specification of QoS parameter and mapping of QoS parameters, has been presented in [DR03]. An overview of QoS specification languages in general, most of them bound to specific types of scenarios or technology, is given in [Gar04].

3.4.1 Approach for customer-oriented QoS

The approach by Markus Garschhammer [Gar04] is based on the MNM service model (Sect. 3.3.1). It addresses a methodology for the customer-oriented specification and measurement of QoS parameters.

requirements

Particularly, it was developed regarding the following requirements: provider/implementation independence, coverage of the whole service life cycle , genericity, expressiveness, coverage of QoS for usage and management: The definition of QoS parameters has to be independent from the provider's service implementation. The QoS definition should be applicable to all phases of the service life cycle (compare Table 3.1 on p. 87), in contrast to many other QoS approaches which most often only cover the usage phase. The QoS definition should be applicable to all kinds of services and should therefore be as abstract as the MNM Service Model. Concerning expressiveness, on the one hand the QoS definition should be as declarative as possible so that it can be also read by a human reader (customer-centric). On the other hand, the definition has to be precise enough to avoid ambiguities. Whereas the QoS definition today mainly deals with the usage functionality of a service, it should also be possible to define QoS parameters for service management.

QoS measurement at the SAP

As already said, the approach is based on the MNM Service Model (see Sect. 3.3.1) which already contains a generic QoS parameter class, but without specifying the way of measuring its fulfillment. For attaining a QoS measurement independent from the service implementation, the idea is to perform the QoS measurement directly at the service access point (SAP) or the CSM access point, respectively (compare Fig. 2.2 on p. 19). The complete QoS measurement process is displayed in Fig. 3.3. It consists of four steps to be discussed in the following.

listening to SAP calls

As a first step of the QoS measurement, (functionality) calls at the SAP are detected. Because it usually cannot be presumed that calls can already be detected for all kinds of interactions, a class called *SAP attachment* is added to the MNM Service Model. Essentially, this class extends the SAP in order to allow a detection of all SAP calls (and corresponding responses) and provides information about the SAP calls as *primitives*.

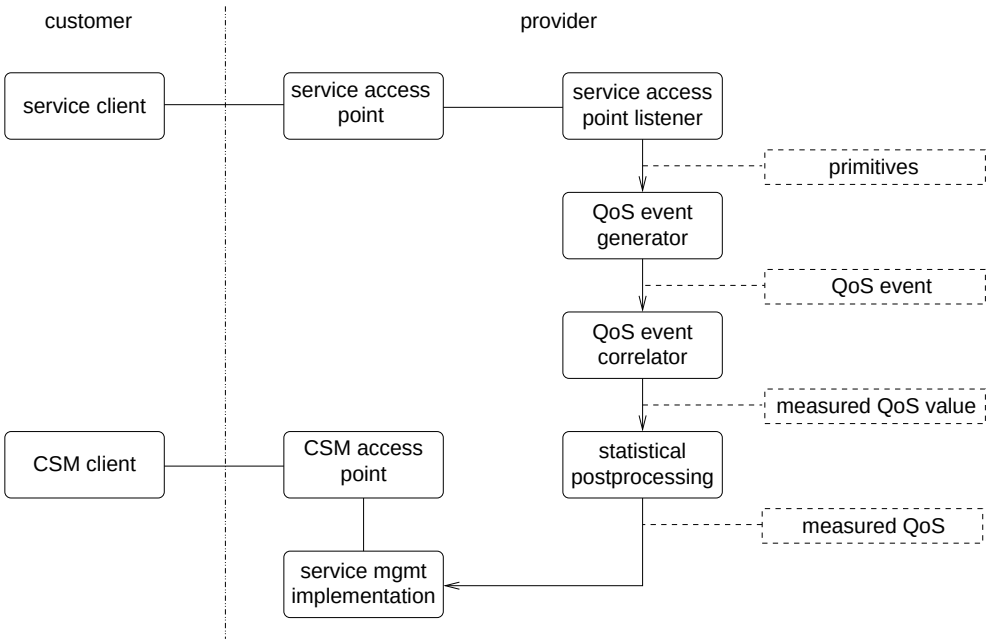

Figure 3.3: Customer-oriented QoS measurement process [Gar04]

generation of QoS events

Second, *QoS events* are produced by a further class *QoS event generator*. For this, the latter class gets *primitives* as input and processes them in a manner that events, meaningful regarding the SLA fulfillment, are generated. The events may be produced may by filtering of primitives or by grouping of similar primitives into a single event. Moreover, it is also possible to define events based on more complex pattern of occurrence for primitives.

QoS event correlation

Third, in the additional class *QoS event correlation* QoS events received from the second step are further correlated. The result in an instance of the class *QoS measurement value*. For example, a correlation can be performed for two events which are related to the request of a web site: The second event indicates the completion of the web site request whose start is indicated by the first event. The difference of the time stamps of both events can be used to calculate the access time.

statistical postprocessing

At last, the further class *postprocessing* receives the QoS measurement values and performs a statistical analysis in order to determine whether the agreed QoS has actually been met.

usage and management QoS

As demanded by the initial requirements (compare above), the QoS measurement can be used to measure both, usage QoS as well as management QoS. For the latter, the listening to interactions has to be performed at the CSM access point, instead of the service access point.

proxy for third-party monitoring

The approach comprises an extension to enable an independent third party to monitor the service quality. For this case the third party has to get access to the SAP primitives. One possibility is to introduce a SAP proxy between the SAP and the service client or service management client, respectively. Then, the primitives are measured at the SAP proxy, and used for the mea-

surement process which is now performed by the third party. Based on this, the measurement result can be accessed at a specific interface by customer and provider.

Regarding I/RA the QoS approach provides a generic way to define and measure QoS in customer-oriented way. That is, is can be used for I/RA for performing the measurement of current QoS parameter values as necessary for the mapping of degradations. assessment

3.4.2 SISL

The SISL language [Lan06, DgFS07] is part of the Service MIB approach, having been treated in general in Sect. 3.3.5. It is a generic, flexible, formal, and declarative XML-based language with the aim to cover all important aspects of service orientation in an integrated way. Particularly, it allows the specification of aggregation relationships between components and service-related in formation.

It was designed with respect to the following requirements: declarativeness, expressiveness, integration with service-related component parameters, covering of aggregation relationships, as well as specification of corresponding thresholds and alarm events. Moreover, the SMONA architecture (compare Sect. 3.3.5) has been devised for the monitoring of actual values, and the ensuring of the fulfillment according to the defined aggregation relationships.

Features of a service, so-called *service attributes* (compare Sect. 3.3.5), contained in the Service MIB, are specified in an understandable, declarative, and expressive manner, based on the aggregation of component parameters. This approach ensures among other issues to be independent of a particular implementation. The aggregation is performed using mathematical and logical libraries containing particular aggregation operators. This approach allows for later necessary extensions. The information about component parameters itself is regarded as being outside of the Service MIB. This information is measured and accessed using the SMONA architecture. But, the control of this measurement by SMONA is configured also in SISL, e.g., by specifying sampling rates and thresholds for the component parameters to be monitored. Designed in this way, SISL can be generically applied, not limited to a specific type of service, especially for the specification of mapping between QoR/QoS parameters.

Concluding, concerning I/RA, SISL can be used for the specification of the dependencies between QoR/QoS parameters and their particular values, as far as necessary for determination of detailed dependencies between degradations. Moreover, the SMONA architecture can be used for the measurement of and access to all additional management information necessary for calculating particular dependencies between QoR/QoS parameter values. assessment

3.5 Related Work for Business Impact Modeling and Recovery Action Modeling

Here related work for business impact modeling (requirement class R3) and recovery action modeling (requirement class R4) is treated. Concerning the requirement R3.1, the support for SLA penalty costs as a basic type of business degradation, SLA specification approaches in general are examined in Sect. 3.5.1. For integrating of new business degradation types with respect to the estimation of future service usage (R3.2), in Sect. 3.5.2 generic extrapolation techniques are presented which can be used to estimate future service usage from current and historic information about the service usage. Furthermore, for the support of additional degradation types beyond SLA violation costs in general (R3.3), in Sect. 3.5.3, approaches from the related area of IT security and risk management, as well as business impact analysis approaches from the area of financial management are treated. This covers also to some extent recovery action modeling.

3.5.1 SLA specification languages

Basically, a *service level agreement (SLA)* is an agreement about the functionality and the associated quality (levels) for the functionality of one or multiple services, agreed between customer and provider.

parts of an SLA

[Sch01] distinguishes 3 general parts of an SLA: the *legal part*, the so-called *service agreement*, and the *proper service level agreement*. The first one covers legal aspects and formal parts required by law, if customer and provider are different enterprises or organizations. This part make the SLA a legal contract to which customer and provider have to adhere both. The second part, the service agreement, describes the usage and management functionalities of an offered service, concerning important, technical and organizational aspects. If the SLA comprises multiple services, each one may have its separate service agreement, being part of the single SLA. The third part, the proper service level agreement, based on the second one, i.e., relating to the specified functionality of a service, defines and restricts the quality levels or service levels of the functionality: That is, respective QoS parameters for each functionality of a service are defined (including QoS metric and measurement methodology), as well as constraints on their value (ranges) are defined. For each service agreement, there should be at least one proper service level agreement, potentially multiple ones for defining different quality or service levels for different user groups or locations of the customer.

Often, the term SLA is used only with respect to the third part, which is describing the quality and service levels of a service. But such an approach is incomplete, as this third part has specifically to be based on a detailed (customer-oriented) specification of the service functionality with which the specified QoS parameter (values) are associated.

SLA specification languages

Various research approaches have been proposed for defining SLAs. Notable examples are the *Quality Management Language (QML)* [FJP99], the *Contract Definition Language (CDL)* [BCS99], the *Web Service Level Agreement (WSLA)* [KL02] and its predecessor *WS-Agreement* [WSA05]. The last one, e.g., defines a set of potential SLA elements which are parameterized for a given scenario. Moreover, SLA elements are also proposed in the *SLAng language* [LSE03]. In [CFK+02] a protocol for SLA negotiation (SNAP) has been devised.

SLAs based on workflow descriptions

In [Sch00, Sch01] SLAs, based on the MNM service model (Sect. 3.3.1), are specified on the basis of workflows to allow to specify in a customer-oriented way the particular functionalities of a service. Actually, the workflows are specified by UML activity diagrams, which correspond to the interaction process specifications of service functionalities within the UML-diagram extended MNM service model (compare p. 87 in Sect. 3.3.1). Furthermore, the activity diagrams contain as diagram annotations explicit constraints, associated with single or groups of activities in the diagram, in order to specify conditions for related QoS parameters in the SLA.

assessment

In general, any approach for SLAs definition allows to specify constraints for restricting QoS parameters and their values in a formal way. To be used for I/RA framework, the SLA definition has to be done in relation to the service (functionality) modeling done, especially regarding the two previous modeling requirements classes R1 and R2.

That is, the approach by [Sch00, Sch01] is a good candidate for this, as it already explicity includes such associations to the service modeling, i.e., to the interaction workflows of the service functionalities also included as part of the SLA.

3.5.2 Extrapolation Techniques for predicting future service usage

Extrapolation techniques from the area of statistics, are widely applied in many different areas today. They represent a kind of data-based (objective) method for predicting future values in a given time series. Particularly, they use historical/current data of a time series as the basis for estimating future outcomes.

trend and seasonality

Extrapolation techniques have to deal with various systematic patterns in the given time series data. Such patterns include damped/linear/exponential trends in the data or additive/multiplicative seasonality, i.e., cyclic recurrency of particular patterns in the data.

various extrapolation techniques

Various extrapolation techniques have been developed: These include methods for moving average (MA) in general, exponential smoothing (exponential moving average), and autoregressive methods (AR). Moreover, combinations of MA and AR, autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA), as well as further extensions

have been developed. A detailed classification and evaluation of can be found in [MWH98].

Moreover, in contrast to data-based (objective) methods for forecasting by extrapolation techniques, there exist so-called judgmental (subjective) methods for prediction. A complete classification and comparison of prediction methods in general, can be found in [JA01].

Particularly, in [SSK+06] an approach for forecasting costs and revenue depending on IT service usage was presented based on a combination of simulation and application instrumentation.

assessment

Extrapolation techniques can be reused for I/RA, that is for estimating future service usage from historic and current values of the service usage. Historic/current data concerning service degradations can be integrated with estimates of future service usage to predict service degradations in the future. Based on this corresponding types of business degradations taking into account this affected future service usage, e.g., for calculating dynamic, future revenue loss, can be defined.

3.5.3 Financial Business Impact and Risk Analysis

Here, methods and concepts related to I/RA from the financial management area are introduced. In general, in industry today, many best practice approaches, utilizing e.g., questionnaires and text templates, for (financial) Business Impact Analysis and Risk Analysis are used.

Business Impact Analysis

Business Impact Analysis (BIA) [Bus] is concerned with the identification the critical business functions within an organization, and the determination of the impact occurring when these business functions are not performed above their maximum acceptable outage. Relevant factors that are normally used to evaluate the impact include customer service, internal operations, legal/statutory and financial. Business Impact Analysis is an essential part of the so-called *business continuance plan*. As such, it includes the exploration of vulnerabilities and potential risks, as well as the planning of strategies in order to minimize these risks. Furthermore, Business Impact Analysis identifies the more important components of the organization and determines a corresponding, necessary funding of these components with respect to potential disasters.

Consequently, Business Impact Analysis has a strong focus towards disaster recovery, being related to the area of ITIL (Sect. 3.2.1) continuity management. Therefore, it is only marginally relevant for the I/RA framework in this thesis, because I/RA is mainly concerned with problem and incident management, rather not being concerned with catastrophic events stopping the whole business, potentially also the total IT infrastructure including components for realizing I/RA.

risk analysis

Risk analysis [Ris, Bus] is concerned with the identification of the most potential threats to the organization and the analysis of related vulnerabilities to

those threats. It includes the evaluation of existing physical and environmental security and controls, as well as the assessment of their appropriateness regarding potential threats. Risk analysis systematically studies the uncertainties and risks of the organization with respect to areas as business, engineering, public policy, and others. In more detail, it identifies the potential risks, understands how and when they arise, and estimates their (financial or otherwise) negative impact.

quantitative risk analysis

Quantitative risk analysis [Ris] uses mathematical models or simulations for a project or a process, including parameters to model the uncertainty which cannot be controlled, as well as decision variables which can be controlled. A quantitative risk model calculates the impact of the uncertainty parameters and the potential decisions concerning aspects as profit and loss, investment returns, and environmental consequences. Generically, a risk is basically defined in terms of two components, its *probability* and its so-called *impact*: $risk = probability \times impact$, the product of the probability of a respective accident related to the risk, and the impact as measure for negative influence of such an accident. In [IRM02] a standard for risk management in general has been published. [SGF02] proposes a risk management guide especially with respect to the management and operating of IT systems.

Key Risk Indicator

A particular kind of risk, being related to I/RA in general, is the so-called *operational risk*: This term comprises risks which arise from the organization's business functions and from the practical implementation of the management's strategy. Examples are information risks, fraud risks, physical or environmental risks. Often, operational risk are assessed by so-called KRIs. A *Key Risk Indicator (KRI)* is a measure to indicate the degree of a particular (operational) risk.

KPI vs. KRI

The concept of KRI is the inverse to the concept of *Key Performance Indicator (KPI)*, which is used also in financial management in general as well as in IT management standards such as ITIL (Sect. 3.2.1). Key Risk Indicators differ from Key Performance Indicators in that the former is an indicator of the possibility of negative, future impact, whereas the latter is as a measure of how well something is performed. The Risk Management Association provides a online-library of defined KRIs [KRI].

Management by Business Objectives

The approach *Management by Business Objectives (MBO)* [BS04, SB04] proposes a method for decision support for IT management, based on mathematical optimization methods concerning the financial impact of an aspect of the business, with the aim to to minimize financial losses. Specifically, in [RSM+07] the scheduling of changes was targeted by this method. This is related to the scheduling of recovery actions, being part of a recommended recovery plan. But the MBO approach is not explicitly concerned with the identification, modeling of particular (recovery) actions, and the modeling of their relationship to post-recovery impact. As MBO is based on mathematical optimization methods, it needs an appropriate mathematical description of recovery actions and their influence as input.

assessment

Concerning the I/RA framework of this thesis, Business Impact Analysis and

risk analysis, especially quantitative risk analysis, may give hints on what information has to be considered and how it is identified, especially concerning the specification and importance of different types of business degradations beyond SLA penalty costs, such as e.g., revenue loss, loss of reputational image, regulatory costs (requirement R3.3). Particularly, the general concept of KRIs may be used or adapted to define and potentially also to prioritize such general types of business degradations.

There is no explicit, integrated modeling of business degradations (requirement class R3) recovery actions and their effect on post-recovery business impact (requirement class R4) as far as IT management and especially resource degradations are concerned, yet. Therefore, the development of the I/RA framework will include such a general modeling covering all requirements of R3 and R4, being compatible with the modeling for R1 to R2. The MBO approach might be used as method for actually recommending recovery plans composed of recovery actions, after the modeling for R3/R4 is converted in an appropriate, mathematical structure as input for MBO.

3.6 Related Work Concerning the Implementation of the Tasks for I/R Analysis

At last, related work concerning the implementation of I/RA, especially necessary to realize the tasks of I/RA (R5.1), is discussed. This comprises generic implementation techniques, mainly to reason about the various types of information, including all types of dependencies, used for I/RA. These techniques will have to process the various parts of the modeling demanded by the modeling requirement classes (R1 to R4).

Decision Trees A generic method for specifically impact analysis in general, being not limited to IT management, are decision trees [KT03, Pea88]. Trees are used to model dependencies: Nodes of the tree are corresponding to dependent objects, and the edges of the tree are corresponding to the dependencies between the objects. But this approach generally is not taking into account a complex structure of the dependent objects, nor at all a complex structure of the dependencies, as needed for I/RA (R1 to R4): For instance, complex structure regarding dynamics in time, complex redundancy patterns, $m : n$ relationships in general, e.g., between QoR/QoS parameters.

Therefore, for the realization of the I/RA tasks (R5.1) more general implementation techniques, which allow to take into account all modeling requirements of R1 to R4, especially for modeling and processing dependencies, are necessary. That is why, in the following *reasoning methods* in general are investigated, to be mainly reused for defining and working with dependencies.

techniques for reasoning about dependencies in general

Reasoning Techniques about Dependencies in General Concerning reasoning in general there are various implementation techniques. Two general implementation techniques which are suitable for reasoning about dependencies are *Rule Based Reasoning (RBR)* and *Case-Based Reasoning (CBR)* discussed in the following. Both techniques are often used for the implementation of expert systems.

rule-based reasoning

Rule-based reasoning (RBR) [Lew99, JW93] uses for the reasoning process a set of rules which in general have the basic form $conclusion$ **if** $condition$. The *condition* is based on received events from the outside world and current state information within the system, more explicitly within the working memory. Whereas, the *conclusion* can on the one hand trigger actions in the outside world, or changes to the system state, or on the other hand affect the choice for the rules to be used next, normally by referencing these rules within the conclusion.

basic architecture of RBR system

In Fig. 3.4 the basic structure of a rule-based system is illustrated. Events are received or observed from the outside world and stored in the working memory. Each time a particular rule from the rule base is selected (according to the condition) and applied (according to the conclusion). This results in

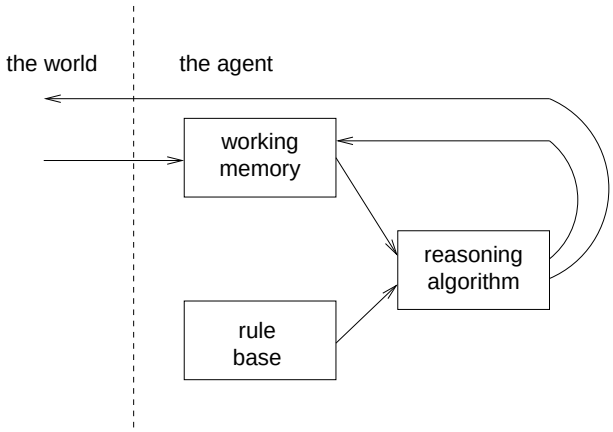

Figure 3.4: The basic structure of a RBR-based system [Lew99]

appropriate updates of the working memory, the triggering of actions in the outside world, or an updated choice for the rules to be used in the following.

RBR and formal logics

RBR systems, being a very general concept, are often related to and particularly realized by logic programming, e.g., with a prolog-like reasoner (PROgramming in LOGigcs) [Llo87]: The logic programming is usually based on the logical inference according to some formal logic, e.g., as the predicate logic or some extension thereof. The formal logic is mainly used for knowledge representation. Usually knowledge is represented by base facts (logical predicates), and logical rules (logical implications), in the formal logic as well as in its realization within the prolog reasoner (as prolog predicates and prolog rules). From the base facts and the logical rules by logical inference (according to the formal logic used) further valid knowledge (facts) can be derived: The logical rules describe a logical interference relationship between assumptions (rule body) and a conclusion (head of the rule). But, in addition to predicates (facts) which are being used for pure knowledge representation in the prolog reasoner and the formal logic alike, there can be special predicates within the prolog reasoner, which have side effects apart from their logical knowledge representation in the formal logic. Such side effects especially comprise the exchange of data observed from the outside world or the triggering of actions.

Utilizing such special predicates with side effects within a prolog reasoner, the corresponding logical programs (logical facts and rules respectively prolog predicates and prolog rules) can be used to realize RBR systems. In this respect, the logical rules correspond to the RBR rules in the RBR rule base, whereas logical facts (logical predicates) correspond to the system state in the RBR working memory.

Especially, for RBR system realized with a prolog reasoner, the conditions of RBR rules are usually expressed in the formal logic used as formal basis of prolog reasoner. Consequently, the formal logic used determines the power of expression of the conditions of the realized RBR rules. That is why in the following, particular formal logics and their features are treated, regarding

their possibility for accurate knowledge representation (concerning R1 to R4) to be used for the realization of RBR systems.

The most common, canonical examples for reasoning by formal logics, are statement logics (propositional logic) and its extension, the predication logics (first order logic, FOL). Both are deductive logics, which means that they are only concerned with the derivation (inferencing) of valid, derived facts from given (base) facts and rules. The former allows only to reason about simple statements, being normally being connected by the basic logical operators AND, OR, and NOT. The latter allows to reason about predicates, which are superior to statements as they allow parameterization with terms (constants, variables and functions).

Originally, prolog-like reasoners [Llo87] are using the predication logic as their formal basis. But, also often extensions of prolog-like reasoners are used to reason according to extensions of the predication logic.

Moreover, originally prolog-like reasoners, in general are only concerned with the proofing of single facts or with the consecutive finding of single, valid facts with a particular structure (concerning their included terms). This is the aspect for which such systems were originally developed and for which efficient algorithms are used within them.

deductive databases

In contrast to such conventional prolog reasoners, the *Deductive Database (DDB)* [CGT90] approach is specifically concerned with efficient finding of all valid facts which are derivable from a given set of base facts and logical rules. This approach is specifically interesting for impact analysis, as the task of impact analysis is to derive all business degradations potentially entailed from given resource degradations, in deductive manner. Therefore, DDB and F-Logic, which can be used as a particular specification/query language for DDBs, will be treated in more detail in Sect. 3.6.1.

Beyond, such extension of prolog reasoners as the DDB approach, various extensions of prolog reasoners exist which use extensions of predication logic or alternatives (not only being deductive) as their formal logic. Such extensions increase the expressiveness (according to the formal logic) when used as basis for realizing RBR system: the conditions and conclusions of the RBR rules, concerning aspects such as the reasoning about time. Such extensions can also be potentially combined with each other, or also be integrated with the DDB approach. Therefore, some extensions of predication logic, interesting for the implementation of I/RA, are discussed in the following.

reasoning about time

The *temporal logics* are approaches for the formal reasoning about time and duration. In addition to normal logic operators, such as AND, OR, NOT, specific operators for time-relationships between predications are used, e.g., the *until* operator: $(B, \mathcal{U}, C)(\phi)$ expressing that there is a instance of time (ϕ_i) until that B is valid and afterward C is valid forever: $(B \,\mathcal{U}\, C)(\phi) \equiv (\exists i : C(\phi_i)) \wedge (\forall j < i : B(\phi_j))$. Various subtypes of temporal logics have been devised, being concerned with different notions of time, e.g., concerning single points in time, paths in time, or even trees expressing

multiple potential paths in time. Even, some approaches have been developed to reason about the combination of time and potential actions and their effects [Lam94], which may be used for planning. Moreover, there is the so-called *event calculus* [KS86] as an alternative for reasoning about time and actions/events and their effects, also suitable for planning.

logics for planning

A further particular logic which is specifically suitable for planning in a generic manner is the Transaction Logic [BK96]. This one is specifically treated in more detail in Sect. 3.6.2.

reasoning by cases

As an alternative to RBR, Case-Based Reasoning (CBR), a type of analogous reasoning, not being based on or connected to formal logics, is treated in Sect. 3.6.3. CBR can, among many other tasks, also be used for planning.

assessment

To sum it up, RBR and CBR are generic reasoning techniques, which can be potentially used for the implementation of the various tasks of I/RA (R5.1): For impact analysis the use of a RBR based on the Deductive Database approach (Sect. 3.6.1) seems to be promising. For the recovery recommendation, which is basically a kind of planning, RBR based on appropriate logics, such as the Transaction logic (Sect. 3.6.2), or alternatively CBR (Sect. 3.6.3) can be used. In addition to that, both approaches based on logic, DDB or Transaction Logic, may be combined with an appropriate temporal logic for the explicit reasoning about time and duration. Similarly, the other I/RA tasks can be realized by one of these methods.

3.6.1 Deductive Database Approach and Frame Logic

combination of prolog and databases

Deductive Databases (DDB) [KLW95] are based on the combination of logic programming (by prolog-like reasoners) and (relational) databases. Basically, a DDB is similar to a conventional prolog-like reasoner, in that its memory contains logical (base) facts and logical rules (together conventionally called a *logic program*). Also similarly, in a deductive manner from the base facts and the rules by (logical interference) valid, derivable facts are determined. But in contrast to conventional prolog-like reasoners, DDBs can handle large amount of facts in a similar manner as a database. The deductively derived facts are not every time recomputed (as for a conventional prolog-like reasoner) when needed, but instead precomputed and cached by efficient algorithms, while still guaranteeing the soundness of the underlying formal logic.

This way, DDB share the advantages of both base concepts, logic programming and databases: On the one hand knowledge is encoded by facts and rules in a declarative fashion, derived knowledge can be automatically be computed (by logical inference), and fact and rules are basically handled similarly. On the other hand, the DDB can guarantee safe storage of large amount of data (facts), providing access by a declarative query language, and can ensure database integrity, e.g., by use of transaction paradigms. In contrast, conventional, (relational) database systems are usually concerned only with (base)

facts, as tuples in relations, whereas a concept like rules is only marginally covered by extended database concepts such as database views or database triggers.

efficient derivation of all derivable facts

DDBs support the deductive reasoning, from given base facts and related rules to the set of all derivable, valid facts. But they consider all potential, derivable facts at once, which is different to conventional prolog reasoners, which can only consecutively determine the valid, derivable facts of a certain pattern step by step, and are recomputing each solution and any necessary intermediates each time again. This difference makes DDBs superior to conventional prolog reasoners in many situations concerning performance as well as computability. Concerning the former, efficient algorithms have been developed for DDBs for pre-calculating all derivable facts (only once). Concerning the latter, conventional prolog often can end up in endless-loops (depending on the used logic program), if cyclic dependencies between logical facts exist. For DDBs instead sound logical semantics have been developed to cover such cases on the one hand, and on the other hand to efficiently cope with such situations.

languages for DDB

Originally, DDB systems were designed as pure query systems, similarly as pure databases. Usually, one common language is used for the specification (of facts and rules) as well as for querying, similarly as in conventional prolog. Various specification/query languages have been developed: For instance, Datalog, Logical Data Language (LDL) [NT89], identical query language (IQL), Frame Logic (F-Logic) [KLW95]. There are also approaches to extend these specification/query languages into real programming languages, without loosing the sound logics semantics, e.g., for Frame Logic in XSB prolog [xsb] with the Flora/Flora2 system [YK00, YKZ03]. That is why Frame Logic in general is treated in the following.

object-oriented DDB approach - F-logic

Basically, F-logic is a knowledge representation language, not limited to DDBs. It can also be used for ontology representation.

In contrast to many other specification languages for logical programming which are only data-oriented (relation-oriented), it is object-oriented. So, F-logic stands in the same relationship to object-oriented programming in general as classical predicate calculus and conventional logical programming stands to relational database programming. Actually, F-Logic combines both approaches: In a declarative way it supports structural aspects of object-oriented and relation-based languages. Therefore, it allows to combine the reasoning about logical relations or predicates in general with the reasoning about specific object-oriented related relationships of objects and classes. Extended features of F-logic include, among others, negation, functions, sets and set-valued functions, object identity, complex objects, query methods, encapsulation. non-monotonic inheritance, polymorphism,

assessment

Impact analysis basically is a deductive task, as it has to determine all potential resulting degradations based on given resource degradations (corresponding to base facts), and dependencies of degradations (corresponding to rules). Thus, the DDB approach in general presents a promising basis for impact ana-

lysis, as it exactly addresses this type of task. Furthermore, F-Logic, being an object-oriented language used for DDBs, seems to be a good candidate for a implementation language for the specification and processing of complex dependencies as demanded from the requirement classes R1 to R3 (concerned with impact analysis). Moreover, this may be combined with a temporal logic (compare p. 105) for the explicit reasoning about time and duration as demanded by R2.1 (dynamics of dependencies).

3.6.2 Transaction Logic

Transaction Logic [BK96] is an extension of predicate logic. It has both, a declarative and a procedural semantic, that describe state changes in logic programming over dynamic databases, i.e., logic databases whose content changes as part of the reasoning process. Therefore, it is addressing various aspects, such as hypothetical updating, nondeterminism, and artificial intelligence via behaviors of object-oriented databases. It is especially suitable for planning tasks.

assessment

Consequently, it is a candidate for implementing the I/RA task of recovery recommendation. That is, it can be used for the planning recovery plans made up of multiple recovery actions.

3.6.3 Case Based Reasoning

reasoning based on past cases

Case-based reasoning (CBR) is a kind of reasoning by analogy. It solves new problems based on the solutions of similar past problems. General information about CBR and its original applications is given in [Kol93, AP94].

A particular application in network management is treated [Lew93, Lew95, Lew99]. This approach uses past symptom reports which have been formalized and entered into a case database along with a found solution. The proposed solution of a current symptom is reusing the solutions stored for past similar situations.

In Fig. 3.5 the general steps of CBR are visualized together with the alternatives that are available to realize step [Lew95], being discussed in the following. The general steps are case retrieval, adaption ,execution, and organization.

case retrieval step

First, during *case retrieval* similar, related cases are identified from the case database. Various alternatives for the actual matching of the current with the related, historic cases are available: Multiple *key terms* being part of the case description can be used for this matching. Such key terms can be predefined from expert knowledge or they may be determined automatically. *Relevance matching* is a refinement of the former method: Rules are applied to map a new symptom to a symptom type. Moreover, in *structure matching*, the structure of the case descriptions is exploited for the matching. Such a structure may be described e.g., by ‘connected-to” or “part-of” relationships. As

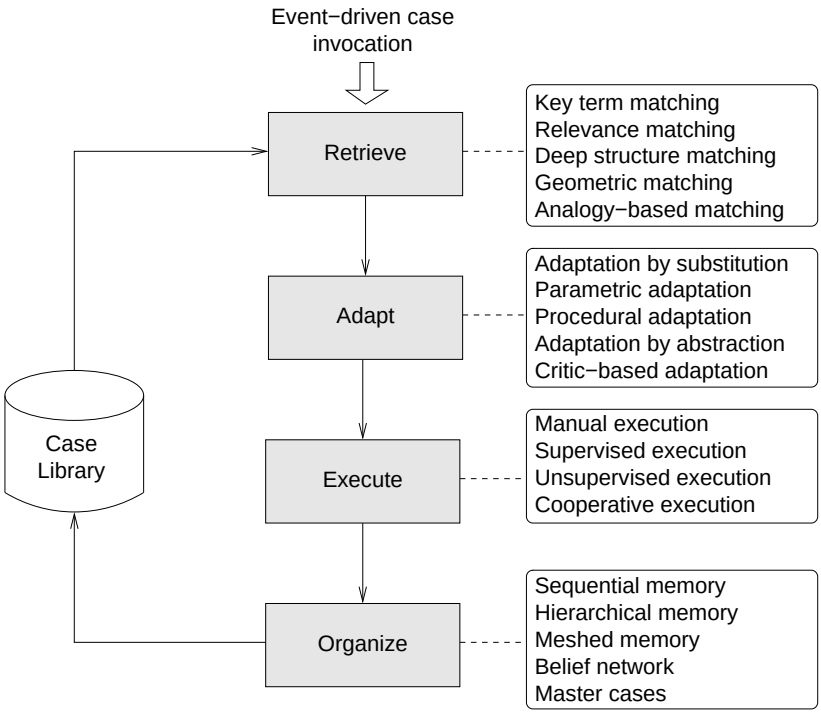

Figure 3.5: The general steps of CBR and realization options for each step [JLB04]

a further alternative, *geometric matching* utilizes a distance metric on cases to determine a distance to prior situations. As a quite different alternative, *analogy-based matching* tries to find a match to cases from a different domain.

adaptation step

As a second step, during *adaptation*, the solution of the matched past cases is adapted to the current situation. One possibility is the *null adaptation* where simply the past solution is taken without any actual adaption. By *adaptation by substitution* respective parts of the past and the current situation are substituted in the past solution to yield a solution for the current situation. Going further, *parameterized adaptation*, recomputes parameters in the past solution according to respective, new input values from the current situation. If possible, generalization from past cases can be applied in *adaptation by abstraction*. *Procedural adaptation* explicitly uses a defined procedure to adapt a past solution. Any of the mentioned adaption techniques can be additionally combined with *critic-based adaptation*, which means that a human operator can check and potentially manually change the automatically proposed solution.

execution step

As third step, during *execution*, it is tried to apply the proposed solution. Generally, it can be distinguished between manual, supervised, and unsupervised execution.

reorganization of case base

As the last step, the code database is updated in order to reflect the new situation as well as its solution found. Different organization schemes of the case database in general play an important role for this: In the simplest case,

sequential organization, new cases are simply added at the end of the code database. In *hierarchical organization* cases have to be organized in a hierarchy concerning their structure. Two particular subcases are important: meshed hierarchical organization or the use of master cases. In the case of the former one, different cases which are actually equivalent (according to some defined similarity condition) are added as additional links to the existing hierarchy. In the case of the latter one, basically only so-called master cases, which have been rated to be important in the future, are kept in the case base. Moreover, for situations with probabilistic features, *belief networks* [Pea88] with appropriate likelihood measures may be applied.

automated case generation

In the beginning of the usage of a CBR system its case database is usually empty so that a time for learning is required to make benefit from the system. A possibility to overcome this, is to try to fill the case database by knowledge from other existing, sources e.g., from existing rule databases.

relationship to RBR

In general, CBR has the advantage over RBR that it can potentially cope with completely new, up-to-now not occurred situations (cases), as long as they are similarly to some situation in the case database. Of course, this is only possible, if the case database has already been filled appropriately with experience over time (or somehow automated in the beginning, compare above). Nevertheless, the performance of CBR is usually less than that of RBR systems.

CBR and planning

CBR can be also used for planning, as it is e.g., in SpectroRx[1] [Apr04] (compare Sect. 3.1).

assessment

As CBR can be used for planning, it especially represents a candidate for the implementation of the I/RA task recovery recommendation.

[1] Originally from Cabletron Systems which were renamed to Aprisma Management Technologies and then acquired by Concord, now part of Computer Associates

3.7 Assessment

In the following, an assessment of the related work introduced from Sect. 3.2 to Sect. 3.5, concerning the different requirement classes for I/RA, R1 to R5, is performed. Here only approaches are considered which to relatively high degree cover the requirements of the respective requirement classes and which are therefore really relevant for the design and implementation of an I/RA analysis framework. The related work investigated for each class is assessed according to the generic requirements (R0) as well as to the requirements of its respective class (R1 to R5). First the requirement class R5 on a conceptual level, whose related work was treated in Sect. 3.2, is treated. Then the classes of modeling requirements, R1 to R4, whose related work was treated from Sect. 3.3 to Sect. 3.5, follow. Last, the implementation techniques introduced in Sect. 3.6, which can be reused or adapted to actually realize the tasks of requirement R5.1, are assessed.

only relevant approaches which have a considerable coverage of the respective requirements

In detail, for any requirement class R1 to R5, each related work which is taken into account is evaluated regarding the respective requirements with one of the following coverage status:

coverage status used

- n/a : requirement not applicable.
- - : no coverage of the requirement at all.
- 0 : insufficient coverage of the requirement.
- + : partial coverage of the requirement.
- ++ : good coverage of the requirement.

In addition to that, the status + or ++ can be marked by (a), i.e., +(a), or ++(a), which means that a suitable adaption or implementation of the related work assessed is needed in order to actually cover the respective requirement.

Concerning the particular requirements shown in the evaluation tables in the following, compare their detailed description of the requirements in Sect. 2.4, as well as their overview in Fig. 2.13 on p. 67.

Table 3.2 provides the assessment of the related work concerning the course of I/R analysis conceptually, namely the assessment the existing IT process frameworks ITIL and eTOM regarding existing workflow and detailed tasks descriptions for I/RA.

related work concerning the course of I/RA (R5) conceptually

Neither ITIL nor eTOM has a detailed description of the tasks of I/R analysis. Moreover, the usage of only assumed degradation, by simulations (R5.2), is not covered explicitly and in detail by any of the two standards. Concluding, for the framework a respective description of the tasks, of I/RA, e.g., by detailed workflows, will have to be developed. Ideally, such workflow description should also include how to use simulations with assumed degradations for proactive management, e.g., in the area of availability management.

requirement	related work assessed	
	ITIL	eTOM
R0.1 - integration	++	++
R0.2 - genericity	++	++
R0.3 - manageability	+	+
R5.1 - tasks of analysis	0	0
R5.2 - range of analysis	0	0
R5.3 - urgency of analysis	+	+

Table 3.2: Assessment of related work concerning course of I/R analysis (R5) on an conceptual level

related work for service modeling in general (R1)

Concerning the requirement class R1, concerned with the modeling of service functionalities and corresponding dependencies, only approaches are considered which really are suitable for modeling of *services* in the high-level, customer-oriented sense in this thesis (compare Sect. 2.1). That is why, approaches like the Internet Information Model, which is mainly used for resource modeling, or the management information model of OSI, which was never widely used above all not for high-level service management (compare p. 80), are not considered here:

Concerning the Internet Information Model, also especially the requirement of manageability (R0.3) would be insufficiently covered by the Internet Information Model. To define new types of objects (of a changing environment of a service provider) is quite difficult, as the modeling approach used by the Internet Information Model is quite simple resulting in inability to allow extensions (nor changes) easily and in structured way. Therefore, the Internet Information Model, even if used today for many IP-related devices, at any rate can not provide a complete, suitable modeling of service (dependencies) as needed for I/RA. The information model of OSI is much more advanced than the Internet Management Model, though nevertheless it has similar shortcomings concerning the explicit coverage of the R1 requirements (e.g., R1.2) as the approaches actually assessed below. Because it was never widely accepted and is today mostly displaced by other standards such as CIM, it is left out from the list of explicitly assessed approaches below.

Table 3.3 presents the assessment of the remaining relevant approaches concerning R1, namely of the MNM service model, CIM, ITIL CMDB, SID, and the Service MIB (SMIB). Actually, the last of these approaches, SMIB, is based on the first one, the MNM service model.

Concerning granularity of functionality definition and corresponding dependencies (R1.2), any approach is missing the possibility for such a definition in detailed granularity yet, with respect to detailed dependencies of Sect. 2.3 such as $r_{\mathrm{websv(x)}}(\textit{configuration} = \text{special}) \rightarrow f_{\mathrm{web/use/apage\_special/mysqconf}}$, or $f_{\mathrm{auth/use}} \rightarrow f_{\mathrm{mail/use/send}}(\textit{authentication} = \text{yes})$. Moreover, the possibility to define and use $m : n$ dependencies (with multiplicity ≥ 1) and their accurate specification, also including related aspects like redundancy and load-

requirement	related work assessed				
	MNM	**CIM**	**CMDB**	**SID**	**SMIB**
R0.1 - integration	+	+	++	++	++
R0.2 - genericity	++	++	++	++	++
R0.3 - manageability	+	+	+	+	++
R1.1 - multi-domain	++	+	++	++	+
R1.2 - granularity	+(a)	0	0	0	++(a)
R1.3 - detail level	+	+	+	+	++
R2.4 - multiplicity	+(a)	0	0	0	++(a)

Table 3.3: Assessment of related work concerning modeling of service functionalities and their dependencies in general (R1)

balancing, for usage towards I/RA is not covered by any of the approaches: with respect to dependencies of Sect. 2.3 such as $r_{\text{websv(x)}}\left(\begin{smallmatrix}\textit{configuration}\\ =\text{normal}\end{smallmatrix}\right)_{x=\atop 1,\ldots,10}, f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}}\left(\begin{smallmatrix}\textit{customer}\\ =\text{TUM}\end{smallmatrix}\right)$ coordinated by a central load-balancer, or $r_{\text{afs\_sv1}}, r_{\text{afs\_sv2}}, r_{\text{afs\_sv3}} \rightarrow f_{\text{store}}$ using cooperative load-balancing, or $r_{\text{dns\_sv1}}, r_{\text{dns\_sv2}} \rightarrow f_{\text{dns}}$ using round robin DNS. But especially the generic MNM service model, which is based on the definition of generic terms suitable for any service scenario, already has a generic basis (including an instantiation methodology) in that it allows to basically specify functionalities and their detailed interaction workflow in a generic manner (UML use case/activity diagrams). Moreover, this includes also the concept of assigning QoS parameters specifically to particular functionalities (and their interaction scheme in UML diagrams), which provides a basis for the next treated requirement class, R2. Therefore, the MNM service model or the SMIB approach based on it may be adapted and refined to support all the aspects for the specification of dependencies, after appropriate types of models for such dependencies have been developed. This will be a particular task during the development of the framework.

related work for degradation and quality modeling (R2)

Table 3.4 presents the assessment of related work concerning the modeling of degradations and quality aspects of service functionalities and their dependencies (R2). Specifically one approach concerning the actual measurement of customer-oriented quality, and one for the specification of mappings between quality parameters are assessed, namely the QoS approach measurement by Garschhammer (Garha), and the specification language SISL.

Basically, concerning QoR/QoS parameters used for I/RA three aspects have to be taken into account: the definition of QoR/QoS parameters (specifying degradations in detail), the measurement of their actual values, and the mapping of parameter specifications as well as of their values (describing dependencies between degradations in detail). An example for a detailed QoR/QoS mapping is the relationship between resource degradation g_{r1} (high link utilization) and the entailed high mail sending delay parameters $q_{\text{mail/delay\_send\_intra}}$, $q_{\text{mail/delay\_send\_extra}}$, and $q_{\text{mail/delay\_mbox}}$ in the example of Sect. 2.3.4. Consequently, in Table 3.4 the requirement R2.2 concerning

requirement	related work assessed	
	Garha	**SISL**
R0.1 - integration	++	++
R0.2 - genericity	++	++
R0.3 - manageability	++	++
R2.1 - dynamics	0	+
R2.2.a - definition of degradation types	++	++
R2.2.b - dependencies of degradation types	0	++
R2.3.a - specification of degradation values	++	+
R2.3.a - measurement of degradation values	++	0
R2.3.b - dependencies of degradation values	0	+
R2.4 - support for multiple degradations	+	+

Table 3.4: Assessment of related work concerning the modeling of degradations and quality parameters for functionalities and their dependencies (R2)

degradation types has been split into definition of degradation types themselves, and definition of their dependencies. Furthermore, the requirement R2.3 concerning degradation value (range) has been split into specification of degradation values, measurement of degradation values, and specification of dependencies between degradation values.

As the Garschhammer QoS approach is designed for the (customer-oriented) definition and measurement of QoS parameters only, it lacks a possibility to specify or derive dependencies between QoS parameters as needed for the purpose of I/RA. In contrast, the language SISL can be used for such a specification of QoD/QoS parameter dependencies, while not being concerned with the actual measurement.

Therefore, a combination of the approaches seems to be a promising basis for the definition, the measurement, and mapping of quality of degradations of service functionalities for I/RA. The detailed appropriate modeling specification of dependencies for service functionalities, having been demanded as the result of the assessment concerning R1, should be further extended to integrate this combination of the Garschhammer QoS approach and SISL, or in general to integrate any other comparable QoR/QoS specification approach (such as the Qual language in [DR03]): This extension should allow to combine the definition of functionalities and their dependencies, in appropriate granularity, with the definition, measurement, and dependency mapping of quality parameters/parameter values.

related work for modeling of business impact and recovery actions (R3+R4)

Table 3.5 illustrates the assessment of related work concerning the modeling of business impact (R3) as well as of recovery action (R4), i.e., financial business impact analysis (BIA), SLA languages in general (such as Schmidt SLA approach, WS-Agreement), and extrapolation techniques used for predicting service usage (ExtraPol).

The specification of SLAs is fairly good covered by various SLA languages,

requirement	related work assessed		
	BIA	**SLA langs**	**ExtraPol**
R0.1 - integration	+	++	+
R0.2 - genericity	+	++	++
R0.3 - manageability	+	++	+
R3.1 - SLA penalty costs	+	++	n/a
R3.2 - integration with future service usage	+(a)	0	++(a)
R3.3 - further financial impact	+(a)	0	n/a
R4.1.1 - granularity of recovery actions	+(a)	n/a	n/a
R4.1.2 - scheduling of recovery actions	+(a)	n/a	n/a
R4.1.3 - duration of recovery actions	+(a)	n/a	n/a
R4.1.4 - effort used for recovery actions	+(a)	n/a	n/a
R4.1.5 - costs for recovery actions	+(a)	n/a	n/a
R4.2 - derivation of post-recovery impact	+(a)	n/a	n/a

Table 3.5: Assessment of related work concerning the modeling of business impact and related recovery actions (R3 and R4)

which basically all define constraints about services/functionalities and their qualities, together with appropriate cost and penalty definitions. For reuse of such languages it is only to note that the specified SLA constraints should uniquely refer to the service functionalities and their quality covered by the modeling mentioned above for the assessment of R1. The SLA approach of [Sch00, Sch01], treated on p. 99 in Sect. 3.5.1, is one possible example which very good fulfills this requirement, as is it based on the MNM service model. Nevertheless, any other SLA language may be used instead. Specifically, in the example scenario only single logical constraints about the QoS parameter of the respective service functionalities have been used instead.

General concepts, methods, and databases used today for financial business impact analysis may be adapted to provide information about further business degradation types beyond SLA violation costs. In part, this information may be specified as business policies which can be retrieved from appropriate policy repositories.

Specifically, business degradation types based on the estimation of future service usage can be specified and calculated with extrapolation techniques from historic and current data.

related work for the implementation of the tasks for I/RA (R5.1)

Table 3.6 illustrates the assessment of related work reused or adapted for the implementation of the tasks of I/RA (requirement R5.1), i.e., of the reasoning techniques about dependencies introduced in Sect. 3.6. These techniques are the deductive database approach (DDB), Transaction Logic (TrLogic), and Case Based reasoning (CBR).

In general, any of these implementation techniques is totally generic, and therefore has to be adapted in order to be used for implementing any of the I/RA tasks. Specifically, in order to fulfill integration as well as manage-

requirement	**related work assessed**		
	DDB	**TrLogic**	**CBR**
R0.1 - integration	+(a)	+(a)	+(a)
R0.2 - genericity	++	++	++
R0.3 - manageability	+(a)	+(a)	+(a)
R5.1.a - impact analysis	++(a)	++(a)	+(a)
R5.1.b - recovery recommendation	-	++(a)	++(a)
R5.1.c - customer notification	+(a)	+(a)	+(a)
R5.1.d - recovery tracking	+(a)	+(a)	+(a)
R5.1.d - model adaption	0	++(a)	++(a)

Table 3.6: Assessment of related work for the implementation of the tasks of I/RA (R5.1)

ability, appropriately coordinated exchange of corresponding model data with the the existing service management and provisioning environment have to be provided. As already discussed in Sect. 3.6, the deductive database approach is well suitable for impact analysis, and other tasks for which deductive reasoning, i.e., the deriving of facts which are valid because of existing facts and rules, is sufficient. Such other tasks may be customer notification and recovery tracking. But for recovery recommendation, and maybe model adaption (depending on the complexity for performing it), deductive reasoning is not sufficient. These two tasks, at any rate recovery recommendation, are involved with a kind of planning. Planning normally needs to potentially propose new facts, e.g., one concrete recovery plan with explicit scheduling, and test the consistency with already existing facts, e.g., current resource degradations, and rules, dependencies of degradations as well as dependencies of potential recovery plan templates/recovery actions to reduced resulting post-recovery impact. That is why recovery recommendation and potentially model adaption may be more appropriately realized by Transaction Logic or Case Based Reasoning, which are both suitable for planning (compare Sect. 3.6).

summary

No particular approach (nor suitable products) exists that allows to support I/R analysis completely in appropriate granularity and detail: Either they lack genericity, i.e., are too specific to a certain technology or type of scenario, or in terms of the service scenario too specific to a certain type of resources or the management technologies, or they do not take cover all modeling and workflow requirement classes (R1 to R5 in Sect. 2.4), at least concerning granularity and level of detail.

But for each particular requirement class, R1 to R5, there are some promising approaches which can serve as basis for an I/R analysis framework. Generally speaking, such approaches have to be adapted, refined, and above all integrated with each other for I/R analysis. Mainly these approaches are the MNM service model (together with its instantiation methodology), the SMIB approach based on it, as well as the Garschhammer approach in combination

with the specification language SISL (or alternatively the QoS specification language Qual). In addition to that, in general definitions and concepts used today in financial business impact analysis, any suitable SLA specification language, and methods for extrapolating future service usage may also contribute to a basis for I/R analysis.

integrated framework needed

Concluding, an integrated I/RA framework has to appropriately adapt and combine all these existing approaches/concepts to allow for unified, integrated I/R analysis, as well as to provide access to interfaces for all the information necessary in a coordinated and consistent manner. In particular, the development of the framework has explicitly has to include (compare above):

- an appropriately detailed workflow description covering the tasks of I/R analysis (for R5.1).
- an appropriately refined and integrated modeling to cover dependencies of service functionalities and resources (in appropriate granularity), including support for aspects like redundancy and load-balancing, in combination with dependencies of degradations for resources and functionalities, i.e., in combination with definition, measurement, and mapping of QoR/QoS parameter (value ranges) of resources/functionalities.

Chapter 4

Impact Analysis and Impact Recovery Framework

Contents

Here, the core of the thesis, the generic framework for impact and recovery analysis is developed. Sect. 4.1, as an overview, introduces the approach for the development of the framework in general, while in Sect. 4.2 to Sect. 4.5 the subsequent steps of this development are performed and discussed.

4.1 Idea and Approach Taken

The issue of this thesis is to develop a framework to support the performing of I/R analysis with as much automation as possible. In the following, this whole framework is referred to as the *impact and recovery analysis framework (I/RAFw)*.

In chapter 2 necessary requirements for such a framework were identified. The complete framework developed here should lastly cover all these requirements.

generic framework and its instantiation methodology

Because the framework should be applicable for any given service provisioning scenario, it will, as already indicated in chapter 2, consist of the following parts: a generic part and an instantiation methodology to apply this generic part to any given scenario. The generic part (or the instantiated version of this) has to fulfill all above identified requirements: The requirement classes service modeling, service degradation modeling, business (financial/reputational) impact modeling and recovery action modeling are concerned with the modeling of some specific pieces of information for the I/R analysis, whereas the requirement class workflow modeling is specifically concerned with the actual run of I/R analysis by using modeling information fulfilling the first four requirement classes.

model part and workflow part

Therefore it is reasonable to divide the framework into a model part (concerned with the first four requirement classes) and a workflow part. Consequently, the generic part of the framework is consisting of a generic model and a generic workflow, which both can be instantiated by the instantiation methodology to a specific model and a specific workflow.

To sum it up, the framework will comprise these parts:

- (1) generic part:
  - (1a) a generic model for all information required for I/R analysis
  - (1b) a generic workflow for performing I/R analysis by using the model (1a) and specified interfaces with existing management concepts/areas/tools
- (2) a instantiation methodology to apply the generic model (1a) and the generic workflow (1b) to a given specific scenario

In this chapter the generic part, the generic impact and recovery analysis framework (I/RAFw), is covered, whereas its instantiation methodology (I/RAFw InstMeth) is treated in Sect. 5.

Actually, the generic framework I/RAFw, is consisting of four framework parts, which are accordingly developed in four consecutive, specific steps. Fig. 4.1 illustrates these specific steps for the approach for the development of the I/RAFw (*I/RAFwAppr*). They are introduced in the following.

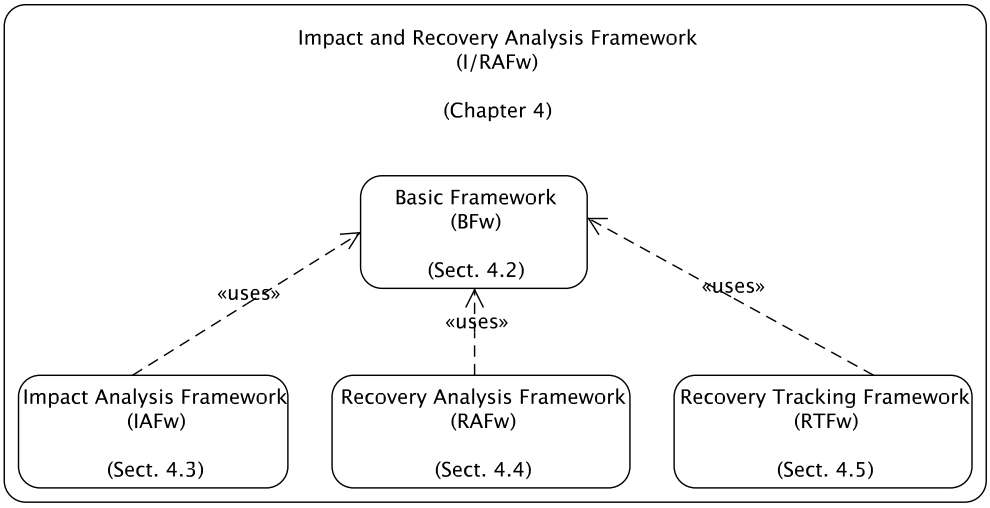

Figure 4.1: Approach for the development of the I/RA framework: basic framework and its extensions

parts of I/RAFw - steps its development

At first, the *basic framework (BFw)* is introduced in Sect. 4.2. This serves as a generic basis for all following extensions. That is why it is concerned with all 3 phases of I/R analysis, i.e., impact analysis, recovery analysis, and recovery tracking, but only to be covered in a generic manner. From Sect. 4.3 to Sect. 4.5, extensions of the basic framework are developed and discussed, each covering one of the specific phases. So there are the following extensions of the basic framework: the *impact analysis framework (IAFw)* discussed in Sect. 4.3, the *recovery analysis framework (RAFw)* covered in Sect. 4.4, and the *recovery tracking framework (RTFw)* treated in Sect. 4.5. There, each extension framework is introduced and developed in an iterative manner, covering the set of requirements related to it in a step-by-step manner. Finally, in Sect. 4.6 the complete development of the framework is summarized.

development in a particular step

Concerning a particular one of these four steps for the development of I/RAFw, the following general statements can be made: In general, it comprises a workflow specification and a corresponding data model, at least it extends and refines the ones of the basic framework. In turn, the workflow specification comprises a set of consecutive workflow activities, i.e., modeling and processing interactions to be performed, which have various input and output artifacts. The used artifacts may comprise parts of the data model, as well as additional input/output data necessary. For each workflow activity it specified which (mostly already existing and reused) components, concepts, or techniques are used for the actual realization of the activities.

Moreover, for each particular step of the development the workflow specification may be designed iteratively, distinguishing an abstract, conceptual workflow, and a realization of this abstract workflow, a realization workflow. For the conceptual workflow only the activities and the used artifacts are considered in an abstract manner. Based on this, for the realization workflow it is specified, in concrete and detailed manner, which (ideally already existing reused) techniques, components, and concepts as well as parts of the data model are used to actually perform the workflow activities.

The following list summarizes the relevant issues which are addressed in this chapter for each step of the development:

- data model: for all information involved, as far as required for I/R analysis.
- workflow: (ideally existing reused) components/techniques and activities (steps), i.e., interactions of components with necessary input/output artifacts, designed in an rough abstract, as well as a detailed, concrete way.

4.2 Basic Framework

In this section the basic framework for I/R analysis (compare Fig. 4.1) is introduced and discussed. This basic framework (BFw) will later on (Sect. 4.3 to Sect. 4.5) be extended to eventually fulfill all requirements identified in Sect. 2.4 with appropriate granularity and sufficient level of detail.

Fig. 4.2 illustrates the approach for the development of this basic framework (BFwAppr). Because it has to constitute a consistent and flexible basis to allow for later extensions, it is developed as follows:

three refinement steps for basic I/RA workflow

First, a *basic abstract workflow (BAWf)*, described in an abstract and rough manner, is introduced in Sect. 4.2.1. This abstract workflow is consisting of a set of *basic abstract workflow steps* and related *basic abstract input and output artifacts*. Actually, two refinement steps of this workflow, BAWf, will be developed as explained below.

first refinement of basic workflow

Following, from Sect. 4.2.2 to Sect. 4.2.4 the basic abstract workflow is refined: Consecutively for each of its basic workflow steps, the abstract output and input artifacts are analyzed and refined in detail, which further allows a refinement of the workflow step. Altogether, these refinements of the individual steps add up to the *basic refined abstract workflow (BRAWf)*.

basic component architecture for executing basic workflow

In Sect. 4.2.5 a *basic component architecture (BCArch)*, which is capable of executing the basic refined abstract workflow (BRAWf), is introduced: First, *basic external interfaces (BExtIfcs)* to existing management tools, data bases, or other data sources are identified, which are used as data sources/destinations of input/output artifacts.

Second, the generic *basic component architecture* itself is specified. It is consisting of multiple, interacting, *basic internal components (BIntComps)*, which are realized by concrete, existing and reused or newly developed concepts, tools, techniques.

In fact, the specification of basic external interfaces as well as - where possible - the specification of internal components is reusing existing management tools, platforms, concepts. This is done in order to ensure that the I/RA framework as a whole will easily integrate in the existing service management and provisioning infrastructure of a service provider, and that it will be easy to use and will be really of help for its operators and practitioners actually faced with I/R analysis.

second refinement of basic workflow

Afterwards, the refined abstract workflow will be mapped to the detailed *basic realized workflow (BRWf)*, which will be implemented using the before specified component architecture.

This realized basic workflow performed by the basic component architecture using the basic external interfaces is the basis for all further refinement, concretization, and extension discussed and introduced in the extension frameworks (Sect. 4.3 to Sect. 4.5).

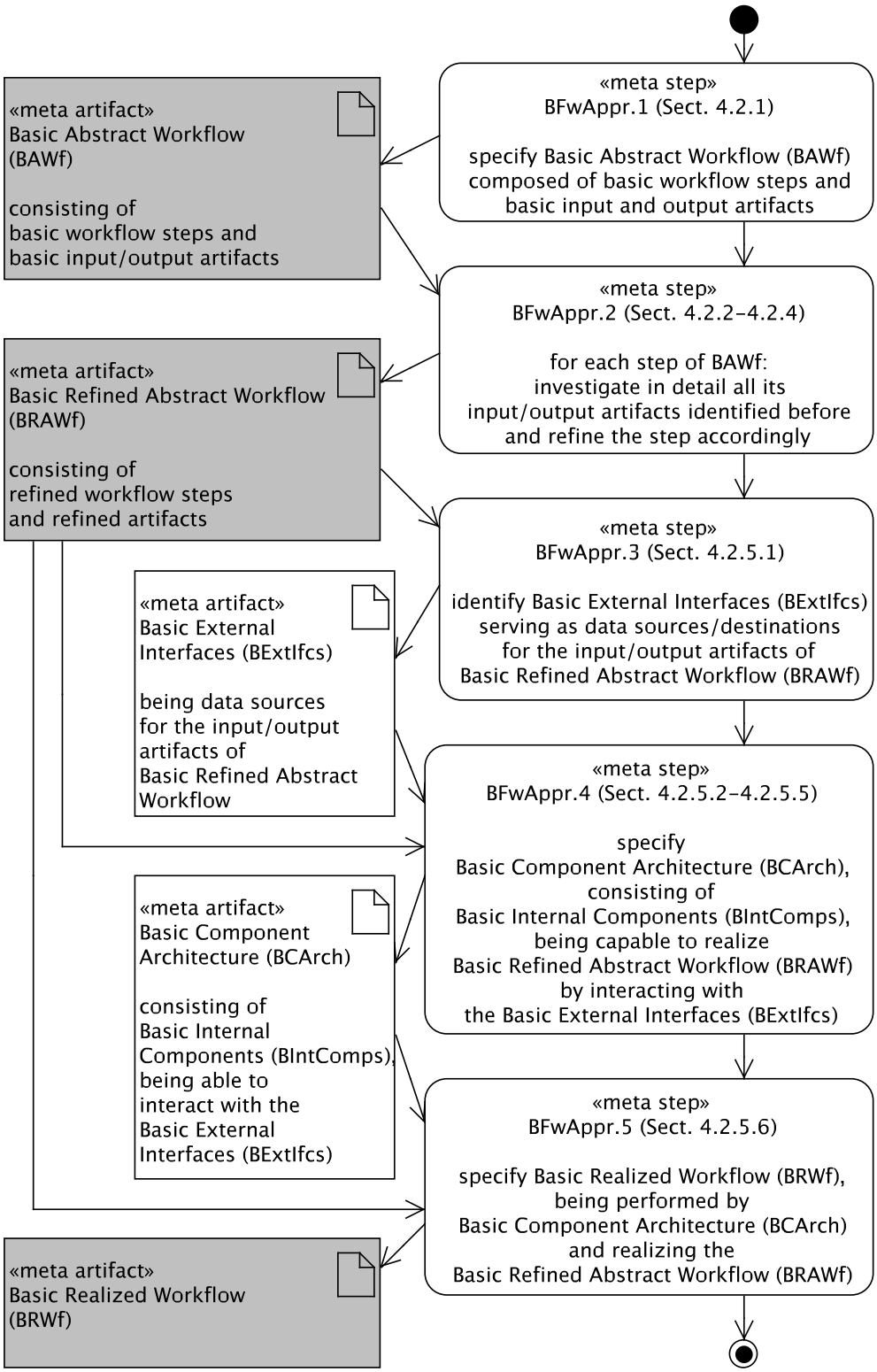

Figure 4.2: Approach (BFwAppr) for the development of the basic framework (BFw)

summary

Concluding, the basic framework comprises three refinement steps for a workflow for I/R analysis, namely the basic abstract workflow (BAWf, Sect. 4.2.1), the basic refined abstract workflow (BRAWf, Sect. 4.2.2 to Sect. 4.2.4, comprising three subworkflows, one for each step of BAWf), and the basic realized workflow (BRWf, Sect. 4.2.5.6, comprising multiple subworkflows, especially one for each step of BRAWf). In addition to that it introduces a basic component architecture (BCArch), consisting of basic external interfaces (BExtIfcs) and basic internal components (BIntComps), which can used to actually perform the basic realized workflow (Sect. 4.2.5).

4.2.1 Basic abstract workflow

Here the abstract version of the *basic workflow (BWf)*, the *basic abstract workflow (BAWf)*, is introduced. For this purpose abstract activities and abstract input/output artifacts are specified as well as a general description how these artifacts are derived from and mapped to each other. However, here only a general, abstract idea of this mapping will be given without detailed and specific mapping information. Too complex, detailed data sources are also not specified here. Rather, the specific details of the artifacts regarded from an abstract level are treated afterwards in Sect. 4.2.2 to Sect. 4.2.5, resulting in the refined realization of the abstract workflow BAWf, the *basic realized workflow (BRWf)*, discussed in Sect. 4.2.5.6. Moreover, the specific and detailed data structures necessary for completely realizing these artifacts and for integrating them with the provider's service provisioning/management platform are eventually fully covered in the particular extension frameworks in Sect. 4.3 to Sect. 4.5.

requirements for the workflow

In general the refined and realized version of this workflow will have to comply to the requirements of Sect. 2.4, especially the ones related to the course of the I/R analysis in general (R5.1 to R5.3). Especially R5.1 (see Sect. 2.4.7), i.e., the tasks eventually to be performed by the I/R analysis, which are impact analysis, recovery analysis, recovery tracking, customer notification, and modeling adaptation, suggest the idea to separate the whole workflow into three subsequent steps or *subworkflows*: The first of the two subworkflows will be concerned with one of the first two tasks each (impact analysis, recovery analysis). The third will be concerned with recovery tracking, including the related tasks of customer notification and modeling adaptation.

steps of basic abstract workflow

Fig. 4.3 shows an overview of the resulting basic abstract workflow of I/R analysis BAWf consisting of the three subworkflows. In detail, these subworkflows are:

- BAWf.1: impact analysis subworkflow: determine impact on services, service instances and customers (service impact) and on the business of the provider (e.g., SLA violation costs as a first step; business impact);
- BAWf.2: recovery analysis subworkflow: recommend recovery action to perform and determine the resulting reduced impact of this recovery ac-

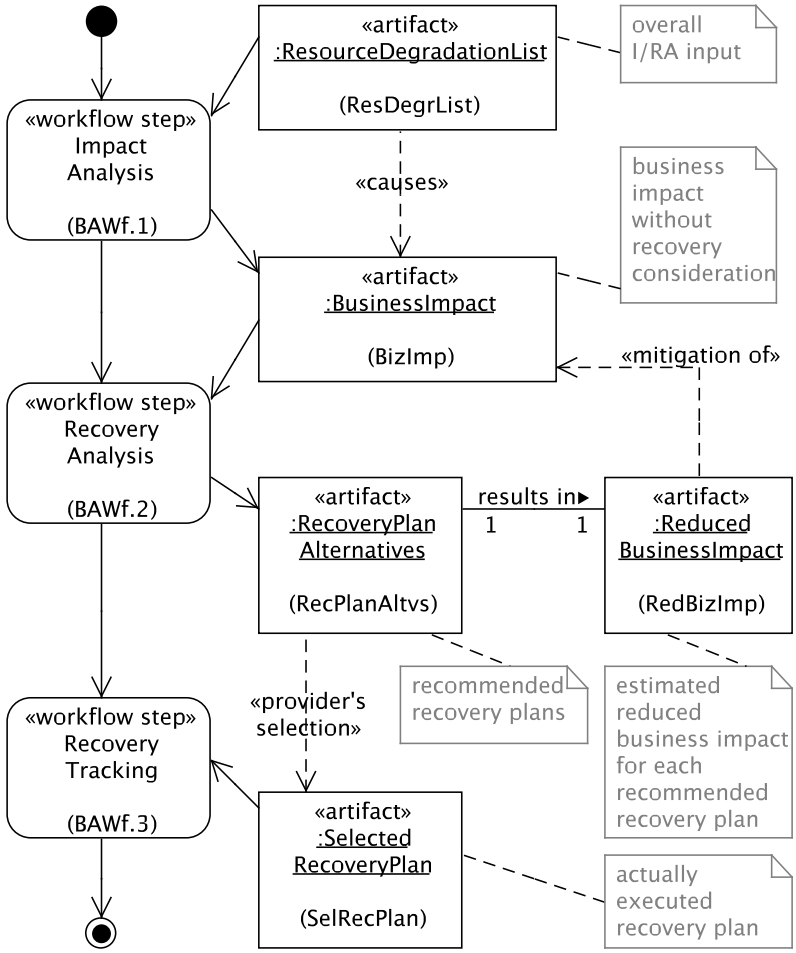

Figure 4.3: Overview of Basic Abstract Workflow of I/R analysis (BAWf)

tion; reduced impact should eventually take into account remaining impact (with more specific duration of service degradation), but also costs and effort of recovery action;

- BAWf.3: recovery tracking subworkflow: track the actual recovery performed (even if it is not the one recommended by the recovery analysis), while customers are kept informed. Additionally, adapts modeling if actual recovery performed has influenced the modeled services;

The first subworkflow performs the impact analysis (*basic abstract impact analysis subworkflow*), and it has a list of current/assumed resource degradations as input and business impact as output artifact. Here no recovery is considered yet. The second subworkflow is concerned with the recovery analysis (*basic abstract recovery analysis subworkflow*), and has as output recommended recovery alternative as well as the estimated reduced impact resulting if this recovery alternative is actually performed. Finally, the third subworkflow performs the recovery tracking of the actual selected and executed recovery alternative (*basic abstract recovery tracking subworkflow*), be it one having been recommended or even a completely different one.

relationship to framework extensions

Furthermore, it is to be said, that this partitioning of the workflow also exactly corresponds to the subdivision of the extension of the basic framework: Sect. 4.3 will be concerned with the refinement and specific realization of the impact analysis, i.e., impact analysis framework, and so will be concerned and refine the impact analysis workflow. Correspondingly, Sect. 4.4 discussing the recovery analysis workflow will refine the recovery analysis workflow, and likewise Sect. 4.5 will refine the recovery tracking workflow.

representation of BAWf

Actually, in Fig. 4.3 BAWf is depicted as an UML activity diagram. Subworkflows are represented by UML activities. Input/output artifacts in general are represented by UML object instances with the stereotype “artifact”. Complex structure and the relationship of multiple artifacts which are the output of the same subworkflow are indicated by UML links (instances of UML associations) between the respective object instances. An example is the “results in” link between “RecoveryPlanAlternatives” and ReducedBusinessImpact”, both being output of activity “Recovery Analysis”. In contrast, relationships between separate input or output artifacts of different subworkflows, which are only semantical and not structural in nature (concerning artifact structure), are indicated by UML dependencies with appropriate stereotypes: An example is the “causes” dependency between the object instances “ResourceDegradationList” and “Business Impact”. The two latter ones are separate outputs of the activities “Impact Analysis” and “Recovery Analysis” respectively, but with the (only) semantical relationship that the latter one specifies (i.e., business impact) what is *caused* by the specification of the former one (i.e., resource degradation list). In contrast, the “results in” link (association instance) between the object instances “RecoveryPlanALternatives” and “ReducedBusinessImpact” denotes a structural relationship between the two object instances which both are output of the activity “Recovery Analysis” as well as input of the activity “Recovery Tracking”.

artifacts of the workflow

The input/output artifacts shown in Fig. 4.3 are the *artifacts essential for the whole workflow* BAWf. In the following analysis of the particular subworkflows the detailed structure of these workflow-essential artifacts is treated. Moreover, for each subworkflow further input/output artifacts may be introduced and equally treated which are actually *only essential for the specific subworkflow*, not essential for the whole workflow. Beyond this, after the analysis of the input/output artifacts essential for a particular subworkflow, even further so-called *additional artifacts* may be introduced, which actually determine how to convert essential input artifacts into essential output artifacts for the particular subworkflow.

representation of subworkflows

Done in parallel to this analysis of input/output artifacts for a subworkflow, each subworkflow is refined accordingly. Refinement of a subworkflow is represented by refinement of the respective UML activity in Fig. 4.3. For such refinements, UML activities denote particular steps of the subworkflow in question, not the subworkflow as a whole. For representation of input/output artifacts in these refinements the rules described above for artifacts of BAWf are applied.

4.2.2 Basic abstract subworkflow for impact analysis

The subworkflow for the impact analysis BAWf.1 (first step of Fig. 4.3) is analyzed and treated in more detail here, although this analysis stays on an abstract level. The result of this analysis is BRAWf.1 as refinement of BAWf.1.

4.2.2.1 Analysis of essential artifacts

basic situation for impact analysis

Basically, the purpose of impact analysis is to determine the impact from degradations of resources used for the service realization on the business of the provider. Fig. 4.4 illustrates the basic situation for impact analysis, i.e., the derivation of service impact and business impact out of resource degradations given.

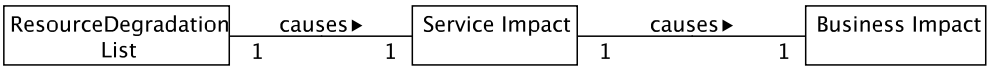

Figure 4.4: Service impact and business impact caused consecutively by resource degradations

Initially, a list of resource degradations, i.e., degradations of resources used for service realization, is given. Each of these resource degradations may entail some negative consequences for the functioning, the quality, and the access of dependent services, as seen from the point of view of the customers and users of these services. In turn, this may entail negative consequences for the business concerning financial or reputational aspects, arising from the degradations of the services, e.g., SLA penalty costs (compare requirements R3.1 to R3.3).

The term *service impact* is used to subsume all the negative consequences on the services as experienced by users or customers, i.e., concerning access, functionality, particular quality. Each of these negative consequences for the functioning, the quality, or the access of a service functionality for a specific set of users and customers is labeled by the term *service degradation*. Here, the term degradation ranges from partial, minor quality degradation, temporary or transient short-time unavailability, high quality degradations, to total unavailability. As already said, a service degradation describes the degradation from the point of view of users or customers. Furthermore, the negative consequences on the business concerning finances and reputation entailed by the service impact are subsumed under the term *business impact* and are individually called *business degradations*. Examples for business degradations are SLA penalty costs, revenue loss, or an increase in the number of customers canceling contract (requirement class R3). Each of them may be further subdivided (e.g., revenue loss per individual service, SLA penalty costs per single contract) where it is appropriate.

situation for impact analysis, refined

So in general, one or multiple resource degradations entail a service degradation, and one or multiple service degradations entail a business degradation. All business degradations (indirectly) entailed by one or multiple of

the resource degradations initially given, represent the business impact to derive. In the recovery analysis performed after the impact analysis, these business degradations entailed by a resource degradation will be used to evaluate and prioritize the resource degradation and to plan and schedule its recovery. Fig. 4.5 illustrates this refined situation for impact analysis, i.e., the derivation of service degradations and business degradations consecutively entailed by resource degradations.

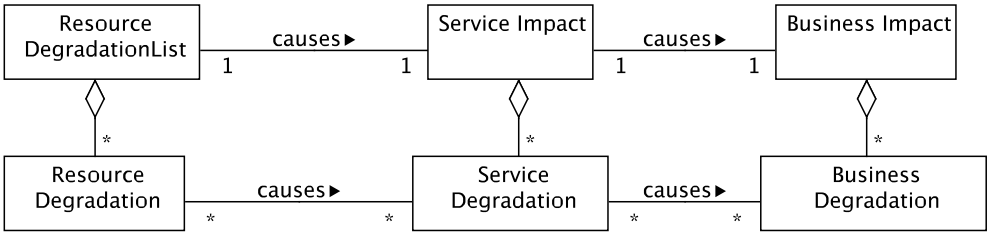

Figure 4.5: Service impact and business impact composed of service degradations and business degradations caused consecutively by resource degradations

example situation for I/R analysis

In the following, an example situation ($ExSit1$), which is taken from example I/R A run of Sect. 2.3.4, is introduced. Later on, this example situation will be used for further illustrations, too. Fig. 4.6 illustrates the example situation. In fact, it is a slight extension of the situation presented in Sect. 2.3.4. Here, the service provider LRZ is faced with three simultaneously occurring resource degradations: resource degradations g_{r1} and g_{r2} from Sect. 2.3.4, and addition a further resource degradations g_{r1b}.

Similar as in Sect. 2.3.4, resource degradation g_{r1} (high utilization of the particular ip link $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$, having risen above 60% permanently; compare Fig. 2.9) entails service degradation g_{s1-1} (mail sending delay for all users degraded: mail sending delay avg. $5.5\ \mathrm{min}$), which in turn entails business degradation g_{b1-1} (SLA penalties defined by $sla\_pnlty_{\mathrm{mail3}}$ for violating corresponding SLA constraint $sla\_cnstr_{\mathrm{mail3}}$, which prescribes avg. delay $\leq 5\ \mathrm{min}$). But, here an additional resource degradation g_{r1b} (long average DNS server processing time of avg. $15\ \mathrm{s}$ per request, instead of normally avg. $< 1\ \mathrm{s}$ per request) is also contributing to g_{s1-1}, making its degraded value range even worse, resulting in an avg. total sending delay of $6\ \mathrm{min}$. Yet, business degradation g_{b1-1} is not becoming worse, as its corresponding penalty definition ($sla\_pnlty_{\mathrm{mail3}}$) is not taking into account the specific value of the exceed of the delay above $5\ \mathrm{min}$ ($6\ \mathrm{min}$ instead of $5.5\ \mathrm{min}$ without g_{r1b}).

Furthermore, the additional resource degradation, g_{r1b} causes a service degradation (g_{s1-2}) of the web page access functionality, namely an increase of the average web page delivery delay (avg. $18\ \mathrm{s}$, for web pages of up to $100\ \mathrm{Kb}$, instead of usually $3\ \mathrm{s}$). As the resulting value range is still within the defined SLA ranges ($\leq\ 20\ \mathrm{s}$ for premium customers, and $\leq\ 30\ \mathrm{s}$ for normal customers), this causes no business degradation (as only SLA penalties are

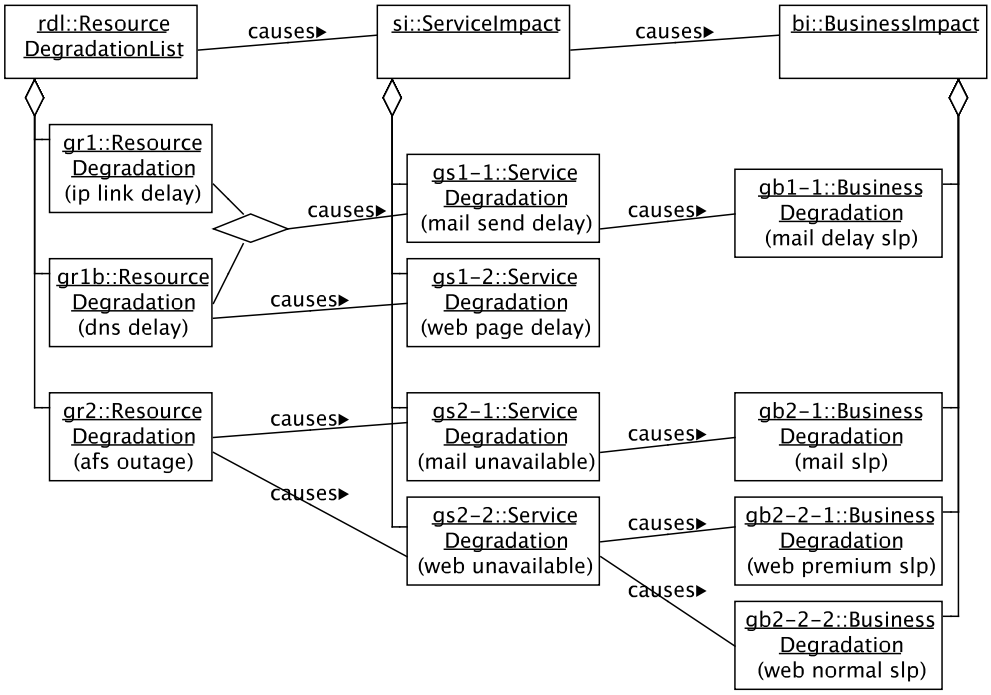

Figure 4.6: Example situation ($ExSit1$) comprising service degradations and business degradations caused consecutively by resource degradations[1]

considered as business impact for this example), only some annoyance for users (not covered here as business impact).

Moreover similarly as in Sect. 2.3.4, resource degradation g_{r2} (AFS storage outage resulting in unavailability of various AFS file paths) results in the two service degradations g_{s2-1} (mail sending completely unavailable for particular customers) and g_{s2-2} (web page access completely unavailable for particular customers). g_{s2-1} results in business degradation g_{b2-1} (SLA penalties for violating constraints sla_cnstr_{mail1} and sla_cnstr_{mail2}), and g_{s2-2} results in the business degradations g_{b2-2-1} (SLA penalties to normal web service customers - constraints $sla\_norm\_contr_{web1}$ and $sla\_norm\_cnstr_{\text{web2}}$) and g_{b2-2-2} (SLA penalties to premium web service customers - constraints $sla\_prem\_cnstr_{\text{web1}}$ and $sla\_prem\_cnstr_{\text{web2}}$).

Concerning details of the slp (SLA penalties costs) mentioned above, compare their definitions on p. 51 in Sect. 2.3.4.

details/issues of degradations

In order to be useful, for the selection and scheduling of recovery measures in appropriate granularity/accuracy (requirement class 4) later on, the business degradations derived by the impact analysis have to be described with appropriate granularity and with sufficient detail level. To allow for this, business degradations are ideally described as functions of time or duration, e.g., the SLA penalty costs for a particular service and for a particular customer over a specific time/duration. So, a description of a business degradation has to sat-

[1] slp=SLA penalty

isfy two requirements: the time granularity has to be selected appropriately and the value granularity per time unit has to be selected accordingly. In order to specify and derive business degradations with these requirements, it is also necessary to distinguish the different resource degradations and the different service degradations in sufficient and appropriate granularity and detail level.

Therefore, where necessary, the description or specification of a resource degradation has to include details like the affected resource, the specific degraded resource quality parameter and its actual value range, the time, the (estimated) duration, and the temporal course of the degradation. Similarly, the specification of a service degradation has to specify details like the specific service functionality affected, the specific functionality parameter value sets for which the service is degraded (particularly the specific service instances, i.e., users and customers affected), the specific affected service quality parameters and its actual degraded value range, the time, the (estimated) duration, and the temporal course of the degradation. Furthermore, in addition to information about time and duration, the specification of a resource degradation or a service degradation may include information about the specific failure pattern, where this is necessary for determining the entailed business impact in sufficient detail. The failure pattern describes the actual or possible course of recurrent occurrences of specific a degradation, e.g., by being characterized as random, transient, intermittent, permanent, or even by more complex means as e.g., statistical distributions. In fact, all these differentiations are related to the requirement classes R1 and R2.

The actual identification of the complete set of details necessary to describe and design the corresponding data structures used to specify resource degradation, service degradations, and business degradations, is actually performed in the Impact Analysis Framework (Sect. 4.3), where these aspects are discussed and treated iteratively. For the abstract purpose of the Basic Framework here, the above examples of details shall be sufficient to illustrate and motivate the notions of resource degradation, service degradation, business degradation, and their inter-relationships. In general, the following may be said about them: Business degradations have to be specifiable and to be determinable with sufficient level of detail, concerning the time granularity/accuracy as well as the value granularity/accuracy per time unit. In order to support this, resource degradations as well as derived service degradations have also to be specifiable and to be determinable with sufficient level of detail concerning the granularity/accuracy of the time domain as well as the granularity/accuracy of the value domain. The time domain for resource/service degradations is concerned with information about time/duration and failure pattern/temporal course of the degradation, and the value domain is concerned with a sufficient subdivision and differentiation of resource degradations and service degradations per time unit, taking into account e.g., different functionalities, different functionality parameter values (e.g., different customers/users), different quality parameters, and different levels/value ranges of a quality parameter.

Summing up, this leads to the following abstract definitions in Table 4.1:

abstract definition of business, service, resource degradation

business degradation: information about a specific financial/reputational consequence on the business of the provider entailed by one/some service degradations (in turn entailed by resource degradations), as a function of time/duration, in appropriate detail/accuracy/granularity concerning time domain and value domain (as necessary for recovery analysis later-on).

service degradation: information about a specific consequence on a service entailed from one/some resource degradations, as experienced by customers/users, in appropriate detail level/accuracy/granularity as necessary for deriving business degradations in appropriate granularity/accuracy.

resource degradation: information about a degradation of a resource used for service realization, in appropriate detail level/accuracy/granularity as necessary for deriving service degradations in appropriate granularity/accuracy.

Table 4.1: Abstract definitions for business degradations, service degradations, and resource degradations

Using these abstract definitions the essential input and output artifacts of the impact analysis subworkflow are described and characterized as follows in Table 4.2:

abstract definition of the basic input and output of impact analysis

input of impact analysis: list of resource degradations, currently occurring or only assumed.

output of impact analysis: derived business degradations together with the service degradations entailing them and in turn with resource degradations entailing the service degradations; in order to allow for evaluation, prioritization, and recovery scheduling of the initially given resource degradations by evaluation of business degradations derived from them.

Table 4.2: Abstract definition of essential input and output for impact analysis

As already stated above, the specific details and data structures to describe and specify business degradations, service degradations, and resource degradations are investigated and developed in the Impact Analysis Framework in Sect. 4.3. Nevertheless as also motivated and discussed above, some generic statements and definitions concerning the data structures necessary to specify the particular types of degradations for I/R analysis can be stated here in abstract manner. Fig. 4.7 illustrates these aspects and issues, which are given as conclusion and summary of the example descriptions of degradations given

abstract definition of details/specific issues of degradations

above. Information describing any particular of the three types of degradation has to cover in appropriate *granularity* and *accuracy* the *time domain* as well as the *value domain*:

details of business degradations

For business degradations, the time domain has to cover issues concerned with the time granularity of time/duration, as well as periodicity and temporal course of the degradations. The granularity of the value domain for business degradations deals with an appropriate subdivision of the overall business impact into individual business degradations. The accuracy of the value domain for business degradations is concerned with measures/metrics for business degradations per time unit which are accurate and detailed enough.

details of resource/service degradations

The time domain for resource as well as for service degradations is concerned with similar issues as the time domain for business degradations, namely time granularity, time periodicity and temporal course in general. Also similarly, the issues for the value domain of resource and service degradations are subdivided into granularity and accuracy of the value domain.

granularity of value for resource/service degradations

About the granularity of the value domain some differentiation are made already on this abstract level here, for both, resource degradations as well as service degradations. In general, concerning the granularity of the value domain for resource/service degradations the degradation subject as well as the manner (or type) of the degradation have to be considered:The *degradation subject* identifies *what is degraded*, e.g., which resource or which service (functionality) for which customers/users. Moreover, the *degradations manner* further specifies *how, i.e., in which way or manner*, the degraded subject is actually degraded. For the degradation of a particular resource (as a degradation subject) different degradation manners can be e.g., identified with different QoR/QoD parameters (or sets of QoR/QoD parameters) which are actually degraded. Similarly, for a degraded service functionality (as degradation subject), different degradation manners can be identified with different QoS parameters (or sets of QoS parameters).

Concluding, the issues concerning the granularity of the value domain for resource/service degradations, are classified by a subdivision into different degradation subjects, (answering *"what is degraded"*) with different degradation manners (answering *"in which manner is the subject degraded"*) each. Whereas, the granularity of value domain for resource/service degradations is concerned with the particular degree of the degradation, i.e., the question *"to what degree"* or more exactly *"to what degree is the specific degradation subject degraded in the specific degradation manner"*. Specifically, the distinction of different degradation subjects is mainly related to requirement class R1, while the distinction of different degradation manners is related to requirement class R2.

accuracy of value for resource/service degradations

The accuracy of the value domain is concerned with an appropriate value measure of a degradation per time unit per degradation subject per degradation manner: For resource degradations, this can e.g., be done by the specification of values conforming to a particular QoR/QoD metric (and corresponding value ranges) for each potential combination of degradation subject (the

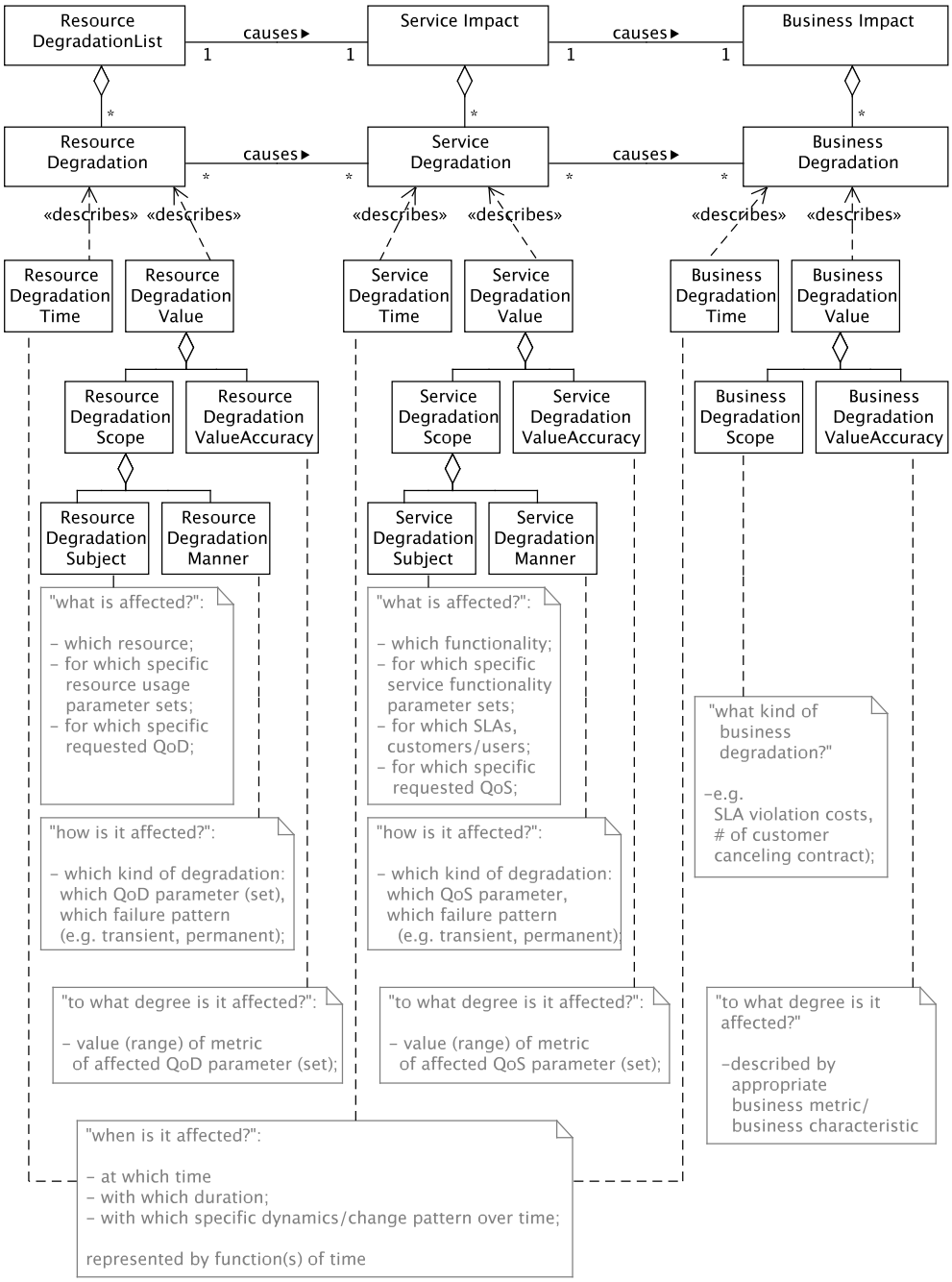

Figure 4.7: Abstract parts of information necessary to describe, specify and subdivide initial resource degradations, entailed service degradations, as well as entailed business degradations

Degradation	Value			Time (during regarded time interval)
	Scope		Value Accuracy	
	Subject	Manner		
g_{r1}	$r_{\text{iplink}}(r_{\text{rt\_lrz}}, r_{\text{r\_sw\_2}})$	high link utilization	$> 60\%$	permanently
g_{r1b}	$r_{\text{dns\_sv1}}$	long DNS processing time	avg. 15 s, ca. $2/3$ of requests	randomly,
g_{r2}	$r_{\text{afs\_sv1}}(\textit{path} \in \text{PathListAfs}^1)$	AFS cell outage	fully unavailable	permanently
g_{s1-1}	$f_{\text{mail/use/send}}$	high mail sending delay	avg. 6 min	permanently
g_{s1-2}	$f_{\text{web/use/apage}}$	high web page delivery delay	avg. 18 s	permanently
g_{s2-1}	$f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{GrpMail}^2)$	unavailability	fully unavailable	permanently
g_{s2-2}	$f_{\text{web/use/apage}}(user \in GrpWeb^3)$	unavailability	fully unavailable	permanently
g_{b1-1}	SLA penalties defined by sla_pnlty_{mail3}		$slp_{\text{mail3}}(t)$	
g_{b2-1}	SLA penalties defined by sla_pnlty_{mail1} and sla_pnlty_{mail2}		$slp_{\text{mail1}}(t) + slp_{\text{mail2}}(t)$	
g_{b2-2-1}	SLA penalties defined by $sla\_prem\_pnlty_{\text{web1}}$ and $sla\_prem\_pnlty_{\text{web2}}$		$slp\_prem_{\text{web1}}(t) + slp\_prem_{\text{web2}}(t)$	
g_{b2-2-2}	SLA penalties defined by $sla\_norm\_pnlty_{\text{web1}}$ and $sla\_norm\_pnlty_{\text{web2}}$		$slp\_norm_{\text{web1}}(t) + slp\_norm_{\text{web2}}(t)$	

Table 4.3: Example of the parts of information describing degradations of example situation $ExSit1$ (compare p. 130)

1 $PathListAfs$ denotes the list of particular file paths which are affected by the AFS cell outage (g_{r2})
2 $GrpMail$ denotes the list of particular customers and corresponding users affected by the mail sending outage (g_{s2-1}; compare Sect. 2.3.4)
3 $GrpWeb$ denotes the list of particular customers and corresponding users affected by the web page access outage (g_{s2-2}; compare Sect. 2.3.4)

particular resource) and degradation manner (the particular QoR/QoD parameter (set)). Alternatively, instead of only values and value ranges (per time), suitable abstractions thereof may be used, such as more fuzzy types of value specifications using probability distributions or possibility distributions. For using the former suitable knowledge about the exact or estimated probability distribution of the metric values has to be in place (or measured), while the latter, being related to fuzzy logic, can be often be used to encode partially existing expert knowledge. Similarly, for service degradations the values/value ranges/abstractions thereof of QoS metrics can be used in a similar manner as the ones of QoR/QoD metrics for resource degradations.

Degradation subject and degradation manner, which both together characterize and determine the granularity of the value domain of resource/service degradation, are subsumed under the term *degradation scope*. Concerning the granularity of the value domain for business degradations, a similar differentiation into subject and manner could be done, but as the definition of business impact and business degradation is highly depending on the specific point of view and the specific requirements of the service provider, which has to define and specify them individually, such a differentiation on this abstract level is not done here. Instead, only the term *business degradation scope* is introduced, in analogy to the term degradation scope for resource/service degradations, concerned with the granularity of business degradation. Consequently, business degradation scope is concerned with the question *"what kind of business degradation is happening"*, whereas in contrast the accuracy of value of business degradation is concerned with the question *"to what degree is the specific kind of business degradation happening"*.

parts of degradation information for example situation

To provide an example, Table 4.3 on p. 136 gives an overview of the parts of information describing the various degradations of the example situation $ExSit1$ introduced above (initially entailed by the resource degradations g_{r1}, g_{r1b}, g_{r2}; see p. 130).

The examples given above and terms introduced above were given in order to provide a general overview and an abstract classification of the general aspects and issues which have to be taken into account when dealing with business degradation, service degradation, and resource degradations for the purpose of I/R analysis. As already said, the realization of these all abstract parts by refined and detailed data structures is actually performed and discussed in the Impact Analysis Framework in Sect. 4.3.

refinement of workflow with detailed basic input/output artifacts

After having analyzed the general situation for impact analysis, the first step of the basic abstract workflow, BAWf.1 (Fig. 4.3), is now refined, based on the analysis of the basic input and output artifacts, which was discussed above. In Fig. 4.8 this refined version of BAWf.1, namely BRAWf.1, is depicted.

In the basic abstract workflow in Fig. 4.3, impact analysis was only a single workflow step, having an essential basic input artifact, namely the list of initial resource degradations - the essential input of the overall I/R analysis, as well as an essential basic output artifact, namely the business impact. The abstract definition and the general issues concerning the structure of the artifacts, as

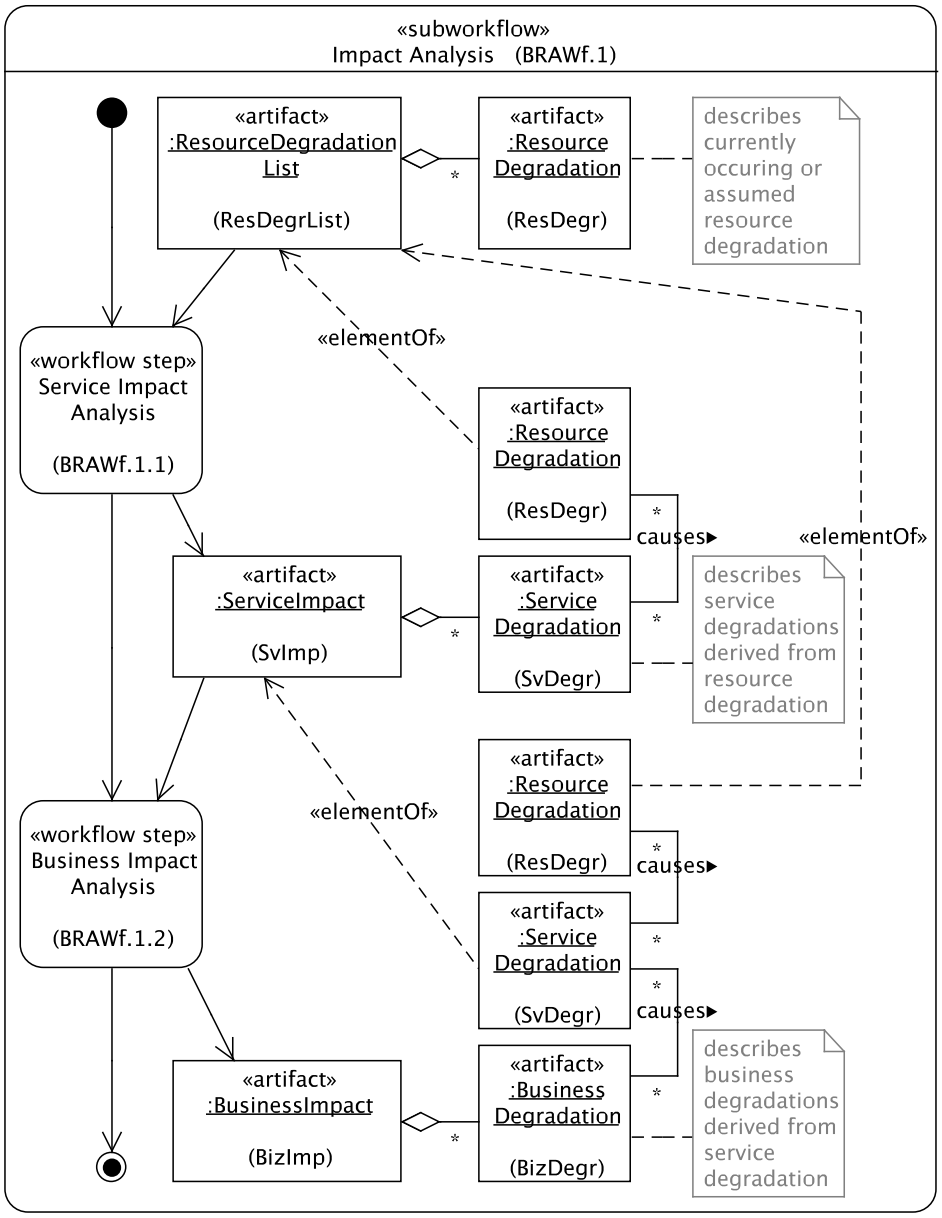

Figure 4.8: Basic refined abstract subworkflow of impact analysis with essential input and output artifacts (BRAWf.1, refinement of BAWf.1 in Fig. 4.3)

well as their inter-relationships have been discussed before (see the abstract definitions on p. 133). Furthermore, as a mediator between them, service impact was introduced and treated in detail, too.

subdivision of impact analysis

Therefore, the impact analysis, BRAWf.1, regarded as a subworkflow instead of a single step of BRAWf, is subdivided into two subworkflow steps: *service impact analysis* (BRAWf.1.1), and *business impact analysis* (BRAWf.1.2). The task of the former is deriving service impact from the initially given list of resource degradations, and the task of the latter is deriving business impact entailed from the service impact.

service impact as additional artifact

That is why between the two subworkflow steps, as additional workflow artifact, the derived service impact is added. It is the essential output artifact of the service impact analysis step, and is the essential input artifact of the business impact analysis step. So, the list of resource degradations, originally being the essential input artifact of the overall impact analysis BAWf.1/BRAWf.1, becomes particularly the input artifact of the service impact analysis step BRAWf.1.1. Likewise, the output artifact of overall impact analysis, i.e., the business impact, indirectly entailed from the resource degradations, becomes particularly the output artifact of business impact analysis step BRAWf.1.2.

The determination of the service impact as a single artifact is necessary, on the one hand because for the provider it is necessary to inform the customer side about the service impact of its subscribed services, and on the other hand the business impact in fact can only be derived based on the service impact. The reason for the latter is that I/R analysis in this work is concerned with business impact entailed by resource degradations via the impact on services.

SIA and *BIA* are introduced as abbreviations for service impact analysis and business impact analysis, respectively. For impact analysis in general the abbreviation *IA* is used.

all essential input/output of IA

Concluding, impact analysis actually deals with three essential artifacts, the list of resource degradations initially given as input for I/R analysis, the service impact, i.e., the service degradations entailed from the resource degradations, and finally the business impact, i.e., the business degradations entailed from the service degradations (for a detailed definition of the types of degradations regarding I/R analysis see definition boxes on p. 133).

relationship of essential artifacts

In Fig. 4.8, also in abstract manner, the structure of and relationship between these artifacts is given. The artifact for service impact consists of subartifacts for each derived service degradation together with a reference to the particular resource degradations from which it is entailed. Similarly, the artifact for business impact consists of subartifacts for each derived business degradation together with a reference to the list of the particular service degradations and in turn the list of particular resource degradations from which it is entailed. Doing so, the recovery analysis performed after the impact analysis, is able to evaluate each single business degradation and by using the back-references to the causing resource degradations to plan an appropriate recovery of the corresponding resources.

To sum it up, the basic abstract impact analysis subworkflow consists of the two following subworkflow steps:

- service impact analysis (BRAWf.1.1):
  - it uses the initially given list of resource degradations as input.
  - it determines as its output the service impact, i.e., impact on services, specific service functionalities, and on their QoS, for each affected service instance, i.e., customer/user, from the point of view of the customer or user.

- business impact analysis (BRAWf.1.2):
  - it is based on the service impact information used as its input.
  - it determines as its output the business impact, i.e., actual impact on the business of the provider, be it financial or reputational (at least SLA violation costs; in general also aspects like revenue loss - e.g., based on service usage prediction, image loss concerning customers or the whole public).

4.2.2.2 Identification and analysis of additional artifacts

further artifacts necessary for impact analysis

After the discussion of the essential basic input and output artifacts of the impact analysis subworkflow, namely the initial resource degradation list as well as its derived service impact and business impact, in the following artifacts necessary in addition to the former ones are introduced and discussed.

Generally speaking, in addition to its essential input and output artifacts, impact analysis, i.e., particularly service impact analysis and business impact analysis, need some kind of information that determines how to derive entailed service degradation from the resource degradation given initially, and how to derive entailed business degradations from the service degradations. That is, information determining how the different types of degradations depend on each other, is required.

impact dependency models

Therefore, for each, for service impact analysis as well as for business impact analysis, an additional abstract artifact, a so called *dependency model*, is introduced: *service impact dependency model (SI dependency model)* for service impact analysis, and *business impact dependency model (BI dependency model)* for business impact analysis. The SI dependency model determines how to derive service degradations entailed from resource degradations, i.e., it determines the mapping from resource degradations to service degradations (*resource-to-service degradation mapping*). Likewise, the BI dependency model determines how to derive business degradations entailed from service degradations, i.e., it determines the mapping of service degradations to business degradations (*service-to-business degradation mapping*). In Fig. 4.9 the relationship of both dependency models to the derivation of service and business degradations from resource and service degradations entailing them is visualized.

In general, both dependency models comprise all information necessary to derive entailed service or business degradations from resource or service degradations, respectively. That is why they are both also subsumed under the term *impact dependency models*, in order to distinguish them from dependency models used in other areas, e.g., as configuration or change management.

As the specific data structures and details of degradations here are only covered in an abstract way and are actually investigated in the Impact Analysis Framework in Sect. 4.3, the dependency models, which correspond to these degradations, are only covered in abstract way here, too.

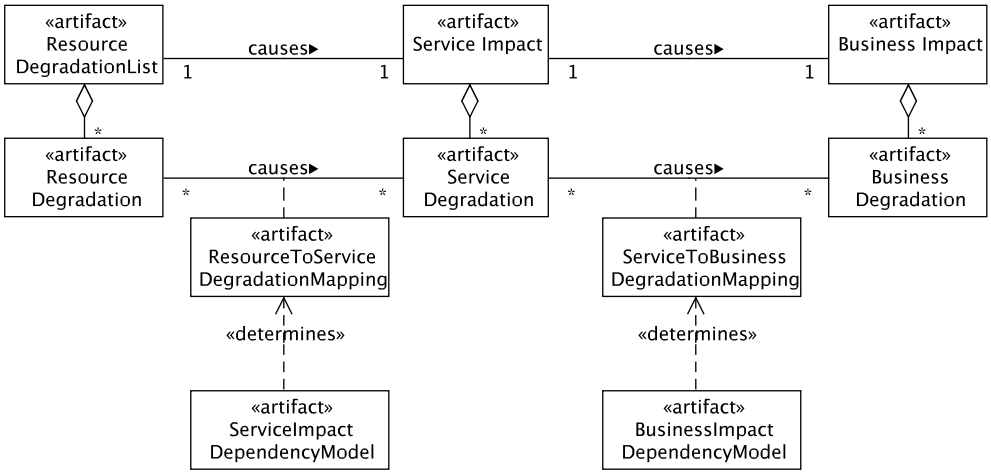

Figure 4.9: Impact analysis dependency models for mapping of resource degradations to service degradations, and service degradations to business degradations

examples of impact dependency model data

Nevertheless, some examples referring particularly to the example situation $ExSit1$ (see p. 130) and referring in general to the example scenario of Sect. 2.3 are here given for motivation:

A very basic type of data necessary for SI dependency models is the information about general dependencies of services on resources used for realizing the services. Simple examples for the mail service s_{mail} are the dependencies $r_{\text{mailout}} \rightarrow f_{\text{mail/use/send}}$, $r_{\text{spamcheck}} \rightarrow f_{\text{mail/use/recv}}$, $r_{\text{mailin\_studlmu}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LMU} \cup \text{stud})$, and $r_{\text{majordomo}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver}\ \text{is mailinglist})$. A slightly more complex example for this type of impact dependency information is the (indirect) dependency of services on resources of subservices: e.g., $r_{\text{iplink}}(r_{\text{rt\_lrz}}, r_{\text{r\_sw\_2}}) \rightarrow f_{\text{mail/use/send}}$ which is a combination of $r_{\text{iplink}}(r_{\text{rt\_lrz}}, r_{\text{r\_sw\_2}}) \rightarrow f_{\text{ip/use/connect}}(\textit{path} = r_{\text{mailout}} \cdots anywhere)$ (compare Fig. 2.9 on p. 49) and $f_{\text{ip/use/connect}} \rightarrow f_{\text{mail/use/send}}$, i.e., the dependency relevant for deriving service degradation g_{s1-1} from g_{r1} in $ExtSit1$.

From such basic, general dependency information at least a derivation of services/service functionalities/service instances probably affected by a given resource degradation is possible. That is, from the degradation subject(s) of a resource degradation, a resource (list), a derivation of the degradation subject, a service (functionality) or service (functionality) instance (set), of a probable service degradation is possible. E.g., from the general dependency $r_{\text{majordomo}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver}\ \text{is mailinglist})$, and any given resource degradation of $r_{\text{majordomo}}$, a service degradation of the service functionality instance set $f_{\text{mail/use/recv}}(\textit{receiver}\ \text{is mailinglist})$ can be assumed. However, nothing can be derived concerning other aspects of such an probable service degradation, e.g., nothing about the degradation manner (which QoS parameter is affected), nothing about the degradation value accuracy (degraded value range of the QoS parameters), nothing about the temporal course/duration of the degradation. Concerning the temporal aspect of the service degradation,

it can of course often be assumed that the temporal course/duration of the service degradation is equal to the one of the resource degradation. But this is not in any case true, as an entailed service degradation might e.g., take longer than the initial resource degradation entailing it, e.g., if a short resource timeout leads to a disruption of a high-level service session which is not resumable without human interaction. So, in order to derive aspects of a service degradation besides the degradation subject, further degradation mapping information has to be available:

As in the example I/R run of Sect. 2.3.4, information is necessary to derive from the resource degradation g_1 (IP link with high link utilization $> 60\%$) that mail sending functionality and mailbox functionality are slowed down, i.e., to derive an increase of the QoS parameters values of $q_{\text{mail/delay\_send\_intra}}$ and $q_{\text{mail/delay\_send\_extra}}$, so that associated QoS constraints are violated (e.g., sla_cnstr_{mail1}, sla_cnstr_{mail2}). This type of information, modeling and describing more exact value range and time dependencies, may be statically modeled (e.g., by mathematical functional specifications or by statistical models) and/or may be derived from actual measurements (performed passively or actively) or actively performed test actions.

Examples of information necessary for BI dependency models, are the definitions of the SLA violation costs, like sla_pnlty_{mail1} and sla_pnlty_{mail2} of example situation $ExtSit1$, making it possible to derive from service degradation information, e.g., outage duration $q_{\text{mail/reliab\_general}} > 30\,\text{min}$ (sla_cnstr_{mail2}), the exact business degradation *mail availability SLA violation costs per time* (as part of g_{b2-1}), specifically described as a function of time: $slp_{\text{mail1}}(.)$ (compare Fig. 4.2.2.1, as well as the definition of $slp_{\text{mail1}}(.)$ on p. 51).

static and dynamic impact dependency model data

Some general statements about both impact dependency models can be still made on the abstract level. They are introduced in the following.

In general, the data of impact dependency models can be divided up into two general parts: rather static, basic data and additional dynamic data.

static impact dependency data as general basis

On the one hand, the dependency model comprises the proper model data, which describes the derivation of service/business degradations from resource/service degradations. This type of data is more static in nature, it is normally explicitly modeled, e.g., according to the SLA between provider and customer, according to the service definition of the provider, or according to the assignment of resources for service realization, between provider and the customer. The access to this type of data is normally fast, takes not much time, and causes no further specific overhead.

dynamic impact dependency data for increased accuracy/granularity

On the other hand, additional, dynamic dependency information for the mapping of degradations is attached to the static data of the dependency model. This type of data is more dynamic in nature, it usually not explicitly modeled, instead the model data contains only references to appropriate data sources, tools, systems, or platforms to gather, collect, or measure this dynamic data on-demand. Furthermore, the access to this type of data takes some additional

time, e.g., for active QoS measurement, active service usage measurement, or active service usage prediction. In general, this dynamic data is necessary to check, to verify, and to enhance the accuracy and granularity of the estimated information based on the static model data.

examples of static and dynamic dependency information

Referring to the examples of impact dependency information given above, statically modeled data are often the dependencies of services on resources (at least the complete possible range of such dependencies, i.e., even if the actual dependency of a service on a resource might change dynamically in time or depending on the particular user/customer at a specific time). So, for $ExSit1$, the dependency $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}}) \rightarrow f_{\mathrm{mail/use/send}}$ is an example for this. A typical refinement of this type of dependency information is the one taking into account not only the degradation subjects (e.g., $f_{\mathrm{mail/use/send}}$), but also the degradation manner (mostly described by the affected quality parameter, e.g., $q_{\mathrm{mail/delay\_send\_intra}}$ for $f_{\mathrm{mail/use/send}}$) and the mapping of degradation manner from entailing degradations to entailed degradations. Even if the mapping of degradation manner (mapping of quality parameters) is mostly not explicitly and statically modeled concerning exact value/time-relationships, nevertheless it often is statically known that a particular degradation manner (e.g., quality parameter, e.g., $q_{\mathrm{mail/delay\_send\_intra}}$) of an entailed degradation is depending on a particular degradation manner (e.g., high link utilization of $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$) of an entailing degradation (e.g., g_{r1} of $ExSit1$). The exact value/time-relationships of degradations are then dynamically determined by current measurements, active test, access to measurement databases. For this purpose, a reference to these measurements or test actions has to be attached to the statically modeled impact dependency model data in order to be able to select and access these dynamic dependency data sources.

A further example of impact dependency information which is most often statically modeled is SLA information (SLA constraints and SLA penalty information, as e.g., $sla\_cnstr_{\mathrm{mail2}}$, $sla\_pnlty_{\mathrm{mail2}}$ in $ExSit1$) for deriving business information from SLA penalties. Such static information can be complemented and extended by dynamically determined dependency information for deriving additional types of business degradations, as e.g., determining revenue loss for a highly dynamically subscribed service by taking into account dynamic measurements of the current service usage as well as dynamic estimation of future service usage.

generic refinement of impact dependency model

Abstracting from these examples, concerning impact dependency information for determining service impact as well as determining business impact, in both cases statically modeled dependency information as well as dynamically determined dependency information (being referenced by the statically modeled information) is involved. Therefore, in the following, for the purpose of describing and specifying impact dependency models, as for SI or BI, two types of data, the *impact dependency static model data* (or *impact dependency proper model data*) and the additional *impact dependency dynamic data* attached to the impact dependency proper model data are distinguished. Fig. 4.10 shows in generic manner the relationship between these types of

data.

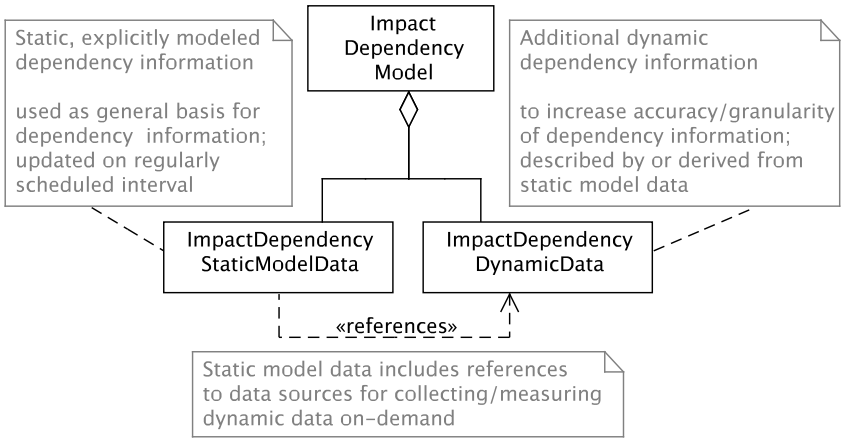

Figure 4.10: Impact dependency model composed of proper, static impact dependency model data and additional impact dependency dynamic data

relationship of static and dynamic dependency data

The static, proper impact dependency model data serves as a basis for the derivation of service/business degradations entailed from resource/service degradations. But in addition to that, the proper model data references the dynamic dependency data, or more strictly speaking, references corresponding data sources or interfaces thereof. By following these references and accessing the data sources for dynamic dependency data, the information about degradations described by or derived from the static model data can be enhanced and refined, with respect to accuracy, granularity/level of detail, validity, and up-to-dateness. For instance, this includes means to specify the triggering of current QoS/QoR measurements, active current service usage measurements, or current service usage prediction. Each of these are examples of data sources known to the static model data for increasing the accuracy of the degradation information. But also data sources for increasing the granularity of degradation information might be considered here, e.g., determining exact information about which service instances of a specific service are really affected by a given degradation.

maintenance of impact dependency models

From the point of view of the actual operation and maintenance of the I/R analysis framework, the static dependency model data is updated on regularly scheduled basis, so that it is available for I/R analysis anytime without much delay and effort. In contrast, additional dynamic dependency data is updated or gathered on-demand, with some access delay (e.g., for actually performing some active measurement), as needed by the currently running I/R analysis.

refinement of SI and BI dependency model

Specifically, for specification and description of the SI dependency model, *SI dependency static model data* and *SI dependency dynamic data* are introduced. Likewise for the BI dependency model, *BI dependency static model data* and *BI dependency dynamic data* are introduced. Fig. 4.11, as refinement of Fig. 4.9, depicts the resulting relationship of proper impact dependency model data together with its additional dynamic impact dependency data for

the derivation of service and business degradations from resource and service degradations.

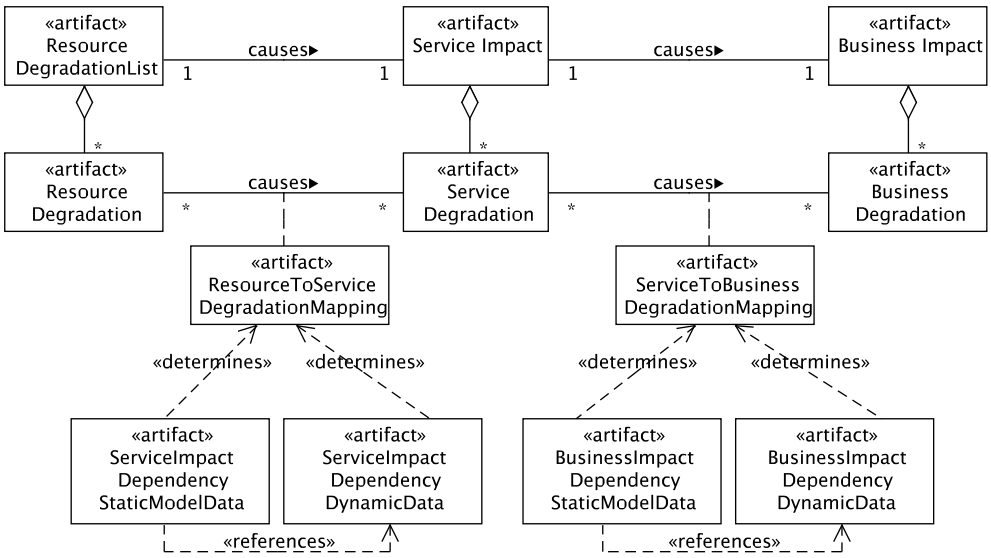

Figure 4.11: Impact dependency models and additional impact dependency dynamic data for mapping of resource degradations to service and business degradations (refinement of Fig. 4.9)

refined issues/tasks of dependency models

Concerning dependency models, on the abstract level, in which these models are treated here - in contrast to their detailed investigation in the Impact Analysis Framework in Sect. 4.3 - still one issue is left to discuss: The introduction of SI dependency model and BI dependency model above, was specifically focused on their general purpose, i.e., the derivation of service degradations from resource degradations in the case of SI, and the derivation of business degradations from service degradations in the case of BI. While this fact remains still generally true, for most applications of I/R analysis the purpose of these impact dependency models has to be extended in a sense, as explicated in the following:

refined mappings of degradations for IA

In reality, in addition to the fact that services or service functionalities depend on resources, there is also the possibility that resources might in the first place depend on other resources. So, degradations of resources can entail degradations of these services directly or indirectly via degradations of other dependent resources. Moreover, in addition to the fact that business degradations, e.g., as SLA violation costs, can be entailed by service degradations directly, a service degradation of some subservice might entail a business degradation indirectly via a entailed service degradations of a main service depending on the subservice.

examples of refined degradation mappings

Some simple examples of such refined dependency information from the example scenario in Sect. 2.3 are the general dependencies from subservices on depending services, as $f_{\mathrm{dns/use}} \rightarrow f_{\mathrm{mail/use}}$, $f_{\mathrm{ip/use/con}} \rightarrow f_{\mathrm{mail/use}}$, or $f_{\mathrm{auth/use}} \rightarrow f_{\mathrm{mail/use/send}}(\textit{authentication} = \mathrm{yes})$.

Furthermore, refined dependency information for the specific example situ-

ation $ExSit1$ (compare p. 130) are given here in order to provide more examples: Fig. 4.12 shows a refinement of Fig. 4.6, introducing additionally service degradations of the involved subservices of s_{mail} and s_{web}. Consequently, there are the newly introduced service degradations g_{s1-0} (ip path delay of IP paths from/to r_{mailout}, entailed by g_{r1}, entailing partly g_{s1-1}), g_{s1b-0} (DNS service request delay, entailed by g_{r1b}, entailing g_{s1-2} as well as g_{s1-1} together with g_{s1-0}), and g_{s2-0} (AFS service unavailability for certain AFS paths, entailed by g_{r2}, entailing g_{s2-1} and g_{s2-2}). Moreover, the figure shows the corresponding refined mapping between all now involved degradations. An example is the split-up of the (indirect) dependency $r_{\text{iplink}}(r_{\text{rt\_lrz}}, r_{\text{r\_sw\_2}}) \rightarrow f_{\text{mail/use/send}}$ (service functionality dependent on a resource of a subservice), into the direct resource-to-service dependencies $r_{\text{iplink}}(r_{\text{rt\_lrz}}, r_{\text{r\_sw\_2}}) \rightarrow f_{\text{ip/use/connect}}(\textit{path} = r_{\text{mailout}} \cdots anywhere)$ and $f_{\text{ip/use/connect}} \rightarrow f_{\text{mail/use/send}}$ - already mentioned earlier.

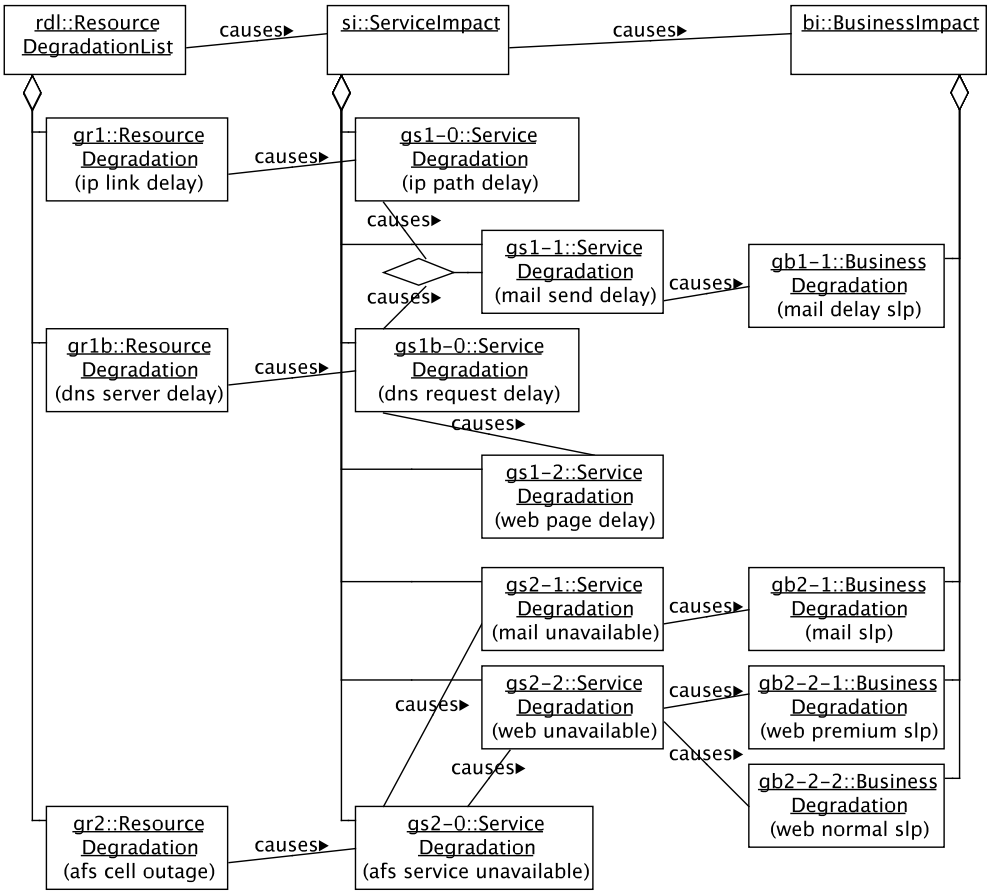

Figure 4.12: Refined example situation $ExSit1$ (refinement of Fig. 4.6)

refined task of SIA

Therefore, as the purpose of SIA is to derive all service degradations (as experienced by users/customers) caused by an initially given list of resource degradations, SIA is not only concerned with the direct derivation of service degradations from the initially given resource degradations. Instead also the derivation of possible further resource degradations, which in turn entail service degradations, as well as degradations of subservices which entail degra-

dations of dependent services have to be taken into account by SIA. Concluding, specifically the SI dependency model, which per definition above covers all necessary information needed for the derivation of service degradations (by static or dynamic data, compare above) for SIA, has to cover all these types of mappings between degradations. To sum it up, SI dependency model has to cover these three types of degradation mappings:

refined dependencies for SIA

- *resource-to-resource degradation mapping*: derivation of resource degradations entailed directly by one or multiple other resource degradations already derived before, starting from the list of resource degradations initially given.
- *resource-to-service degradation mapping*: derivation of service degradations entailed directly by one or multiple resource degradation already derived before.
- *service-to-service degradation mapping*: derivation of service degradations entailed directly by one or multiple service degradations already derived before.

Furthermore, it might be necessary in some cases to combine some of the three cases in one single degradation derivation. An example is the derivation of a degradation of a main service (e.g., high mail sending delay) being entailed directly by a combination of a resource degradation (slow subservice client, e.g., DNS resolver for mail sending) and a service degradation of a subservice (high DNS response delay). These issues are in detail treated in the Impact Analysis Framework in Sect. 4.3.

refined task of BIA

A similar extension as for SIA has to be made for BIA: the BI dependency model has not only to cover the derivation of business degradations entailed directly by service degradation derived before, but instead has also to cover the derivation of complex business degradations being entailed by other business degradation already derived before. A simple example are overall SLA violation costs for a specific service composed of the SLA violation costs for specific customers of this service. That is, the BI dependency model should also allow for the aggregation of individual business degradations, where this is appropriate or necessary depending on the specific definition of business impact of the specific provider.

refined dependencies of BIA

Concluding, BI dependency model has to cover these two types of degradation mappings:

- *service-to-business degradation mapping*: derivation of business degradations entailed directly by one or multiple service degradations already derived by SIA.
- *business-to-business degradation mapping*: derivation of business degradations entailed directly by one or multiple other business degradation already derived before.

Fig. 4.13 is an extension of Fig. 4.11, depicting the detailed relationship of the impact dependency models and the mapping between the various kinds of degradations for I/R analysis.

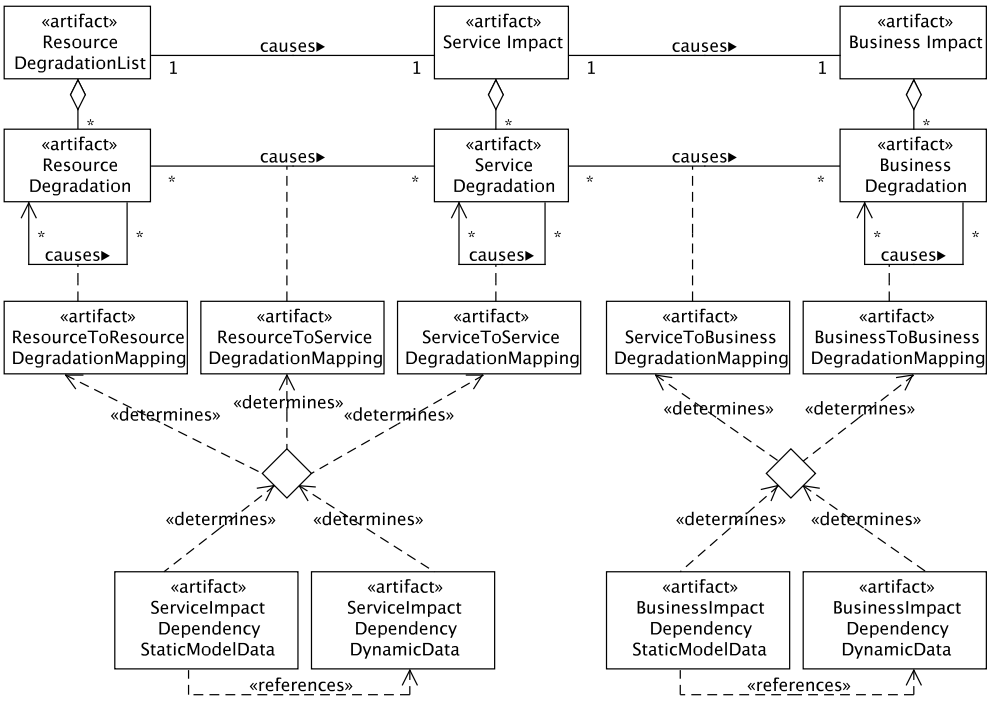

Figure 4.13: Impact dependency models and additional impact dependency dynamic data for mapping of resource degradations to service and business degradations in refined manner (Extension of Fig. 4.11)

refinement of workflow with all necessary artifacts

After having completely analyzed all artifacts necessary for IA in abstract form, including artifacts needed in addition to the essential input/output artifacts, the first version of basic refined abstract impact analysis subworkflow BRAWf.1 (Fig. 4.8), is now further extended with corresponding artifacts. Fig. 4.14 shows this complete version of BRAWf.1.

This refined version includes all newly identified artifacts and so also artifacts needed in addition to the essential input/output artifacts, i.e., in addition to the initial resource degradation list, the service impact, and the business impact. Namely, these additional artifacts are the SI dependency model and BI dependency model used for derivation of entailed degradations, i.e., the determination of degradation mappings. The SI dependency model is used in the service impact analysis step (BRAWf.1.1), the BI dependency model is used in the business impact analysis step (BRAWf.1.2). Each of them comprises proper model data as well as attached, additional dynamic data.

In fact, this highly refined version of the impact analysis subworkflow as part of the abstract workflow also represents the impact analysis part of the basic refined abstract workflow (BRAWf), which is used as a whole in Sect. 4.2.5 to devise the basic component architecture (BCArch) of the basic framework.

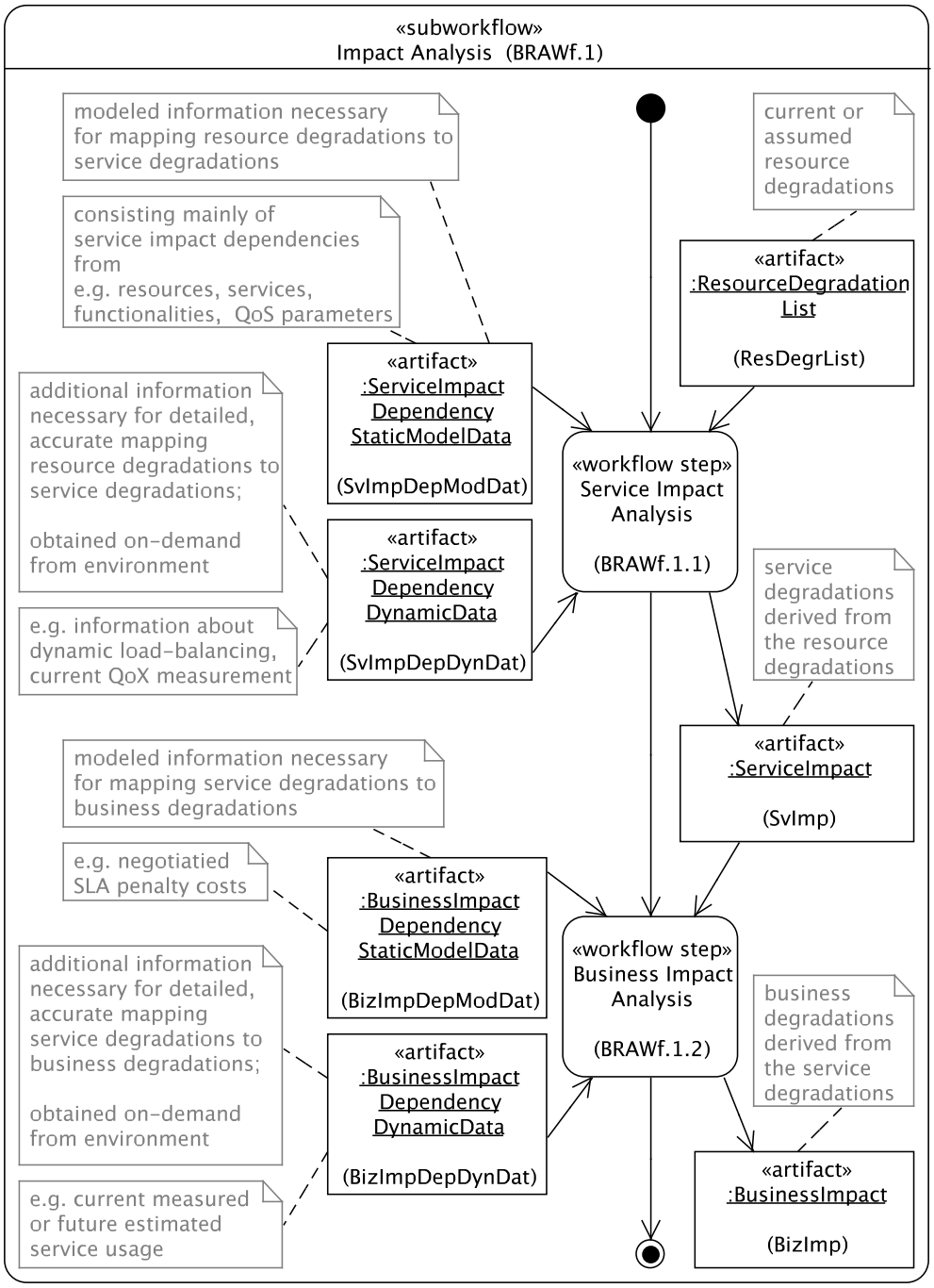

Figure 4.14: Basic refined abstract subworkflow of impact analysis with all artifacts necessary (BRAWf.1, refinement of Fig. 4.8)

4.2.3 Basic abstract subworkflow for recovery analysis

Here, the recovery analysis subworkflow of the basic abstract workflow, BAWf.2 (second step in Fig. 4.3) is analyzed and discussed on an abstract level and refined accordingly to BRAWf.2.

4.2.3.1 Analysis of essential artifacts

basic situation of recovery analysis

Fig. 4.15 illustrates the basic situation for recovery analysis, which is described in the following.

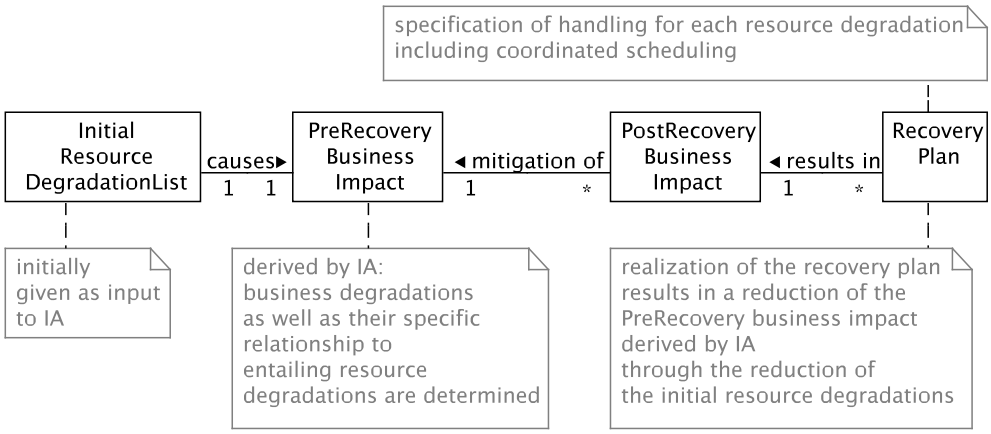

Figure 4.15: Basic situation of recovery analysis

Recovery analysis (RA) follows impact analysis (IA) based on the IA output artifact, i.e., the computed business impact, which has been derived by IA in relationship to the list of originally entailing resource degradations. Of course, these resource degradations entail the business degradations indirectly via directly entailed service degradations (compare definitions on p. 133 in Sect. 4.2.2.1). However, for the ease of discussion and illustration in the following, this relationship of resource degradations entailing business degradations via service degradations is often not explicitly stated, and instead is only abstractly referred to as the relationship of resource degradations entailing (indirectly) business degradations. Similarly, the fact of the initial resource degradation list entailing (indirectly) the business impact has to be understood.

purpose of a recovery plan

The task of RA is to devise and propose a recovery plan, whose execution should result in an optimal minimization or mitigation of the business impact derived by IA. Consequently, there are two types of business impact being considered as far as RA is concerned (compare artifacts of BAWf.2 in Fig. 4.3 on p. 127): The business impact derived by IA (essential input of RA), as well as the *reduced business impact* resulting from the realization of the proposed recovery plan (essential output of RA).

The business impact derived by IA does not take into account any recovery measures or recovery plan yet. It describes the estimated development of the financial/reputational business impact over time threatening unless the provider intervenes in some way against the initial resource degradations. In contrast, the reduced impact is the estimated result of the realization of a recovery plan intervening against these initial resource degradations. The realization of the recovery plan either completely eliminates, or at least reduces the business impact originally threatening, through the respective complete elimination or partial reduction/mitigation (possibly considering trade-offs) of the original resource degradations. Therefore, for the purpose of RA, the business impact (as derived by IA) is more exactly denoted as *pre-recovery business impact*, and correspondingly the estimated reduced impact (derived by RA) is denoted as *post-recovery business impact*.

follow-up of IA example situation

Now, following up the example I/RA run of Sect. 2.3.4 and its extension, the specific example situation $ExSit1$ of Sect. 4.2.2 introduced on p. 130, an example of a possible recovery plan, $ExRecPlan1$, for the example situation $ExSit1$ is outlined.

Generally speaking, the recovery plan, as a description how to actually perform the recovery, includes two basic pieces of information: descriptions for the handling of the various resource degradations (DgrHndl), as well as some sort of order or scheduling information for these handlings (DgrHndlSched).

In order to remind - in the example situation $ExSit1$ three initial resource degradations are involved: g_{r1} (high IP link utilization), g_{r2} (AFS cell outage), and g_{r1b} (high DNS server response delay), each along with its indirectly entailed business degradations (compare p. 130). The recovery plan $ExRecPlan1$ for $ExSit1$ is devised with the following order of handling ($DgrHndlSched1$) for the involved resource degradations: The recovery described by $ExRecPlan1$ is first concerned with the handling of g_{r2} paying most effort ($DgrHndl(g_{r2})$), second with the handling of g_{r1b} paying less effort ($DgrHndl(g_{r1b})$), and last with the handling of g_{r1} also paying less effort ($DgrHndl(g_{r1})$). It is assumed and estimated by RA in this example that by using this order of handling for the resource degradations the resulting post-recovery business impact will be minimized. The outline of this example recovery plan will be continued and refined in the remainder of this section, serving as a continuous example.

business impact to specify impact situation

Pre-recovery business impact, as well as post-recovery business impact along with its associated recovery plan, all being artifacts of RA analysis, describe or relate to a specific *impact situation*, a term introduced in the following. Fig. 4.16 (compare to Fig. 4.15) illustrates these relationships.

On the one hand, pre-recovery business impact, which is determined by IA, describes the *actual impact situation*, existing currently or evolving in future if no recovery takes place (“what is the actual situation?”). On the other hand, post-recovery business impact, which is determined by RA, specifies the *targeted impact situation* to be reached, as being reachable from the actual impact situation and by acceptable effort (“which targeted situation should be

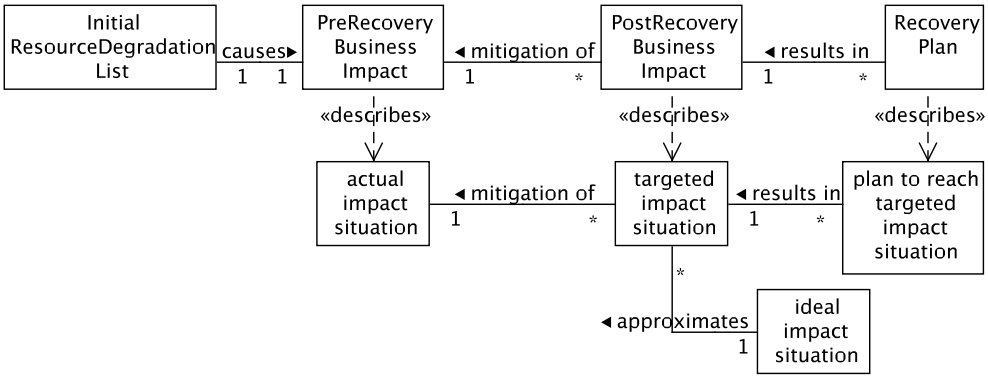

Figure 4.16: Essential artifacts of RA in relationship to actual/targeted impact situations

reached?"). Furthermore, a recovery plan, also determined by RA, specifies the recovery necessary to reach the targeted impact situation ("what should be done in order to reach the targeted situation?").

In general, the targeted situation should be equal or at least be approximating to the *ideal impact situation*, which is of course the current/future situation of having no actual business impact left over time.

Nevertheless, the targeted situation, described by the reduced business impact as determined by RA, is only an estimation for the potential result that the execution of the corresponding recovery plan will result in.

necessity of multiple recovery alternatives

Furthermore, the proposal of a single recovery plan (with attached reduced impact) by RA might be too restricted: Often there might be multiple possible solutions available for a recovery, which all at least seem to result in the same impact mitigation. In addition to that, different possibilities for a recovery plan are focused on different priorizations of the threatening degradations. Moreover, for the choice of an appropriate recovery plan various trade-offs might have to be considered.

Concerning the possibility of multiple recovery alternatives, both may vary: the actual performed handlings of resource degradations, as well as their scheduling.

example for multiple recovery alternatives

In case of the example recovery plan $ExRecPlan1$ for situation $ExSit1$, the order/scheduling for the handling of the given resource degradations is already fixed ($ExDgrHndlSched1$, see above). There are still different options how to handle one of the particular three resource degradations, each of these options considering different trade-offs.

For the handling of the severest resource degradation g_{r2}, i.e., the handling of the AFS cell outage ($DgrHndl(g_{r2})$), there are the following options: One option is to try to fix the broken device directly, e.g., to reboot it and try to continue its service, which can be done very fast in $3\,\mathrm{min}$ (option $DgrHndl(g_{r2}, 1)$). But if the original problem is still located within this device, it might break down again. So, this option might result in subsequent reboots of the AFS device, from time to time. Such behavior will be at least

annoying for users and customers, but will lastly also result in an aggravated availability degradation and so only partially reduce the business degradations originally caused by g_{r2}. A second option is to use a backup device, which can be quickly made ready, but which is working with reduced performance (option $DgrHndl(g_{r2}, 2)$). This reduced performance will result in an aggravation of the already existing service degradation g_{s1-2}, i.e., the web page access delay. This web page delay will be aggravated at least for the particular web service customers whose web pages are located in the broken AFS cell. A further option is to try to check whether some software updates were performed recently, which might be the cause of the problem, and to try to roll-back these updates (option $DgrHndl(g_{r2}, 3)$. But this option will probably take more time than the other two ones. Moreover, an option is the restriction or capacity limiting of other AFS users (option $DgrHndl(g_{r2}, 4)$): It could be checked if the cell outage is somehow caused by another AFS user (group), i.e., other user(s) directly accessing AFS and not indirectly by the mail or web hosting service. Such an external user (group) may have overloaded the AFS service and may have caused the problem. In that case the access of this user (group) could be completely restricted or at least its allowed capacity for AFS file transfers could be limited. Finally, a last option is to buy a new device to replace the old one (option $DgrHndl(g_{r2}, 5)$). However this can hardly be a short-term solution, as it may take days or weeks until it can be realized. At last, further options can be derived from the above mentioned ones by combining some or all of them: E.g., the combination of option $DgrHndl(g_{r2}, 1)$ and $DgrHndl(g_{r2}, 2)$, i.e., trying option $DgrHndl(g_{r2}, 1)$, and if does not work at all or if it leads to too much restarts of the AFS device falling back to option $DgrHndl(g_{r2}, 2)$ (option $DgrHndl(g_{r2}, 1+2)$). Similarly, the combination of all the above options (option $DgrHndl(g_{r2}, 1+\ldots+5)$) is possible.

Similarly, there are handling options for the resource degradation g_{r1}, that is the handling of the high IP link utilization ($DgrHndl(g_{r1})$). One possibility is the restriction or at least the limiting/under-prioritization of other IP traffic of this link not originating from the affected mail service (see entailed service degradation g_{s1-1}) (option $DgrHndl(g_{r1}, 1)$). Alternatively, it could be tried to reroute such other IP traffic to a different IP path, if somehow possible (option $DgrHndl(g_{r1}, 2)$). Moreover, an additional IP link for redundancy purposes could be added as support of the existing one (option $DgrHndl(g_{r1}, 3)$). In this case, the service provider LRZ would have only to utilize an additional wavelength on the physical dark fiber, over which it has complete control, and on top of which it realizes the affected, single IP link currently. So, this option would take ca. $1\ \mathrm{h}$ for preparation and initialization, which is remarkably quick for installing a new IP link. Again, for each of these option different trade-offs have to be considered, and combination of some/all options are possible as further options.

In order to allow for the consideration of different trade-offs, RA in general does not only propose a single recovery plan for describing a single recovery solution, but instead proposes a list of multiple recovery plans describing different recovery alternatives. To each of these alternative recovery plans is

multiple recovery plans describing different recovery alternatives

attached its corresponding, estimated reduced business impact. Fig. 4.17 (refinement of Fig. 4.15) illustrates this refined situation for recovery analysis considering multiple recovery alternatives.

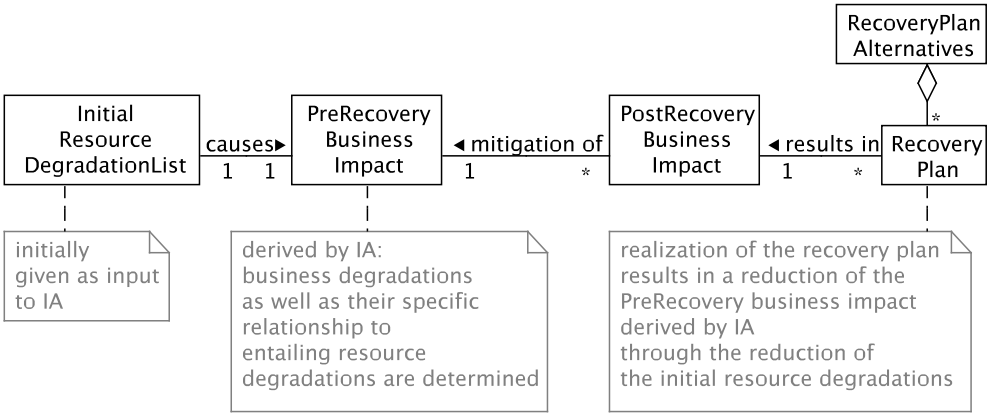

Figure 4.17: Refined situation of RA with multiple recovery alternatives (refinement of Fig. 4.15)

The different recovery alternatives described by different recovery plans should all result in a reduced business impact of a similar order of magnitude. Nevertheless, each resulting reduced business impact for an alternative may be consisting of different reduced business degradations: Each alternative might consider different trade-offs, or might differ in the accuracy and/or granularity of the estimation of the post-recovery impact. E.g., one alternative might be very risky, in that its reduced impact is not completely assured, but the mitigation of this reduced impact can be very high in case of success. In contrast, another alternative might be of more conservative character, in that its estimated result is relatively assured if the recovery alternative is used, even if its mitigation is not so high as in the aforementioned case.

The remainder of this section is structured as follows. First, the structure of post-recovery impact (reduced impact for a specific recovery plan) as well as its detailed relationships to pre-recovery impact is discussed. Second, the structure of a recovery plan as well as its detailed relationships to post and pre-recovery impact are outlined. Afterwards it is analyzed how RA actually derives recovery plan and reduced impact from the pre-recovery impact. Consequently, the basic abstract RA subworkflow is refined accordingly.

detailed structure of post-recovery impact

Fig. 4.18 gives an overview of the detailed structure of post-recovery impact and its detailed relationships to pre-recovery impact, which are treated in the following.

pre/post-recovery business degradations

Pre and post business impact specifically consist of one or multiple business degradations. Each pre-recovery business degradation is entailed by pre-recovery resource degradations, of which an initial subset is given to IA (further ones may be derived by IA, compare Sect. 4.2.2, especially p. 146). The post business impact also comprises business degradations, which are entailed by post-recovery degradations - in addition to costs which are treated

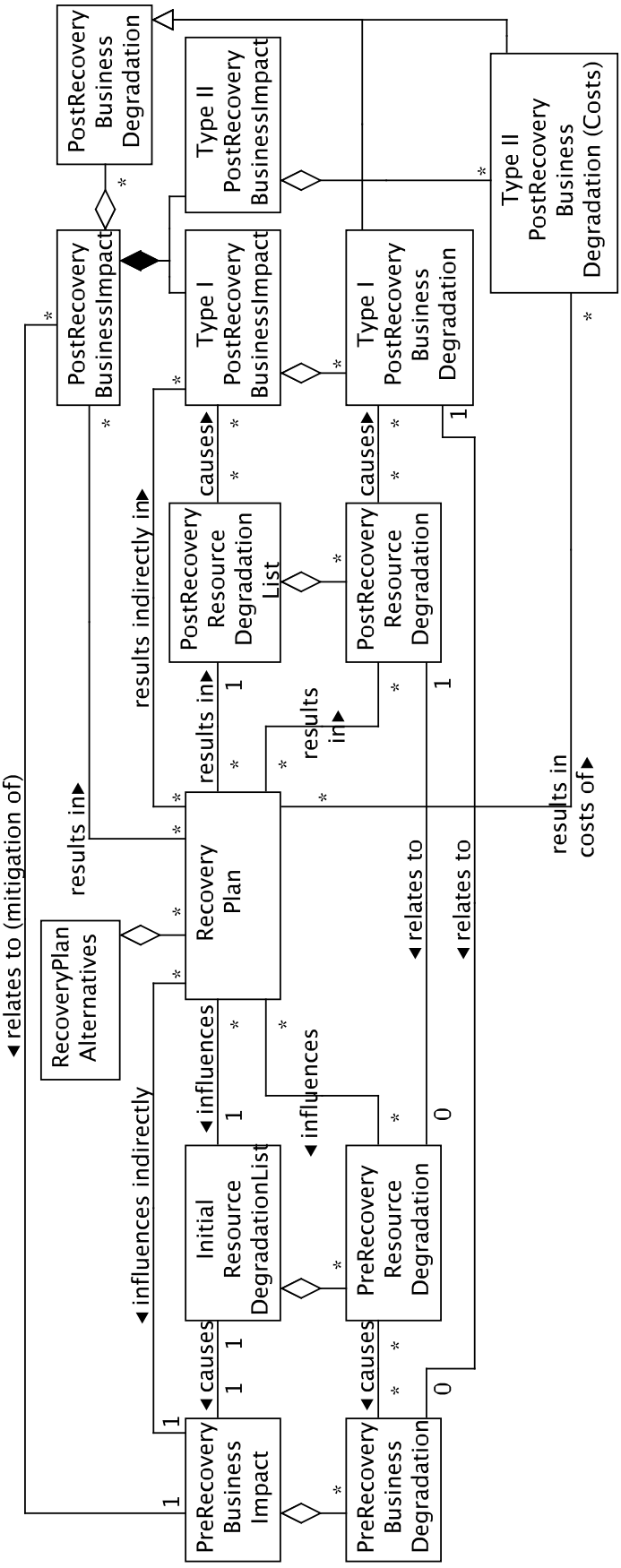

Figure 4.18: Refined situation of RA taking into account the detailed structure of post-recovery impact (refinement of Fig. 4.17)

below. Consequently, these respective business degradations are also called *pre-recovery business degradations* and *post-recovery business degradations* accordingly in the following. Furthermore, correspondingly resource degradations given or derived by IA are called *pre-recovery resource degradations*, whereas the (estimated) resource degradations resulting from the realization of a proposed recovery plan and leading as a whole to reduced business impact, are called *post-recovery resource degradations*.

related pre/post-recovery degradations

A pre-recovery business degradation has, as a counterpart in the post-recovery impact, a particular post-recovery business degradation, which describes the resulting change of state of the business degradation after performing the recovery alternative. This counterpart is called the *related post-recovery business degradation*. This change of state of the business degradation is actually attained by the recovery alternative through the change of state of entailing resource degradations. That is, pre-recovery resource degradations also have, as a counterpart, a particular post-recovery resource degradation, the *related post-recovery resource degradation*, which describes this resulting change of state of the resource degradation by performing the recovery alternative. In general as a change of state a positive one, i.e., the elimination or at least reduction of the specific degradation, is desired. But, when considering trade-offs also a negative change of state, for some resource degradations and therefore possibly also for some business degradations is possible.

example for related degradations

For the example recovery plan $ExRecPlan1$ and its resource degradation handling option $DgrHndl(g_{r2}, 2)$ (compare p. 152) the pre-recovery degradation g_{r2} is related to its related post-recovery degradation $g_{r2,post2}$: Option $DgrHndl(g_{r2}, 2)$ means a partially reduced resource degradation g_{r2} and reduced entailed business degradations thereof. After nearly a duration of $35\ \mathrm{min}$ for preparing and initializing the backup device the AFS outage itself will have been handled. So, the related post-recovery degradation $g_{r2,post2}$ and its entailed business degradations are now limited in duration. Nevertheless the remaining duration of $35\ \mathrm{min}$ are still causing (even if reduced) business degradations, i.e., some SLA violation costs. Moreover, this option actually aggravates the service degradation g_{s1-2} initially only entailed by the resource degradations g_{r1b} (compare Fig. 4.6 on p. 131): Because the backup AFS device is only working with reduced performance, it is causing a further rise of the web page access delay, at least for the specific web hosting users those storage is located on the affected AFS cell. This eventually leads to an additional, new business degradation.

That is why as pre/post business impact is consisting of one or multiple pre/-post business degradations entailed by pre/post resource degradations (strictly speaking, via service degradations), a recovery alternative might have to handle - either fully or at least to some extent, depending on assigned priorities - multiple of the initial (pre-recovery) resource degradations.

The final result of a recovery alternative on a particular resource degradations as well as entailed business degradations may vary. E.g., particular post-recovery degradations may be reduced or may be aggravated in comparison

with their pre-recovery counter part. But of course as overall result the whole post-recovery business impact as the sum of all post business degradations should be minimized.

Before continuing further with a detailed identification of types of post business degradations with respect to the positive or negative effect a recovery alternative may exert on them, a generic terminology for such effects is introduced: On the one hand, a particular recovery alternative *influences* pre recovery degradations in a positive or negative manner. And specifically, recovery alternatives are chosen exactly because of such influence in a positive manner. On the other hand, when this (negative or positive) influence is really going to happen, various post-recovery degradations *result*. In order to make a clear distinction, in the following the term *recovery (alternative) influence* shall be reserved for the relationship of recovery alternatives and pre-recovery degradations/impact, whereas the term *recovery (alternative) result* shall be concerned with (estimated) resulting post business degradations/impact. Of course, influence and result of recovery alternatives are often very related - similar as pre and post-recovery impact - only the point of view (pre or post recovery) decides which of the both terms is used. For instance, if a recovery alternative reduces some pre-recovery degradation as its influence, the recovery alternative also results in a related reduced post recovery degradation. Moreover, in order to allow for a complete comparison between pre and post recovery impact, for each influenced pre-recovery degradation there is always some resulting post-recovery degradation, be it eliminated, reduced or aggravated.

influence and result of a recovery alternative

In the I/RA framework developed in this thesis only recovery alternatives shall be considered which influence pre-recovery business degradations indirectly through the handling of one or multiple entailing pre-recovery resource degradations (via corresponding handling of entailing service degradations). Further extensions of this I/RA framework could also consider more high-level recovery handling of business degradations: E.g., future changes to the service offering (recovery action handling the service degradation directly), future changes to the pricing of the services (recovery action handling the business degradation directly), negotiation with the customer for not paying the full amount of SLA penalties in exchange for better service conditions for this customer (also recovery action handling business degradations directly). Most of these high-level recovery options target only long-term future business degradations, while current and short-term future business degradations are covered hardly. Moreover, such high-level recovery alternatives and their resulting influence on business degradations might be hard to model and to formalize. That is why in this I/RA framework, only recovery alternatives directly targeting at the originally entailing resource degradations, and not directly at the entailed service degradations or even the (indirectly) entailed business degradations, are considered.

only handling of resource degradations

Concerning the type of resource degradations, pre-recovery impact situation and post-recovery impact situation do not differ. Simply, both include some

comparison of pre/post business impact

resource degradations, of course the post-recovery impact situation in general of weaker ones. But this similarity does not completely hold for business degradation of pre recovery impact situation and post-recovery impact situation. Even if post-recovery business impact is to be targeted as a mitigation/reduction of pre-recovery business impact, its structure is in a sense more complex: Pre-recovery business impact only consists of business degradations which are always indirectly entailed (via service degradations) by initial resource degradations. Of course, particular post business impact can also comprise such type of business degradations entailed directly from resource degradations: For instance this is the case if an originally existing entailing pre-recovery resource degradation could not be completely eliminated by the recovery alternative, or in contrast has even been aggravated.

recovery costs as further business degradation

But in addition to resulting post business degradations entailed by post resource degradations, each recovery alternative often results also in a completely new type of post business degradation: the costs/effort necessary for realizing the recovery alternative. In general, these *recovery costs* also have to be taken into account, and have to be combined/compared with the post business degradations by post resource degradations, in order to provide a realistic perspective for appropriate recovery decisions.

For example, in the case of $ExRecPlan1$, at least the acquiring of a new AFS device, i.e., handling option $DgrHndl(g_{r2}, 5)$ may result in various specific costs. But also for the other handling options costs may be assigned, e.g., additional extra payment of employed staff for working on weekend or in the late evening if it is necessary for realizing the recovery.

Type I and Type II business degradations

Concluding, these two generic types of business degradations have to be distinguished concerning post-recovery impact:

- *business degradations (indirectly) entailed by resource degradations (via service degradations)*, also called *Type I business degradations*.
- *recovery costs*, also called *Type II business degradations*.

Type I business degradations are exactly the type of degradations of which any pre-recovery business impact consists of. In contrast, post-recovery business impact consists of both, Type I and Type II business degradations.

The sum of all Type I recovery business degradations of a post-recovery impact is called *Type I post-recovery business impact*, and the sum of all Type II business degradations (sum of all recovery costs) is called *Type II post-recovery business impact*.

Type I post degradations and impact dependency models

Type I post-recovery business degradations are (indirectly) entailed by post-recovery resource degradations, the same way as pre-recovery business degradations are (indirectly) entailed by pre-recovery resource degradations. That is why the mapping between Type I post-recovery business degradations and entailing post-recovery resource degradations can be described by the impact dependency models, which have originally been introduced for IA (see p. 140 in Sect. 4.2.2.2).

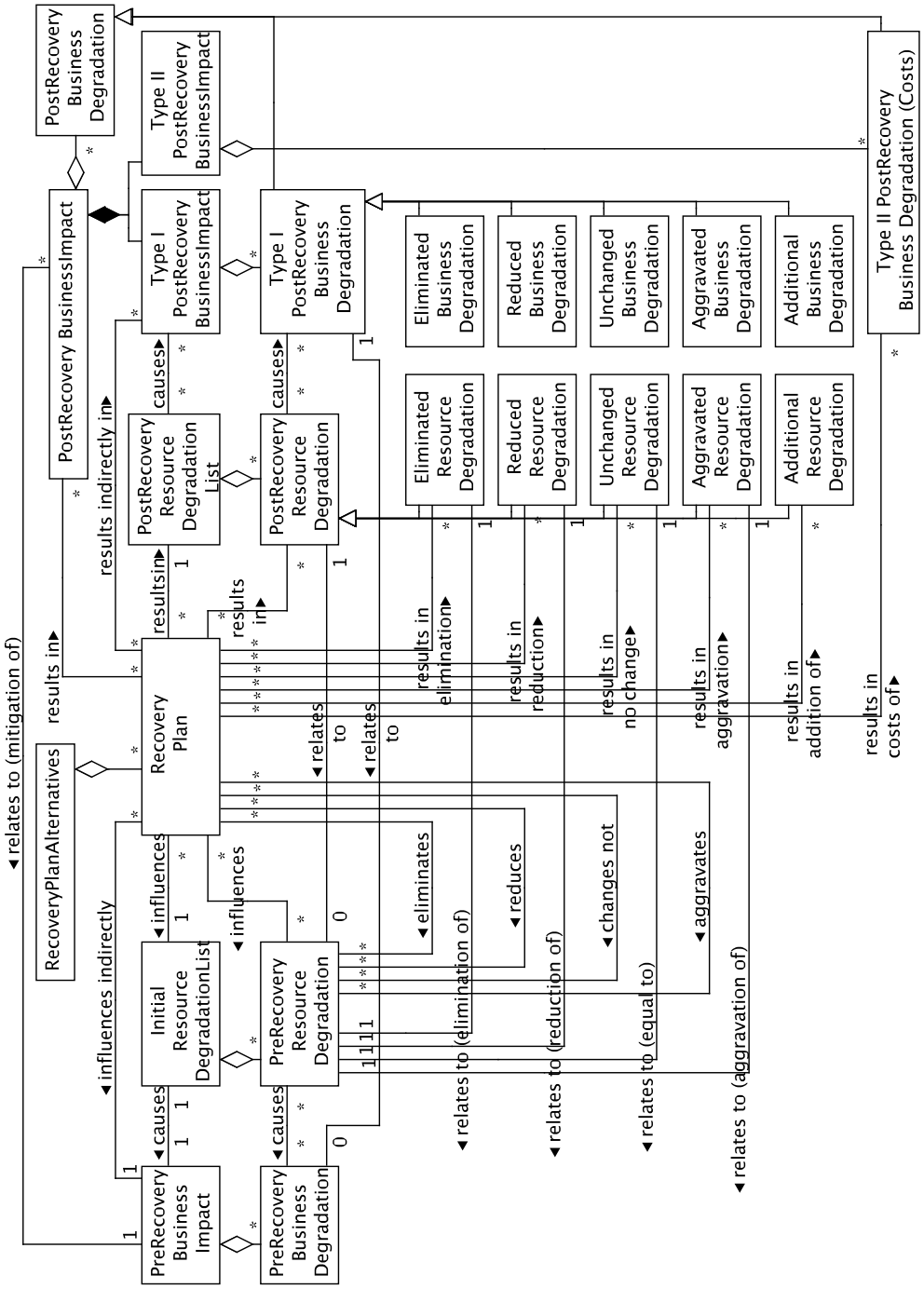

Figure 4.19: RA situation taking into account detailed classification of post recovery degradations (refinement of Fig. 4.18)

subtypes of Type I post-recovery business degradations

Whereas Type II business degradations, i.e., costs, always can be seen a negative result of recovery alternative, this is different for Type I business degradations resulting from a recovery alternative. E.g., they may represent a reduction (positive result) or an aggravation (negative result) in comparison with a pre-recovery business degradation, i.e., one which already existed as part of pre-recovery business impact. Moreover, a very positive result is of course the complete elimination of a pre-recovery business degradation. In contrast, also a negative result is the introduction of a completely new Type I post-recovery business degradation, which was not present as part of the pre recovery impact at all.

Therefore, in the following the post-recovery business degradations of the post recovery impact are classified by comparison with their counter-parts in the pre recovery impact: classifying them as an elimination, a partial reduction, an aggravation, or a new addition of a degradation. Fig. 4.19 gives a refinement of the RA situation by taking into account the different types of Type I post-recovery business degradations.

positive influence of recovery

So, as discussed above, in general two types of positive influence/result by recovery alternatives on the business impact in comparison to pre business impact are possible:

- *BI Elimination*: complete elimination of a pre-recovery business degradations: no business impact over time left.
- *BI Reduction*: reduction of a pre-recovery business degradation: some impact left, at least in a bounded time interval, or impact more long-term in future but with reduced degree.

These types of Type I post-recovery business degradations are both attained by an appropriate elimination or partial reduction of entailing pre-recovery resource degradations. Also they both represent the desired result of a recovery alternative, as they in fact minimize/mitigate business impact.

negative influence of recovery

When trade-offs are considered, negative influence is possible, too:

- *BI Aggravation*: aggravation of a pre-recovery business degradation.
- *BI Addition*: addition or introduction of a new post-recovery business degradations not having a counterpart in the pre-recovery business impact.

These types of Type I post-recovery business degradations are both caused lastly by the aggravation or addition of some of entailing post-recovery resource degradations which outweigh all reduction or elimination of other pre-recovery resource degradations also involved in entailing this business degradation. The addition of an business degradation is a case where no counterpart for the resulting post business degradation already existed in pre-recovery impact. So, there is no specific influence on any pre business degradation as such, but it could be said that the pre business impact as a whole is influenced.

examples of post-recovery degradations

In the example case of the recovery handling option $DgrHndl(g_{r2,2)}$ the pre recovery degradation g_2 itself is reduced, i.e., limited to $35\,\mathrm{min}$ for preparing and initializing the AFS backup device, and correspondingly yields a reduced business degradation (limited SLA violation costs for violating SLA constraints $sla\_cnstr_{\mathrm{mail1}}$, $sla\_cnstr_{\mathrm{mail2}}$, $sla\_norm\_contr_{web1}$, $sla\_norm\_cnstr_{\mathrm{web2}}$, $sla\_prem\_cnstr_{\mathrm{web1}}$ and $sla\_prem\_cnstr_{\mathrm{web2}}$). Nevertheless, this handling option aggravates the service degradation g_{s1-2}, i.e., it leads to a rise of the web page access delay (for web pages stored on the specific AFS cell). This will eventually results in an additionally introduced business degradation (SLA violation costs for g_{s1-2}), if the resource degradation originally entailing g_{s1-2}, i.e., g_{r1b} (high DNS server response delay), is not handled in time.

unchanged business degradations

In addition to the positive and negative Type I business degradations, there might be the case when a recovery alternative does not change a pre-recovery business degradation at all, i.e., an unchanged pre-recovery business degradation.

post-recovery business degradations summary

To sum it up, the types of post business degradations (as illustrated in Fig. 4.19) are:

- Type I business degradations: entailed by resource degradations.
  - *BI Elimination*: complete elimination of a pre-recovery business degradation.
  - *BI Reduction*: partial reduction of a pre-recovery business degradation.
  - *BI NoChange*: no change in comparison to pre-recovery impact.
  - *BI Aggravation*: aggravation of a pre-recovery business degradation.
  - *BI Addition*: additional, newly introduced post-recovery business degradation.
- Type II business degradations, i.e., *recovery costs*

Clearly, for each recovery alternative, the sum of all such negative influences are always required to be less than the sum of all positive influences.

mapping of post-recovery resource to business degradations

Per definition Type I post-recovery business degradations are entailed indirectly by post-recovery resource degradations. Similar as the entailed Type I business degradations, these post-recovery business degradations can be classified as elimination, reduction, no change, aggravation, addition of a resource degradation in comparison of pre/post impact situation. But the specific relationship of particular a type (such as elimination, reduction) of Type I post-recovery business degradations and a particular type of post-recovery resource degradation is not completely trivial. The reasons for this are that the mapping of resource degradations to entailed business degradations in general is a m:n association, and that each particular entailing resource degradation can have been influenced in a negative or positive way. In fact, this mapping can be described by the impact dependency models, originally introduced for IA (see

p. 140 in Sect. 4.2.2.2). For instance, a post-recovery aggravated resource degradation might not entail a post recovery aggravated business degradation, as other entailing pre-recovery resource degradations might have been reduced or eliminated, overall resulting in a reduced or eliminated entailed business degradation. Furthermore, an aggravation of a resource degradation even might be of too small degree in order to also aggravate the entailed business degradations.

summary of the structure of reduced impact

Concluding, the post-recovery impact of a particular recovery alternative can be summarized in the following manner: On the resource level the overall result of the recovery alternative consists of various post-recovery resource degradations, which may be classified as completely eliminated, reduced, unchanged, aggravated, or newly introduced in comparison to pre-recovery impact. On the business level the overall result of the recovery alternative consists of Type II post-recovery business degradation (recovery cost), as well as Type I post-recovery business degradations, which may also be classified as completely eliminated, reduced, unchanged, aggravated, or newly introduced in comparison to pre-recovery impact. The particular mapping from post-recovery resource degradations to Type I post recovery business degradations, including the specific classification of the Type I post-recovery business degradations, can be determined and described by the impact dependency models originally introduced for IA.

basic structure of recovery plans

Previously the detailed structure of the (reduced) post-recovery business impact and its detailed relationships to pre-recovery business impact was introduced. In the following, the detailed structure of a recovery plan, i.e., the a description of a recovery alternative for actually reaching such a reduced business impact, and its detailed relationships to reduced business impact is treated. Fig. 4.20 illustrates a refinement of the RA situation in Fig. 4.17 revealing the generic inner structure of recovery plans.

As in fact pre and post business impact specifically consist of one or multiple business degradations, a recovery alternative in general also has to handle multiple pre-recovery business degradations. Moreover, as explained above, here only recovery alternatives are considered which handle pre-recovery business degradations by handling the corresponding, entailing pre-recovery resource degradations. Therefore, each possible recovery alternative in general has to handle multiple of these pre-recovery resource degradations.

These degradation handlings have also to be coordinated by an appropriate scheduling. But in addition to that, even the handling of a single particular resource degradation may comprise multiple particular activities or action to be done, in order to guarantee a complete handling of the degradation. These particular actions may be executed subsequently or partly in parallel, and additionally the actual execution of such an action may be dependent on the outcome of a previously executed action (compare the handling options and their combinations for g_{r1} and g_{r2} outlined on p. 152).

notion of recovery actions

Consequently, it seems reasonable to let a particular *recovery plan (RPlan)* consist of multiple so-called *recovery actions (RActs)*. Each recovery action

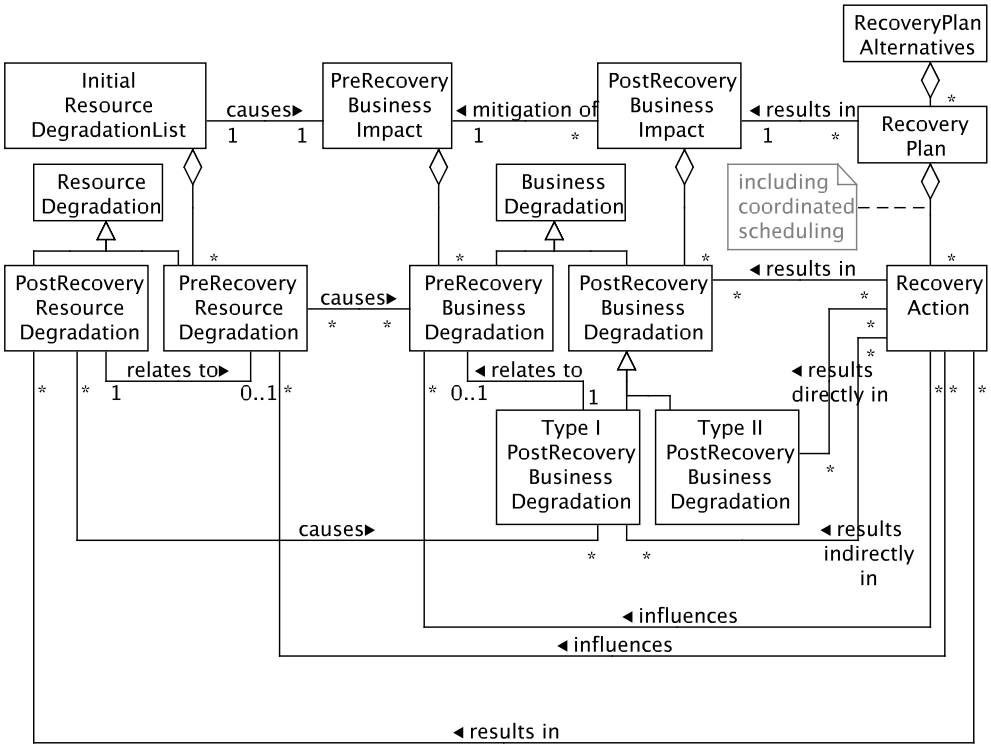

Figure 4.20: Refined RA situation considering multiple recovery actions in relationship to specific business degradations (refinement of Fig. 4.17)

takes part in the handling of one or multiple pre-recovery resource degradations. Concerning this, a recovery plan comprises information necessary for describing the future, actual execution of each recovery action. Furthermore, each recovery action contains appropriate priority and scheduling information in order to determine whether, when, and how in relationship to others a particular recovery action is going to be executed.

Thus, the notion of recovery actions and their coordinated scheduling concretizes the abstract notion of handling of degradations and their respective scheduling, having been introduced at the beginning of this section.

influence and result of recovery actions

Each specific recovery action may *influence* one or multiple of the pre recovery business degradations, and correspondingly may *result* in one or multiple post-recovery business degradations. The terms *influence* and *result* here are used in a similar sense as introduced on p. 157 for the overall recovery plan. That is, *influence* is used with reference to pre-recovery degradations, and *result* with reference to post-recovery degradations. Using these conventions, the aggregation of all influence of all recovery actions on a particular pre-recovery degradation represents the influence of the recovery plan as a whole on this particular pre-recovery degradation. Likewise, the aggregation of all results of all recovery actions on a particular post-recovery degradation represents the result of the recovery plan as a whole on this particular post-recovery degradation.

Influence and result of recovery action on a resource degradation may be classified as negative or positive, depending on whether the recovery action contributes in elimination/reduction or in aggravation/new-introduction of the influenced/resulting resource degradations.

Via its influenced pre-recovery resource degradations a recovery action indirectly influences pre-recovery business degradations. Likewise, via its resulting post-recovery resource degradations a recovery action indirectly results in Type I post-recovery business degradations. In addition to that, a recovery action can directly result in Type II post recovery business degradations, i.e., recovery costs for realizing the recovery action.

recovery actions of $ExRecPlan1$

In case of $ExRecPlan1$ for the example situation $ExSit1$, which is entailed by the resource degradations g_{r1}, g_{r1b}, and g_{r2}, the handling of each resource degradation has to be concretized by a sequence of particular recovery actions. For the handling of g_{r2} with combined handling option $DgrHndl(g_{r2}, 1+2+3+4+5)$ this means (compare introduction of the handling option on p. 152):

First, it should be checked whether a reboot of the AFS device is possible and might result in stable continuation of work (recovery action $RecAct(g_{r2}, 1)$). If this handling fails, the next option is the fast preparation of a backup AFS device for replacing the broken one (recovery action $RecAct(g_{r2}, 2)$). As this will result in a reduced performance and so the threat of new resource degradations, the software update history of the device should now be checked in order to find out if a roll-back to previous version might help (recovery action $RecAct(g_{r2}, 3)$). Next, the possible restriction of other AFS users could be evaluated (recovery action $RecAct(g_{r2}, 4)$). Finally and to some part done in parallel to the previous ones, the acquiring of a replacement for the device should be considered if the previous recovery actions failed (recovery action $RecAct(g_{r2}, 5)$). Consequently, the recovery plan $ExRecPlan1$ using handling option $DgrHndl(g_{r2}, 1 + 2 + 3 + 4 + 5)$ includes multiple recovery actions for handling resource degradation g_{r2}. Other handling options for g_{r2} are concretized by respective combinations of recovery actions.

Similarly also the handlings of the other resource degradations are concretized. The simple recovery handling options $DgrHndl(g_{r1}, 1)$, $DgrHndl(g_{r1}, 2)$, $DgrHdnl(g_{r1}, 3)$ for resource degradation g_{r1} can be directly mapped to specific recovery actions $RecAct(g_{r1}, 1)$, $RecAct(g_{r1}, 2)$, $RecAct(g_{r1}, 3)$. Consequently, the more complex recovery handling options for g_{r1}, obtained by combining some or all of these simple ones, can be mapped to sequences of the respective recovery actions.

information details of recovery actions

In the following, the information to describe recovery actions in the recovery plan, is discussed, but only on an abstract level. However the introduction of detailed and specific data structures for this is deferred until the discussion of the recovery analysis framework in Sect. 4.4. Fig. 4.21 illustrates the abstract details of recovery actions and as a comparison also provides the information details of degradations (compare to Fig. 4.7 on p. 135).

Generally speaking, the information about recovery actions in a recovery plan

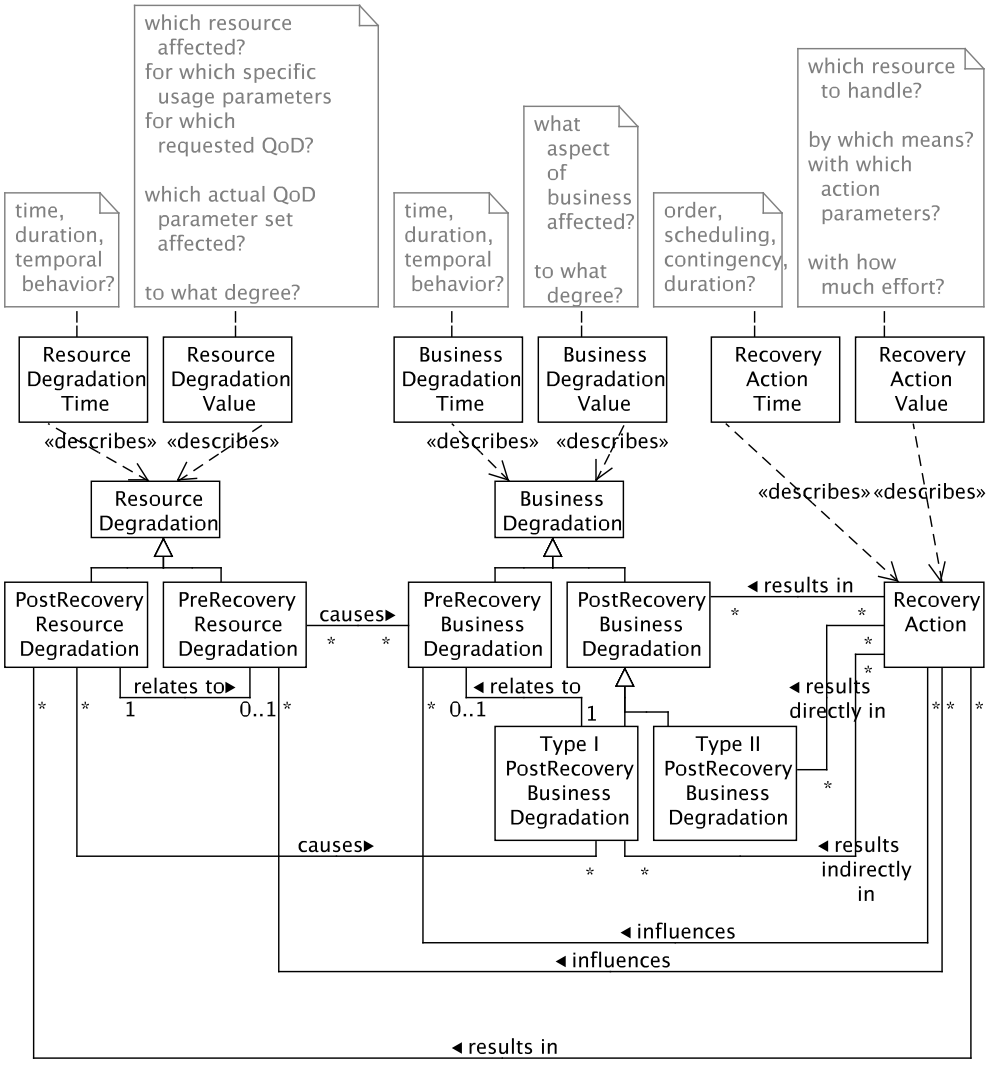

Figure 4.21: Abstract aspects and details of recovery actions in comparison to the aspects and details of degradations (compare to Fig. 4.20)

has to be detailed enough to allow later-on the appropriate and coordinated execution of all recovery actions of a selected recovery plan.

There are the two general abstract parts of information to describe a recovery action being part of a recovery plan: informations relating to time domain and information relating to value domain of the recovery action:

time domain of recovery actions

The *time domain for recovery actions* is mainly concerned with temporal course as well as coordination of multiple recovery actions: aspects as order, or more detailed specific scheduling, contingency (conditional execution), as well as the duration of the recovery actions are subsumed under this term.

value domain of recovery actions

By contrast the *value domain of recovery actions* is concerned with the specific details of a particular recovery action without much taking into account the other recovery actions: Generally, the targeted resource to be handled and somehow the manner or means how it is handled have at least to be specified. In addition to the information about the specific resource also information

about the QoR/QoD parameters degradations which are influenced by the recovery action may be specified. Furthermore, the information about the manner for handling the resource often includes further parameters concretizing the manner as well as information about the degree of estimated repair effort (e.g., expressed in man hours and/or in financial costs). The probability of success (possibly as a function of repair duration) may also be specified if the specific recovery manner is known to not succeed in any case.

This subdivision of information for describing recovery actions is similar to the subdivision of information for describing resource degradations, service degradations, and business degradations treated in Sect. 4.2.2.1 on p. 134: Along these lines, specifically the information about the affected resource is called the *recovery action subject*, the information about the handling manner and possibly the influenced QoS parameters is called the *recovery action manner*. Correspondingly, both recovery action subject and recovery action manner are subsumed under the term *recovery action scope*. The recovery action scope mainly describes, or more accurately classifies, the specific type of recovery action, i.e., is concerned with the *value granularity* of the information of describing recovery actions. In contrast further specifications about e.g., effort necessary, probability of success are more related to the *value accuracy* of the recovery action information.

execution information

The introduced aspects and details of recovery actions, i.e., recovery action time domain and recovery action value domain and the subdivision of the latter, are mainly concerned with the specification of the future execution of a particular recovery action as part of a recovery plan. That is why they are all subsumed under the term *recovery action execution information*.

The execution information is directly concerned with recovery action, but in general it is also related to the information about the recovery degradations influenced by or resulting from this recovery action: The scope (relating to value granularity) of the recovery action (subject + manner) can be used to derive the scope of the post-recovery resource degradations (and thereby indirectly post-recovery business degradations) which are influenced - negatively or positively - by this recovery action. Similarly, the information about value accuracy of a recovery action can be used for deriving information about the value accuracy of influenced degradations, i.e., mainly the degree of estimated value change of a degradation (per time). Finally, the information about time domain of a recovery action can be used for deriving the change/shift in time development of influenced/resulting degradations.

influence/result information

The specific information about the influence on pre-recovery degradations by a recovery action is called *recovery (action) influence information*. Correspondingly, the specific information about the result of a recovery action, i.e., the post-recovery degradations resulting from the recovery action (compare p. 157), is called *recovery (action) result information*. In Fig. 4.21 influence information and result information are represented indirectly via the “influences” and “results in” associations respectively, whereas in contrast execution information is represented directly by the classes “recovery action time”

and “recovery action value”.

example of execution information

For example, execution information for the recovery action $RecAct(g_{r2}, 1)$ (see p. 164) includes the following: The subject is the restarted AFS device. The handling manner is repeatedly rebooting/restarting, failing in case of the interval between required, successive restarts is too short (< 5 min). The time domain includes the position/priority for scheduling (executed first, immediately), further on the expected duration for a single reboot (ca. 3 min).

determination of recovery alternatives

Whereas up to now the structure of the essential output artifacts of RA has been analyzed and determined, in the following it is discussed how these output artifacts, i.e., the list of recovery alternatives along with their respective reduced impact, is actually determined by RA. This is done only on an abstract level, as the specific data structures for realizing the recovery plans, recovery actions, as well as degradations are not to be treated before Sect. 4.3 and Sect. 4.4, where the impact analysis framework and the recovery analysis are actually presented.

On an abstract level, the following can be said about the determination of recovery alternatives: It is based on the pre-recovery business impact given by IA, which in general is consisting of multiple pre-recovery business degradations being caused by multiple pre-recovery resource degradations. Therefore, some kind of evaluation and priorization of pre-recovery business degradations as well as the pre-recovery resource degradations has to be done.

In the usual case, before taking recovery decisions, a provider evaluates and prioritizes the different business degradations, based on some kind of ranking comparing the value domain of the different business degradations. E.g., the one with the highest values (over time) should be handled before the others, i.e., a corresponding recovery plan should - if somehow possible - consider recovery actions having influence on this business degradation to be executed first with appropriate effort and manner.

requirements for recovery plans

Generally speaking, this evaluation and prioritization of business degradation has the purpose of setting requirements on the recovery alternatives to be designed. As explained previously (compare p. 151 and Fig. 4.16), pre recovery impact describes the current impact situation. In contrast, post-recovery impact describes a lastly targeted impact situation, which - along with its corresponding recovery alternative necessary to reach it - has to be designed by RA, and which should be as most as possible approximating an ideal impact situation. Using these terms, the evaluation and prioritization of pre-recovery business impact, does some kind of requirement analysis for the targeted impact situation before performing the actual design of the targeted impact situation and its corresponding recovery alternative.

rating of the example situation

In the following, evaluation and prioritization is shown for the example situation $ExSit1$ and its recovery plan $ExRecPlan1$ introduced on p. 151. $ExSit1$, initially introduced on p. 130 in Sect. 4.2.2.1, was actually introduced as an extension of the situation in the example I/RA run presented in

Sect. 2.3.4. There, for rating the specific costs (more specifically, SLA penalties) over (elapsed) time are compared.

As motivated by the corresponding impact analysis performed in Sect. 2.3.4, the business degradations entailed by resource degradation g_{r2} (AFS cell outage) are more severe than the ones of resource degradation g_{r1} (high IP link utilization). So, they get assigned the highest priority. This even stays true, if g_{r1} is combined with g_{r1b} (DNS server problem) like in $ExSit1$, which does not yet lead to aggravated business degradations. This combination of resource degradations is only making the handling of its entailed caused business degradations more complex, as probably both resource degradations have to be handled in order to significantly reduce the entailed business degradations.

Concluding, the business degradations entailed by g_{r2} always have the highest recovery priority, in the case of the I/RA example run of Sect. 2.3.4 as well as in the case of example situation $ExSit1$. The other business degradation, g_{b1-1} entailed by g_{r1} and g_{r1b} gets a minor priority.

business impact rating

All the actions necessary for evaluation and prioritization of pre-recovery business impact are comprised by the term *business impact rating* in the following, as the different pre-recovery business degradations are somehow compared among each other and rated accordingly.

I.e., while IA identifies each business degradation and further-on collects all necessary information about each of them, business impact rating unifies this collected information so that the degradations can be directly compared: Aspects to be unified are e.g., different business degradation types (e.g., SLA violation costs vs. revenue loss of dynamically subscribed service), different instances of the same business degradation type (e.g., SLA violation costs for different services), and specifically different temporal development behavior of a business degradations (i.e., different time/value relationship patterns over time). Moreover, rating of different business degradations is always heavily dependent on the specific requirements and policies of the service provider. An example is the preference for the problems of a customer, with which the provider has been having a relationship for years, vs. similar, equally severe problems of a more short-term customer.

indirect rating of resource degradations

Such a rating for each pre-recovery business degradation, indirectly determines also a rating of the pre-recovery resource degradations which (via service degradations) entail these pre-recovery business degradations. This indirect determination can be actually performed by the two impact dependency model (see p. 140 in Sect. 4.2.2.2) applied in the reverse direction, because these models describe the mapping of resource degradations to business degradations (via service degradations). The more business degradations with high priority a resource degradation is entailing, the higher is its assigned priority.

example of indirect rating

In order to continue the example of business impact rating performed for the $ExSit1$ above (see p. 167), the corresponding indirect resource degradation

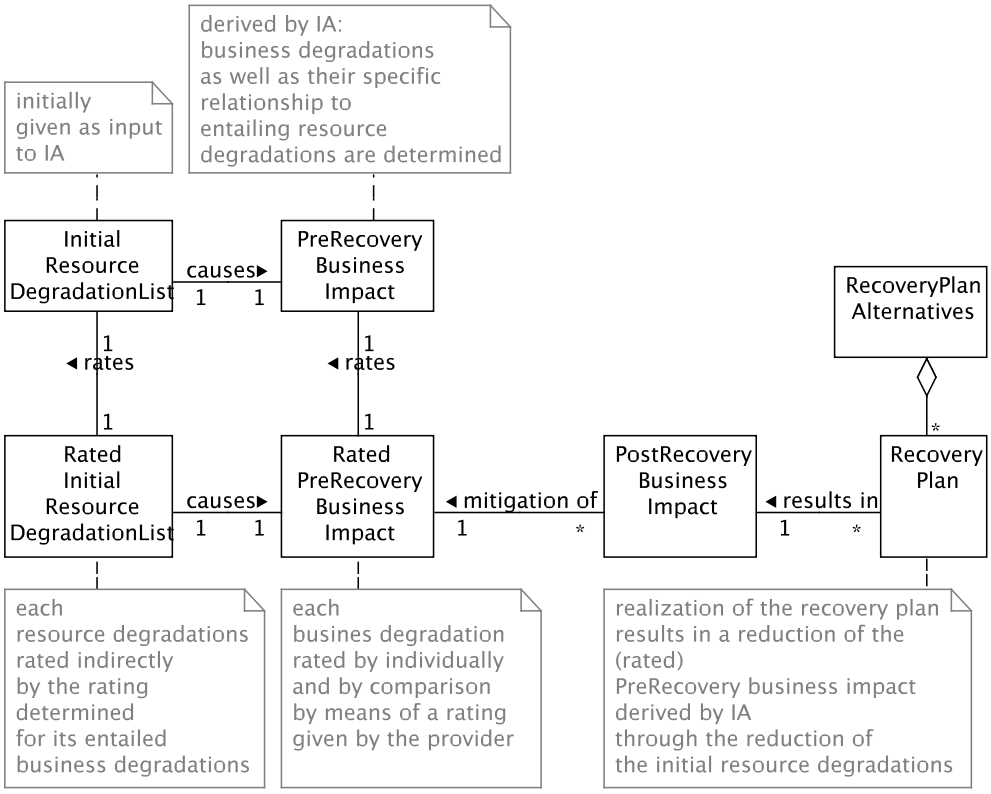

Figure 4.22: RA situation taking into account rating of pre business impact (refinement of Fig. 4.17)

rating is shown: Because the business degradations entailed by g_{r2} have been assigned the highest recovery priority, correspondingly g_{r2} gets the highest priority among the resource degradations. So, any recovery plan for $ExSit1$ should focus on g_{r2} first with as much as effort necessary, and afterwards should try to handle g_{r1} and g_{r1b}.

Concerning the comparison of the remaining resource degradations g_{r1} and g_{r1b}, the following can be said: both resource degradations are contributing to the high avg. mail sending delay of $6\ \mathrm{min}$, which exceeds the limit negotiated in the SLA of $5\ \mathrm{min}$. As the normal value of this avg. mail sending delay is about $3\ \mathrm{min}$. That is, the current value of $6\ \mathrm{min}$ means an exceed of $3\ \mathrm{min}$ compared to the normal value. In fact, $0.5\ \mathrm{min}$ ($6\ \mathrm{min}$ with g_{r1} and g_{r1b} compared to $5.5\ \mathrm{min}$ with only g_{r1}) of this aggravation are ascribed to the degradation g_{r1b}, whereas to g_{r1} the remaining $2.5\ \mathrm{min}$ of this exceed above normal are ascribed. Therefore, the priority assigned to g_{r1} should be higher than the one assigned to g_{r1b} (5 times as large, as $2.5/0.5 = 5/1$).

From the priorities assigned to the initial resource degradations, the actual order/scheduling for their recovery handlings can derived: The recovery plan $ExRecPlan1$ for $ExSit1$ should be concerned first with the handling of g_{r2} paying much effort, second with less effort to the handling of g_{r1b}, and last with probably even less effort to the handling of g_{r1}. In fact, the example scheduling order $DgrHndlSched1$ for $ExRecPlan1$ (see p. 151) is defined to meet exactly these requirements.

issue for using impact dependency models

Nevertheless, the reverse application of the impact dependency models for indirect resource impact rating, i.e., the mapping from rated business degradations to rated resource degradations is not completely trivial, because the relationship between resource degradations and business degradations is not always 1:1, but in generally can be m:n. E.g., in $ExSit1$ g_{r1} and g_{r1b} are together entailing g_{b1-1}. So, in order to use the impact dependency models, originally introduced for IA, for indirectly rating resource degradations, some possibility to specifically compare resource degradations, which entail some business degradations together, has to be foreseen. The actual data structure for realizing both impact dependency models, being discussed in Sect. 4.3, will have to be designed with taking into account this issue.

rated impact artifact

Consequently, for RA an additional, intermediary artifact (between its essential input and its essential output) is introduced, namely the *rated (pre recovery) business impact*, which is produced from the corresponding un-rated pre-recovery business impact (essential input of RA) by business impact rating. This *rated (pre-recovery) business impact* is consisting of *rated (pre-recovery) business degradations*, which are (via service degradations) entailed by *rated (pre-recovery) resource degradations*. Concluding, the initial (pre-recovery) resource degradation list given as input to I/RA is indirectly rated via the rated pre-recovery business impact, and results in a *rated initial resource degradation list*.

The direct rating of pre-recovery business impact, and the indirect rating of pre-recovery resource degradations (above all of the initial resource degradations) derived from the business impact rating, are both comprised by the generic term *(pre-recovery) impact rating*. Fig. 4.22 depicts the RA situation taking into account impact rating.

After rating the business degradations and thereby indirectly also the resource degradations, the actual design of recovery alternatives along with their resulting reduced impact takes place. This *recovery plan design* uses the rated pre-recovery business impact as input artifact and has as output artifact the overall output artifact of RA, i.e., namely the recovery alternatives.

recovery analysis = impact rating + recovery design

Summing up, recovery analysis is subdivided into the two steps *impact rating* and *recovery plan design* (or *recovery design* in short). For this reason, the recovery analysis subworkflow BAWf.2 (second step of Fig. 4.3) is accordingly subdivided into two corresponding steps. Fig. 4.23 shows the result of this refinement, namely BRAWf.2.

4.2.3.2 Identification and analysis of additional artifacts

further artifacts for RA

Actually, now all essential input and output artifacts of RA haven been identified and analyzed. Still an open question is how to actually derive from the respective essential input artifacts the essential output artifacts during RA. I.e., concerning business impact rating, how to rate the pre-recovery impact given by IA, and concerning recovery plan design, how to design recovery plan alternatives from the rated pre-recovery impact. In the following, further

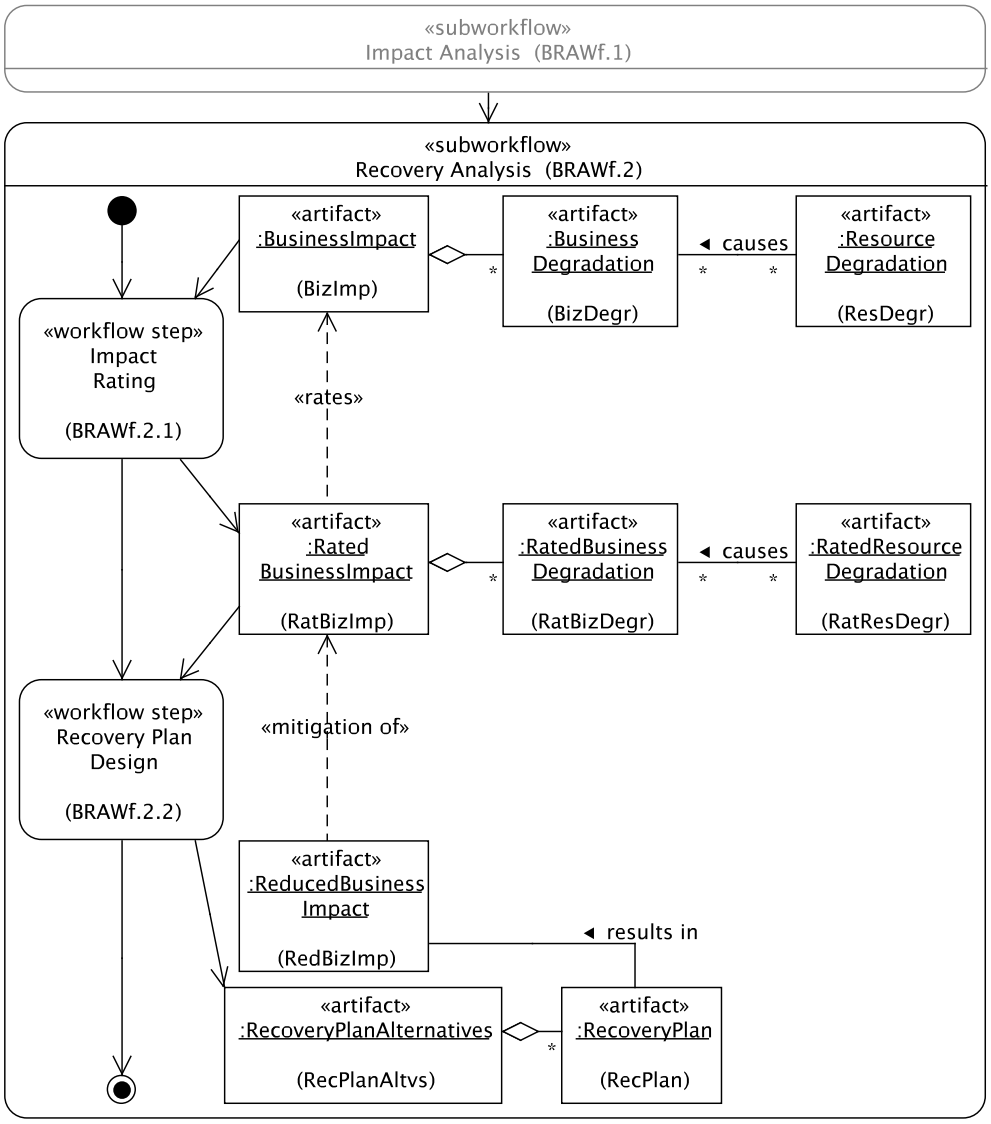

Figure 4.23: Basic refined abstract subworkflow of recovery analysis with all essential basic input and output artifacts (BRAWf.2, refinement of BAWf.2 in Fig. 4.3)

artifacts for answering the questions are identified and discussed. Fig. 4.24 shows a refined RA situation with these additional artifacts. For clarification and comparison, also the two impact dependency models used for IA (compare p. 140 in Sect. 4.2.2.2) are shown, too. For each step of RA, impact rating as well as recovery plan design, additional artifacts are introduced.

impact rating model

As introduced previously, a rating of pre-recovery business degradations (as well as indirectly of pre-recovery resource degradations) has to be performed. This rating is very depending on the specific demands and policies of the service provider, i.e., the rating is concerned with the specific ranking of the different business degradations by the service provider. Therefore, a new artifact, a so called *business impact rating model*, is introduced. It is specifically given by the provider in order to rank/rate business degradations according

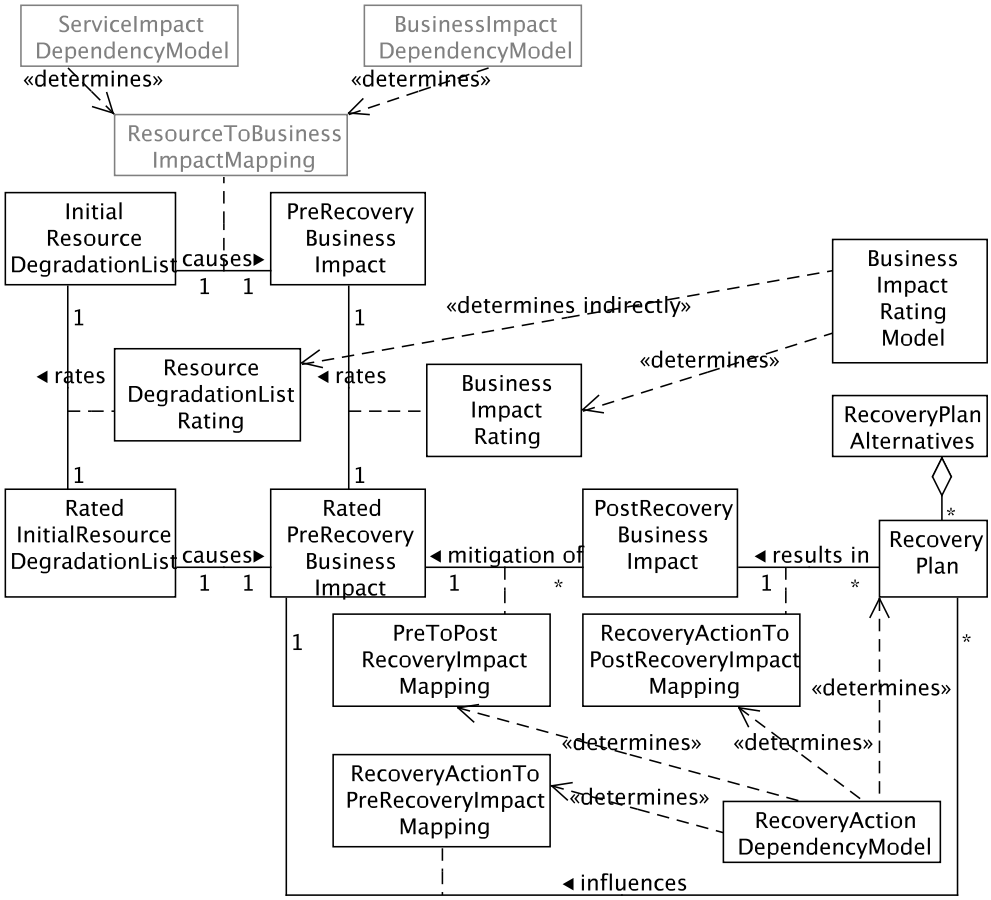

Figure 4.24: RA situation with rating model and recovery action dependency model (refinement of Fig. 4.22)

to the needs of the service provider and to allow to derive appropriate requirements for the recovery plans to be designed accordingly. Based on the specific ranking of the business degradations, a ranking for the entailing resource degradations is also provided indirectly by the *business impact rating model*. That is why it is also called *impact rating model* in general.

In the example situation $ExSit1$ (see p. 167) as a rating model the threatening costs over elapsed time, as a unified metric, are used. In general, this can become more complex, as different types of business degradations (e.g., SLA violation costs and expected revenue loss for a highly dynamically subscribed service) have to be rated in a unified way and suitable for the needs of the service provider. Additionally, the service provider possibly also wants to include weighting of degradations of the same type. For instance, a problem of a long-term customer, with which the provider has a customer relationship for years, may get a higher rating than a similar problem of a short-term customer (weighting of SLA violation costs of different customers).

recovery (action) dependency model

For the recovery plan design, that is the actual design of recovery plan alternatives along with their specific reduced impact based on the rated pre recovery impact, information about all possible recovery actions, their specific execution information details as well as their specific influence/result on the

pre/post-recovery impact is necessary.

For this reason, a further artifact is introduced: a so-called *recovery action dependency model* or *recovery dependency model* in short, which is named along the lines of impact dependency models used for IA. But while impact dependency models (see p. 140 in Sect. 4.2.2.2) allow to determine dependencies among and between various kind (resource, service, business) degradations, recovery dependency models allow to determine for a (rated) degradation all possible recovery actions as well as their specific influence on the initially threatening or further introduced degradations. The recovery dependency model is used to plan and devise all possible recovery action schedules (a recovery plan), each together with its estimated reduced business impact. Fig. 4.25 visualizes this specific relationship between recovery action dependency model, recovery actions (as parts of recovery plan) and degradations (as parts of impact) in detail.

example for recovery dependency model data

Examples of pieces of information described by the recovery dependency model in the case of the example situation $ExSit1$ and its recovery plan $ExRecPlan1$ (see pages 151, 152, 164, and 167) are: The various handling options for each resource degradation (see p. 152); how they are mapped to recovery actions along with appropriate scheduling (see p. 164); the execution information of all recovery actions (see p. 167)); further the influence/result of each recovery actions: E.g., the fact that recovery action $RecAct(g_{r2}, 2)$ for handling business degradations entailed by g_{r2} additionally results in an aggravation of the service degradation g_{s1-2}, which was initially entailed by g_{r1b}, and therefore also results in an additional business degradation entailed from post recovery degradation g_{s1-2}.

contents of the recovery dependency model

The recovery dependency model allows to determine for each (rated) resource degradation all possible recovery actions along with their specific parameters, effort as well as their specific influence on post business impact. Consequently, conceptually on an abstract level, the recovery dependency model contains for each possible (rated) resource degradation all possible recovery actions, and further allows to derive all necessary information concerning the recovery action. This information comprises the following (compare p. 164):

execution information: time domain as well as value domain (granularity/scope and accuracy).

influence information: pre-recovery degradations influenced (eliminated, reduced, aggravated) by this recovery action (possibly partially in connection with further recovery actions).

result information: post-recovery degradations for whose state (negative or positive in comparison to pre-recovery state) the recovery action is (possibly in connection with further recovery actions) responsible for.

In fact, these three types of information - at least on an abstract level - are very inter-related: E.g., influence on a particular pre-recovery degradation relates specifically to a result as a particular post-recovery degradation Furthermore,

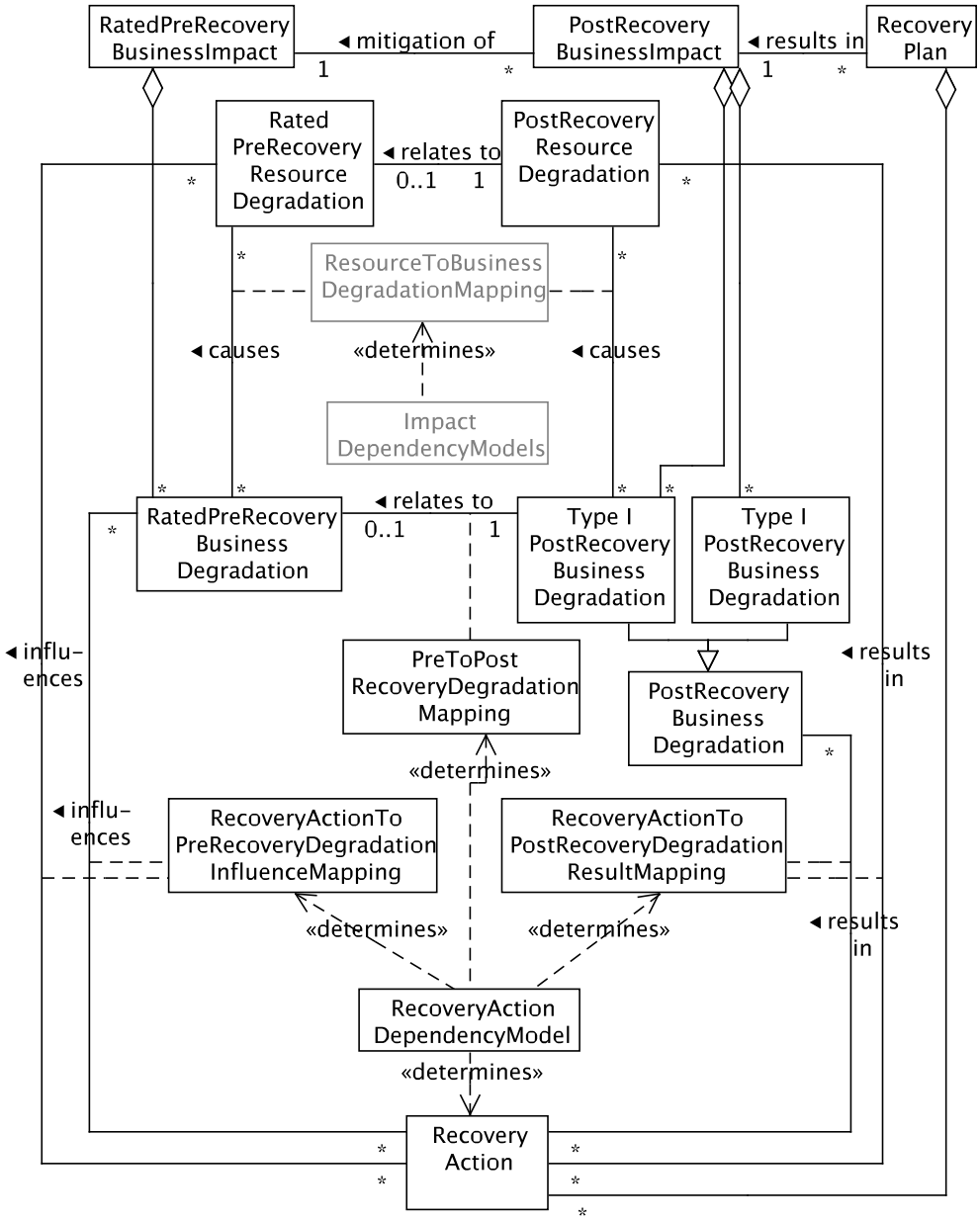

Figure 4.25: Detailed relationship between recovery action dependency model, recovery actions, and degradations (refined part of Fig. 4.24 with some details of Fig. 4.20)

specific values for action parameters as e.g., amount of effort (part of execution information), are related to a specific value range for the value domain the resulting degradations. A similar relationship holds for the time domain of recovery actions and the time domain of resulting degradations.

mappings between recovery actions and degradations

Related to these parts of information for recovery actions are various mappings between recovery actions and degradations. These mapping can be regarded from the point of view of particular degradations or regarded for the complete impact altogether as a whole (compare Fig. 4.25):

- *recovery (action) to pre-recovery degradation influence mapping* and *re-*

covery (action) to pre-recovery impact influence mapping, described by the recovery (action) influence information.

- *recovery (action) to post-recovery degradation result mapping* and *recovery (action) to post-recovery impact result mapping*, described by the recovery (action) result information.
- *pre to post-recovery degradation mapping* and *pre to post recovery impact mapping*, representing the difference information of the former two mappings.

All three types of mappings shall be subsumed unter the term *recovery mappings* in the following.

use of the recovery mappings

The recovery to pre-recovery degradation influence mapping is specifically useful for identifying all possible recovery actions for handling a particular pre-recovery degradation. Or reversely, by using this influence mapping all pre-recovery degradations influenced by a given recovery action can be found. By contrast, the recovery to post-recovery degradation result mapping is useful for deriving the actual (estimated) benefit of a recovery action resulting from its execution. Finally, the pre to post-recovery degradation mapping is useful for directly comparing the influenced pre-recovery degradations and resulting post-recovery degradations of a recovery action. Therefore, in a sense it represents the difference between influence mapping and result mapping of a recovery action.

The actual order of use of the three types of recovery mappings for the recovery plan design depends on the particular methods used for realizing the recovery plan design. For instance, one possible order for the use of the recovery mappings for designing a recovery plan is starting from rated pre-recovery degradations by the use of the influence mapping to identify all possible recovery actions and secondly filter these actions by their resulting degradations, which are determined by the result mapping. So, in this case recovery plan and its recovery actions are directly determined by evaluating their resulting targeted impact situation. But depending on the actual used method, also other orders of use are possible, e.g., using the pre to post-recovery mapping first in order to more deeply design the targeted situation, and second by using the result mapping (in the reverse direction as used previously) in order to determine the actual recovery actions to reach the targeted situation. Even, interleaved versions of such orders are possible, e.g., first in coarse-grained manner identifying possible recovery actions, second determining their resulting degradation also in coarse grained manner, and afterwards refining the specification of the recovery actions in order to optimize the design. Concluding, all three types of recovery mapping are useful for recovery plan design, but the order of their usage (possibly interleaved multiple usage) varies depending on the specific method used for realization of recovery plan design. For this reason, the recovery dependency model was introduced as a single abstract artifact, and was not subdivided into multiple specific artifacts which determine a particular one of the three types of recovery mappings.

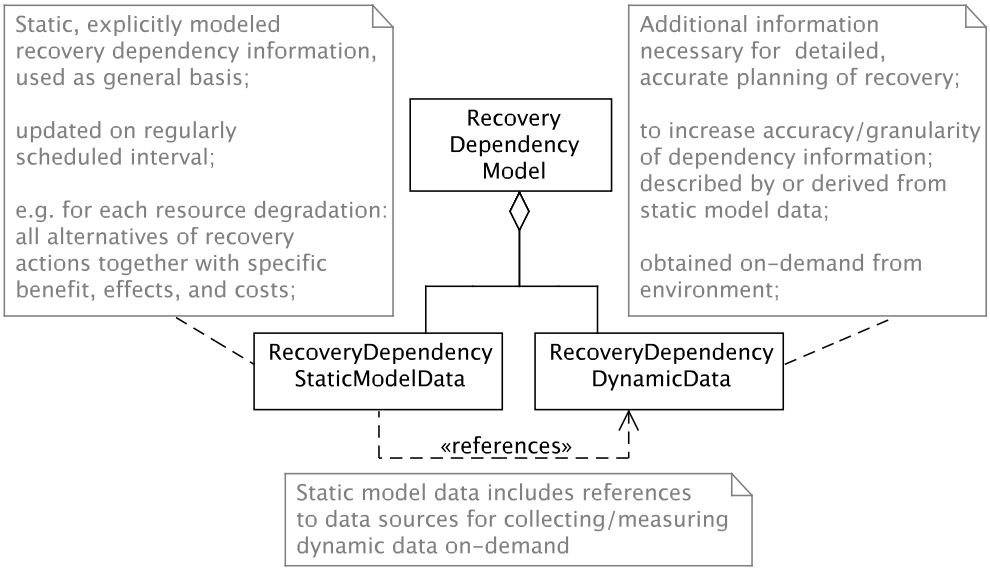

Figure 4.26: Recovery dependency model composed of proper, static recovery dependency model data and additional recovery dependency dynamic data

generic structure of recovery dependency model

As this discussion of recovery analysis is done on an abstract level, the actual data structures used for realizing the recovery dependency model are not discussed here. Moreover, as these data structures heavily depend on the actual realization method of recovery plan design, their introduction is deferred until the treatment of the recovery analysis framework in Sect. 4.4.

Here only a generic statement about the structure of the recovery dependency model is made: Similar to impact dependency models (see p. 143 in Sect. 4.2.2.2), for recovery dependency models a differentiation is made between explicit (static) modeling data, which is more static in nature and explicitly modeled, and additional dynamic data, which is more dynamic in nature and accessed/gathered/measured only on-demand. Consequently these parts are called *recovery (action) dependency static model data* and *recovery (action) dependency dynamic data*, and the recovery dependency model as an artifact is subdivided into these two. Fig. 4.26 shows the relationship between these two types of data (compare to Fig. 4.10 on p. 144 in Sect. 4.10).

static model data

Recovery dependency proper model data is explicitly modeled and used as a basis for deriving the recovery plan. This type of data is directly and without considerable costs/time accessible by recovery plan design, because it is updated and synchronized with information from the provider's service provisioning/management infrastructure on regularly scheduled intervals.

additional, dynamic data

By contrast, recovery dependency dynamic data comprises additional information which complements, details, refines in granularity, and makes more accurate the proper model data. Its data sources, which are known and referenced by the proper model data, are accessed on demand whenever recovery plan design needs such more detailed, more fine-grained, and more accurate recovery dependency information. But the access to this kind of data normally

also results in higher costs and time for the data access.

examples of static/dynamic data

Examples for static model data in the case of $ExSit$ are the various recovery handling options and their known result/influence on pre/post impact. But specific details may be gathered on demand dynamically, e.g., for recovery action $RecAct(g_{r2}, 2)$ the information about whether a backup AFS device is actually available, for recovery action $RecAct(g_{r2}, 3)$ whether any software update has recently done, for recovery action $RecAct(g_{r1}, 1)$ whether limiting/prioritizing of other IP traffic is currently somehow possible.

refined abstract RA subworkflow

After having identified all artifacts necessary for RA, the recovery analysis subworkflow (BRAWf.2, see Fig. 4.23) can be completely refined, taking into account all of these artifacts. Fig. 4.27 shows the result of this refinement, which yields the complete version of the basic refined abstract recovery analysis subworkflow, BRAWf.2. Actually, the refined abstract RA subworkflow now also includes the business impact rating as additional input artifact for impact rating (BRAWf.2.1), as well as recovery (action) dependency proper model data and recovery (action) dependency dynamic data as additional input artifacts for recovery plan design (BRAWf.2.2).

general analogies for I/RA with focus on RA

Concluding the analysis of the recovery analysis subworkflow, the following general analogies concerning I/RA - mainly seen from the point of view of RA - are given (compare p. 151, Fig. 4.16, and discussion on p. 167) in the following and are summarized in Table 4.4.

I/RA step	abstract artifact/goal	general analogy
impact analysis (IA)	determination of current impact situation	problem statement
recovery analysis (RA)	determination of targeted impact situation	
(pre-recovery) impact rating	requirements for targeted impact situation	requirements analysis
recovery plan design	design of targeted impact situation	design phase
(recovery realization) and recovery tracking	realization of targeted impact situation	realization phase

Table 4.4: Analogies for I/RA steps, especially from RA point of view

First, impact analysis (IA, done before RA) is responsible for the *statement of problem* concerning recovery, as its output, the pre-recovery impact, describes the current impact situation (i.e., the *the problem*). Second, impact rating performs some sort of *requirements analysis* for the recovery, as it identifies the requirements for targeted impact situation through rating of degradations. Third, recovery plan design covers - as the name already suggests - the *design phase* of the recovery, i.e., the design of targeted impact situation as well as the design of recovery plan to reach it. Lastly, the recovery actually executed - which is not covered by I/RA framework explicitly - together with recovery

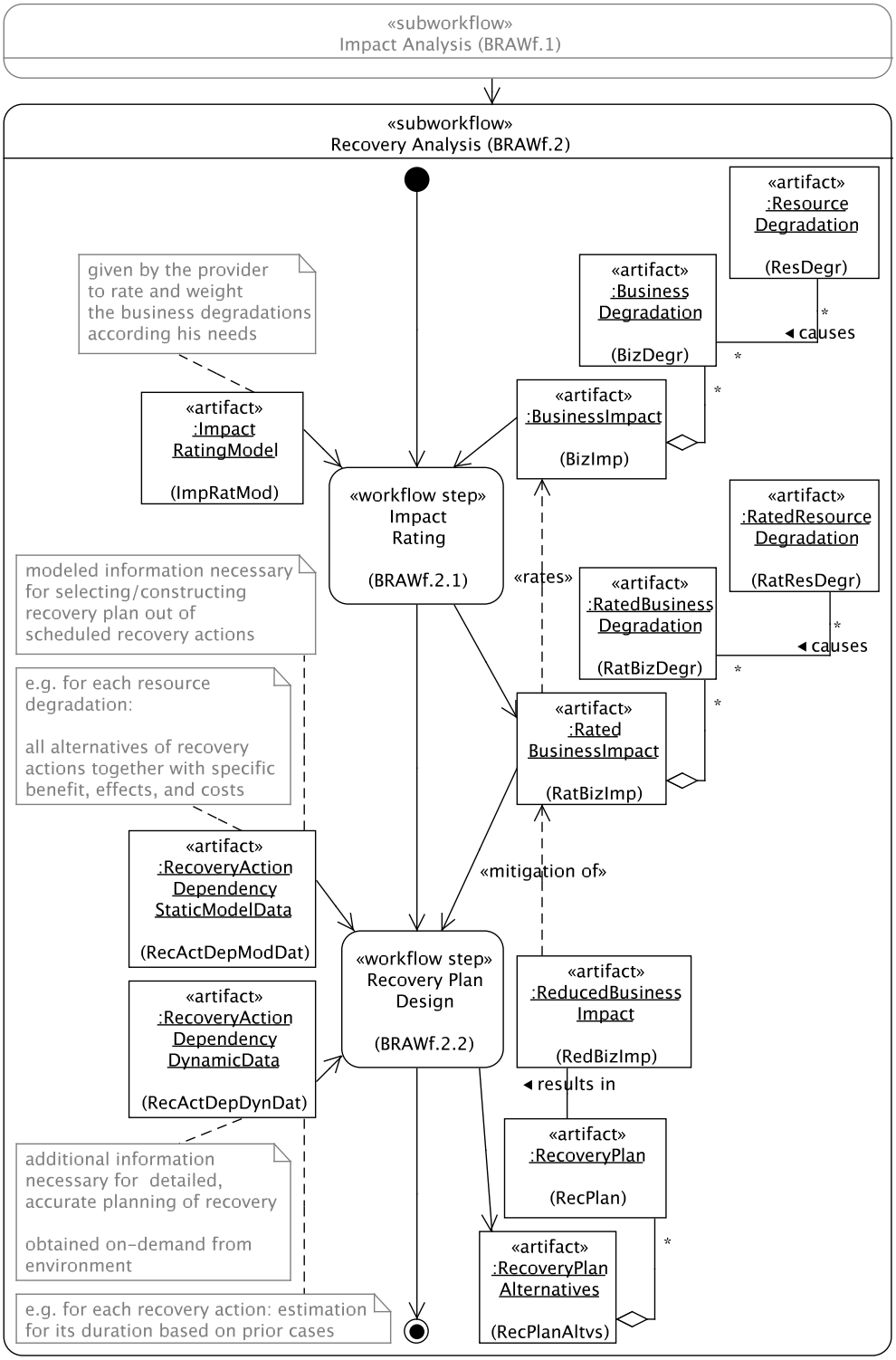

Figure 4.27: Basic refined abstract subworkflow of recovery analysis with all necessary artifacts (BRAWf.2, refinement of Fig. 4.23)

tracking (both done in parallel) cover the *implementation or realization* of the designed recovery.

4.2.4 Basic abstract subworkflow for recovery tracking

Here, the recovery tracking subworkflow of the basic abstract workflow, BAWf.3 (third step in Fig. 4.3) is discussed in detail and refined into BRAWf.3 accordingly.

4.2.4.1 Analysis of essential artifacts

basic situation of recovery tracking

Fig. 4.28 illustrates the basic situation for *recovery tracking (RT)*, which is treated in the following.

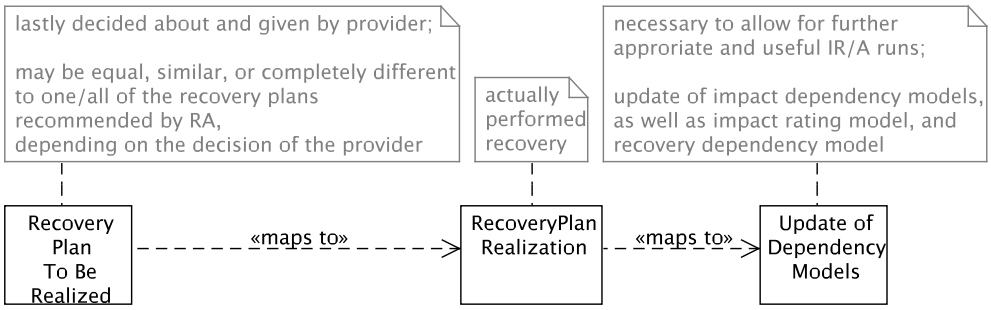

Figure 4.28: Basic situation of recovery tracking

essential input of recovery tracking

After performing recovery analysis, which recommends one or multiple recovery alternatives, each described by a recovery plan and its estimated reduced post recovery impact, the actual selection for a specific recovery alternative to be realized has to be done by the service provider. This actual selection now serves recovery tracking as its essential input in order to prepare for tracking of important changes to the service infrastructure resulting from the performed recovery, and accordingly to consolidate and synchronize all necessary I/RA model data with these changes. This consolidation of model data with the changes in the actual service infrastructure is necessary in order to allow for appropriate impact and recovery analysis in the future for resource degradations which are related to the changed service infrastructure.

I/RA models to update

In general, the model data which has to be updated and consolidated comprises all models necessary for performing IA and RA, i.e., both impact dependency models (see p. 140 in Sect. 4.2.2.2), the impact rating model and the recovery (action) dependency model (see p. 171 in Sect. 4.2.3.2), as well as any additional models necessary for performing recovery tracking itself.

However, as recovery actions taken into account by this I/RA framework only target directly at the resource layer, and not directly at the service layer nor even the business layer (compare p. 157 in Sect. 4.2.3.1), in most of the cases

the business impact model should not need any update. This is even more true for the business impact rating model which is more dependent on the provider's policies than on the service infrastructure. Nevertheless, an update also of these both models could be necessary and so in general this is taken into account here.

examples of model update

As continuation of the example situation $ExSit1$ and its recovery plan $ExRecPlan1$ (see pages 151, 152, 164, 167, and 173 in Sect. 4.2.3) examples for model updates are given in the following: The update of the SI dependency model (see p. 140 in Sect. 4.2.2.2) concerning the changed service dependencies for the AFS service when the broken AFS device is replaced with a backup device (with less performance) as result of recovery action $RecAct(g_{r2}, 2)$ (compare p. 164 in Sect. 4.2.3.1). A similar statement holds for the addition of an new IP link resulting from the recovery action $RecAct(g_{r1}, 3)$. Specifically, for the replaced AFS device, the recovery dependency model (see p. 172 in Sect. 4.2.3.2) has to be updated to reflect that in future this backup device is no longer available, and other alternatives should be added.

recovery plan overridden by provider

In the selection process the provider may even modify the recommended recovery plan if it seems not to fit it needs. There may be anytime additional circumstances which are not taken into account by I/RA framework and so it is necessary to allow for such external modifications to the recovery plan by the provider. Lastly, it is the actual decision of the provider which recovery alternative should be realized, and I/RA framework can only make recommendations. It should be noted here, that I/RA framework is designed as a support for impact and recovery analysis, which assists the human operators in the recovery decision, e.g., by fetching and correlating degradation information from various data sources. But IR/A framework does not strictly prescribe a recovery alternative to be done.

Nevertheless, recovery tracking has in any case to get the actual recovery plan to be realized as input and have the possibility to update and consolidate its necessary data models with the changes in the external service infrastructure. Otherwise its future operation will become impossible or at least inaccurate.

notion of recovery changes

In fact - as already stated - the recovery plan realization, i.e., the actual execution of the planned recovery, is not covered by I/RA framework explicitly. Recovery tracking is performed in parallel to or (e.g., depending on expected duration) after this recovery plan realization, in order to allow for the tracking of changes of the service infrastructure resulting from the realization of the recovery plan. Any recovery plan realization may result in multiple *recovery changes*. Above all, this may be true if a recovery plan has to handle multiple resource degradations. But of course not each handling of a resource degradation has to result in a recovery change.

examples of recovery changes

Sometimes an appropriate recovery handling may be only a reboot of a component or a restart of a crashed process, which does not lead to a change, as in recovery action $RecAct(g_{r2}, 1)$. However, e.g., a recovery handling which modifies the dependencies of a service on resources (as far as the SI depen-

dency model is concerned) has to be tracked by recovery tracking and the SI dependency model has to be updated accordingly. A specific example is the already above mentioned replacement of broken AFS device by a backup device with reduced performance resulting from recovery action $RecAct(g_{r2}, 2)$. A further example is the resulting changed routing configuration for other IP traffic not originating from the mail service resulting from recovery action $RecAct(g_{r1}, 2)$.

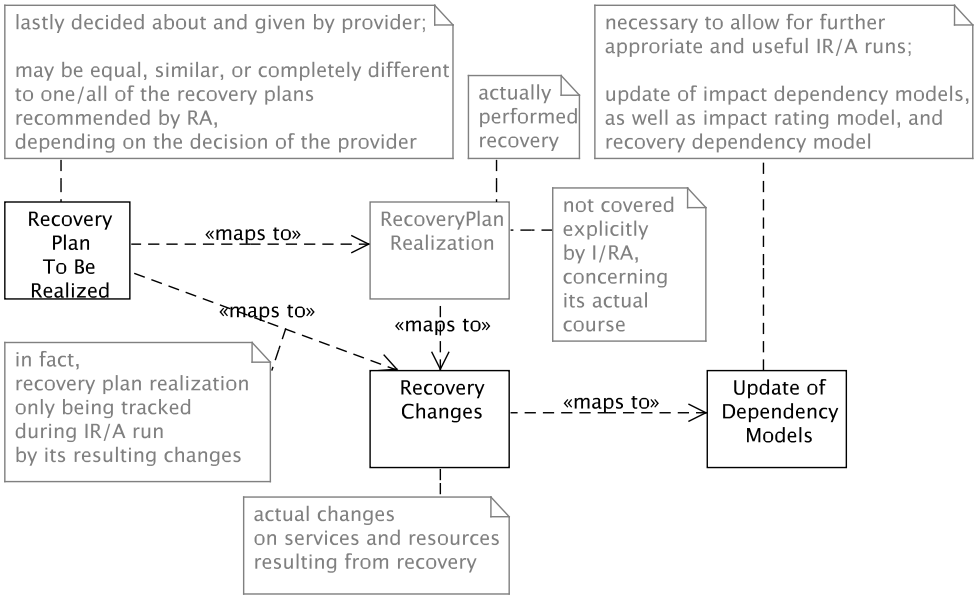

Figure 4.29: Situation of recovery tracking including recovery changes (refinement of Fig. 4.28)

mapping recovery plan to recovery changes

The information about which types of service changes can be expected for the selected recovery alternative, has to be determined from selected recovery plan by recovery tracking first. Second, recovery tracking has to monitor the determined types of recovery changes, and afterwards to update I/RA model data accordingly.

Concluding, recovery tracking takes the provider's selection of a recovery plan to be realized as input, uses this to derive information about which types of service changes have to be monitored, actually monitors such changes, and lastly updates and synchronizes all I/RA model data with related external information sources. Fig. 4.29 illustrates this slightly refined situation for recovery tracking taking into account recovery changes.

For instance, in case of recovery action $RecAct(g_{r2}, 2)$ the actual change of the AFS devices has to be monitored, and in case of recovery action $RecAct(g_{r1}, 2)$ the update of IP routing configuration has to be monitored.

As indicated above, recovery tracking comprises two general activities:

- *recovery changes tracking*: determination of types of recovery changes to monitor, and actually monitoring of such types recovery changes.
- *model adaption*: updating and synchronizing I/RA model data related to the actually monitored recovery changes.

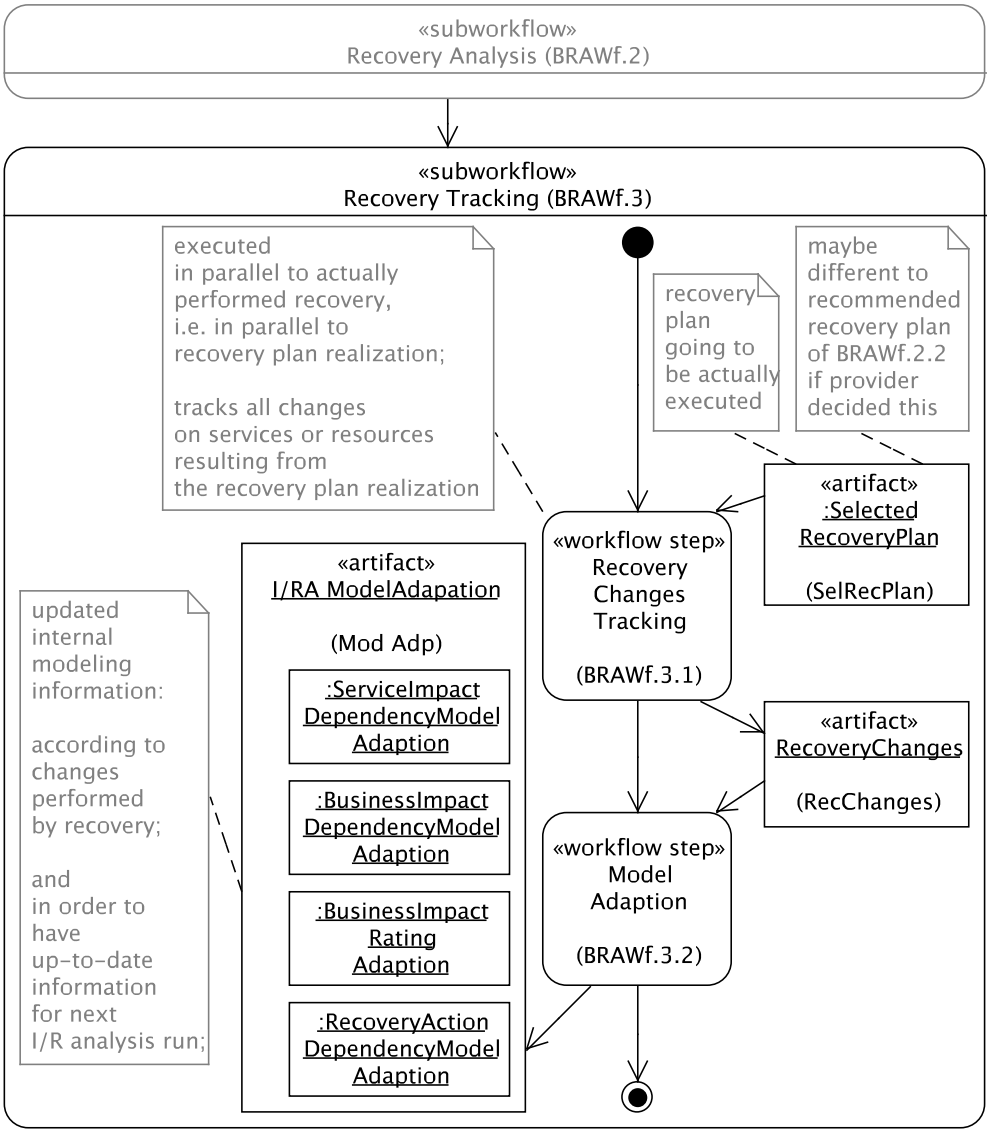

Figure 4.30: First version of basic refined abstract subworkflow of recovery tracking (BRAWf.3, refinement of BAWf.3 in Fig. 4.3)

refinement of recovery tracking subworkflow

Consequently, the recovery tracking subworkflow is refined into these two steps. Recovery changes tracking, the first of the two steps, is concerned with the determination of which types of recovery changes have to monitored, and with the actual monitoring and collecting of these types of recovery changes. It has the selected recovery plan as an input artifact and the observed recovery changes as its output artifact. These observed recovery changes are in turn

the input artifact for model adaption. Finally, model adaption has as its output artifacts the updates of the different I/RA models. Fig. 4.30 shows the resulting refinement of BAWf.3, namely BRAWf.3, consisting of both steps and including all artifacts identified so far.

aggregation of recovery changes

The output artifact of recovery changes tracking, i.e., the observed recovery changes, are in general determined in the following way: First, individual recovery changes are monitored during or after the recovery plan realization. Afterwards these individual recovery changes have to be appropriately aggregated in order to really be a useful input for model adaption. During this aggregation also some kinds of data preparation such as removing of duplicates, joining of corresponding information parts, adding of synchronized time stamps can be performed.

For example, in the case of the changes resulting from recovery action $RecAct(g_{r1}, 2)$ the individually monitored recovery changes are IP routing changes which have to be aggregated to the complete resulting routing situation.

Taking into account, the particular monitoring of individual recovery changes and their aggregation, yields a further refinement of the situation of recovery tracking, illustrated in Fig. 4.31.

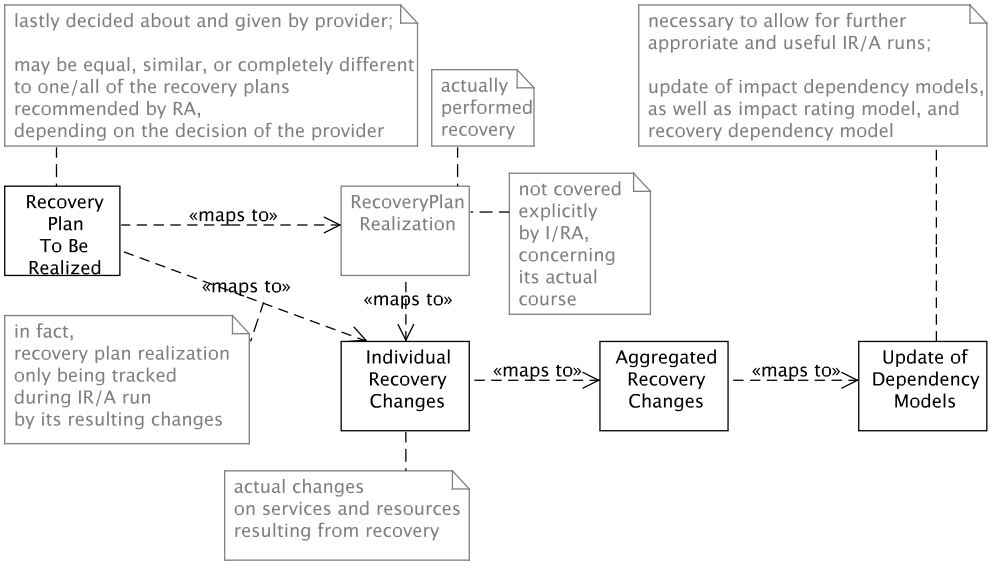

Figure 4.31: Situation of recovery tracking taking into account individual and aggregated recovery changes (refinement of Fig. 4.29)

further refinement of subworkflow

Consequently, the recovery tracking subworkflow in Fig. 4.30 can be further refined concerning its artifacts: Recovery changes tracking, the first step, gets the *individual recovery changes*, which are actually gathered by monitoring the service infrastructure, as a further input artifact. Additionally, its output artifact, previously called service changes, is renamed to *aggregated recovery changes* to reflect the aggregation of individual recovery changes performed in this step. Fig. 4.32 shows this further refined version of subworkflow of recovery tracking BRAWf.3.

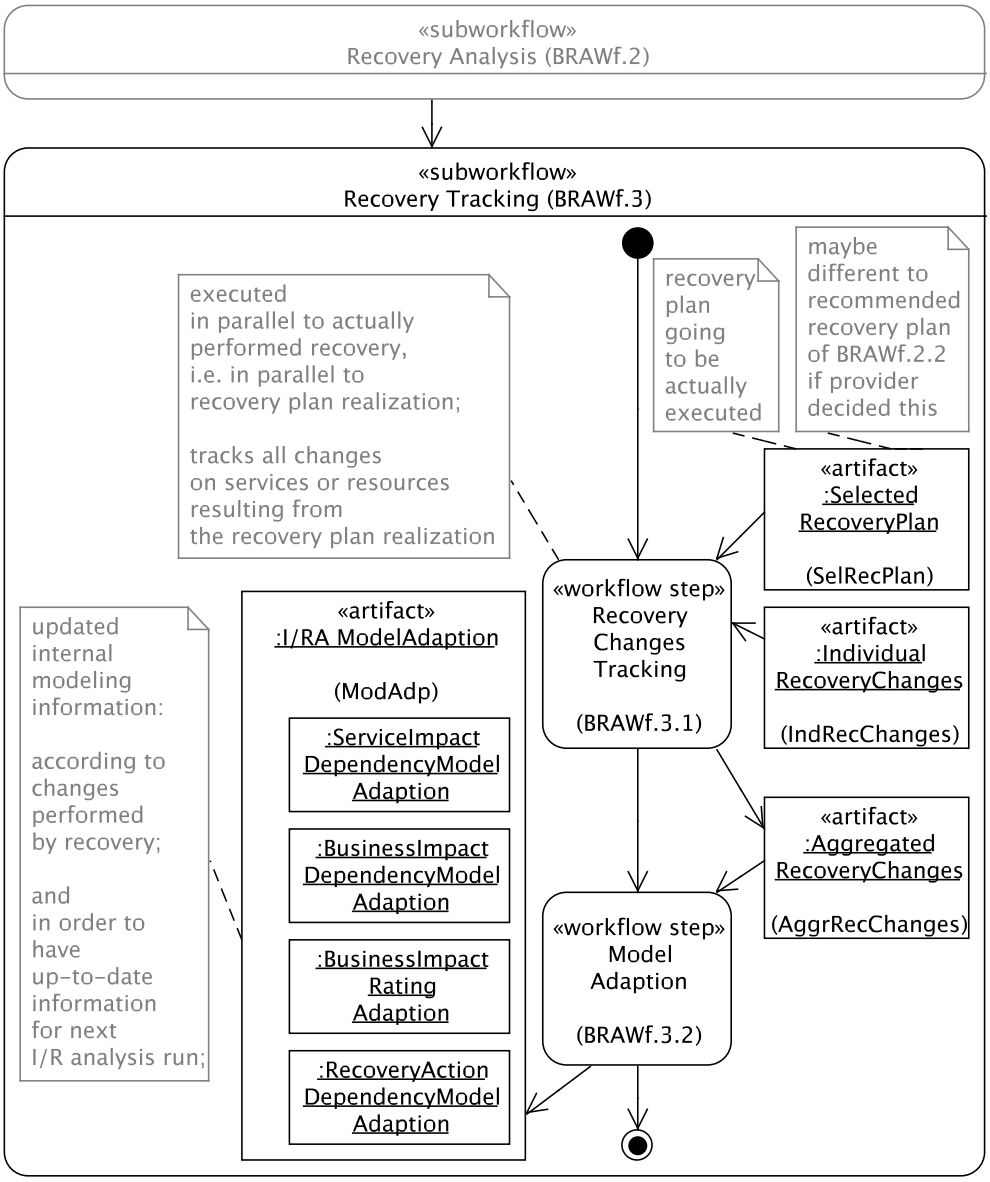

Figure 4.32: Second version of basic refined abstract subworkflow of recovery tracking with all essential artifacts (BRAWf.3, refinement of Fig. 4.30)

4.2.4.2 Identification and analysis of additional artifacts

Up to now all essential input and output artifacts necessary for recovery tracking have been identified and the recovery tracking subworkflow has been refined accordingly. In the following, additional artifacts necessary for recovery tracking are identified and analyzed. These additional artifacts are in general concerned with the question how to actually derive all the various mappings in Fig. 4.31.

Fig. 4.33 illustrates the refined situation of recovery tracking including additional artifacts, which are discussed in the following.

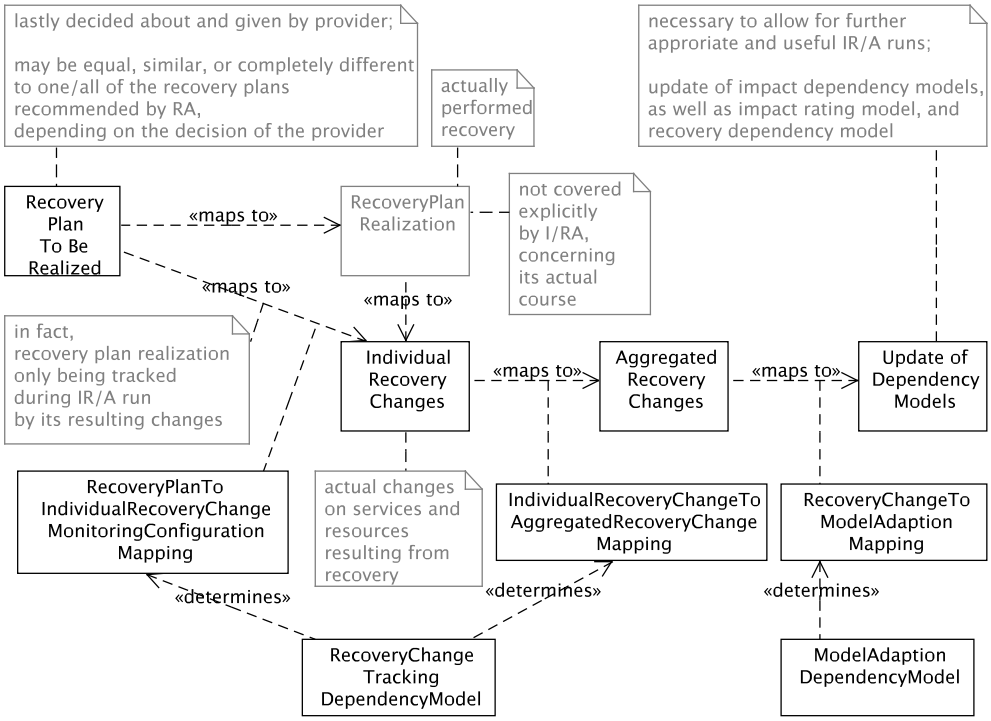

Figure 4.33: Situation of recovery tracking taking into account necessary dependency models (refinement of Fig. 4.31)

recovery change tracking dependency model

As motivated on p. 182 in Sect. 4.2.4.1, recovery changes tracking (BRAWf.3.1) is concerned with three consecutively executed but inter-related tasks, namely the identification of types of recovery changes to be monitored by examining the recovery plan, the actual monitoring of the recovery changes, and finally the aggregation of these recovery changes. Therefore, additional information is necessary mainly for the first and the last of these tasks, i.e., information for mapping the selected recovery plan to the types of changes to be monitored (*recover plan to individual recovery change monitoring configuration mapping*), and information about how to aggregate such monitored changes (*individual recovery change to aggregated recovery change mapping*). Because these pieces of information are somehow related, they are covered by a single, additionally introduced artifact, a so-called *recovery change tracking dependency model*, which actually allows to describe and determine both mappings in an integrated, appropriate manner.

example

For $ExSit$ and its recovery action $RecAct(g_{r1}, 2)$, first, the recovery change tracking dependency model includes, e.g., a description of the different types of individual recovery changes to observe for $RecAct(g_{r1}, 2)$, namely types of recovery changes indicating routing changes (*recover plan to individual recovery change monitoring configuration mapping*). There may be different such types of recovery changes, e.g., specific *SNMP (Simple Network Management Protocol)* traps, and observations by protocol analysis of the routing protocol traffic at different important locations in the network. Secondly, the recovery change tracking dependency model describes how to aggregate these observed, individual recovery changes indicating routing changes in order to

get a consolidated list of the actual changes of the routing situation (*individual recovery change to aggregated recovery change mapping*).

model adaption dependency model

Furthermore, for model adaption (BRAWf.3.2) information is necessary for actually mapping aggregated recovery changes to updates/adaption of the various I/RA models (*recovery change to model adaption mapping*). So, the description and determination of this mapping is covered also by an additionally introduced artifact, the so-called *model adaption dependency model*. Both new dependency models introduced for RT are subsumed under the term *RT dependency model*.

example

In case of $ExSit$ and $RecAct(g_{r1}, 2)$, the model adaption dependency model determines, e.g., which parts of the SI dependency model, namely parts of dependencies for the IP service, have to be updated and to be resynchronized with the new situation of the IP routing.

static and dynamic model data

Similarly as done for impact dependency models of IA (compare p. 143 in Sect. 4.2.2.2), and for the recovery dependency model of RA (compare p. 175 in Sect. 4.2.3.2), concerning the abstract structure of both RT dependency models the following can be said yet: Each RT dependency model may comprise static model data as well as additional dynamic data. The former is explicitly modeled, more static in nature as it is synchronized with data from the actual service provisioning/management infrastructure on a regularly scheduled basis, and therefore access delay for this type of data is relatively short and not causing additional effort for RT. By contrast, for additional dynamic data only the respective data source and access methods are known and referenced by the static data model. This data is only accessed by RT on-demand, e.g., in order to detail or make more accurate the information derived from the static modeled data. Its access actually may take some time and cause some additional effort.

adaption of RT dependency model

In the course of model update (BRAWf.3.2), both RT dependency models may have to be adapted/updated itself. That is, specifically the model adaption dependency model may have to be used to describe and determine how itself has to be updated.

The actual design of detailed data structures for both RT dependency models, which is treated in the discussion of the recovery tracking framework in Sect. 4.5, has to take into account this requirement. Nevertheless, the determination of updates to the model adaption dependency model itself may sometimes be too complex to be covered in a fully automated, pre-determined way. That is why, in such complex cases human operators may be forced to make changes to the model adaption dependency model. Nevertheless, a concrete, robust model adaption dependency model maybe designed in such a way that it at least detects such cases and maybe also proposes default values or estimated values in order to partially support human operators concerning their own update.

refinement of RT subworkflow

Concluding, the RT subworkflow (BRAWf.3 in Fig. 4.32) is refined to take into account static and dynamic data for both RT dependency models, as

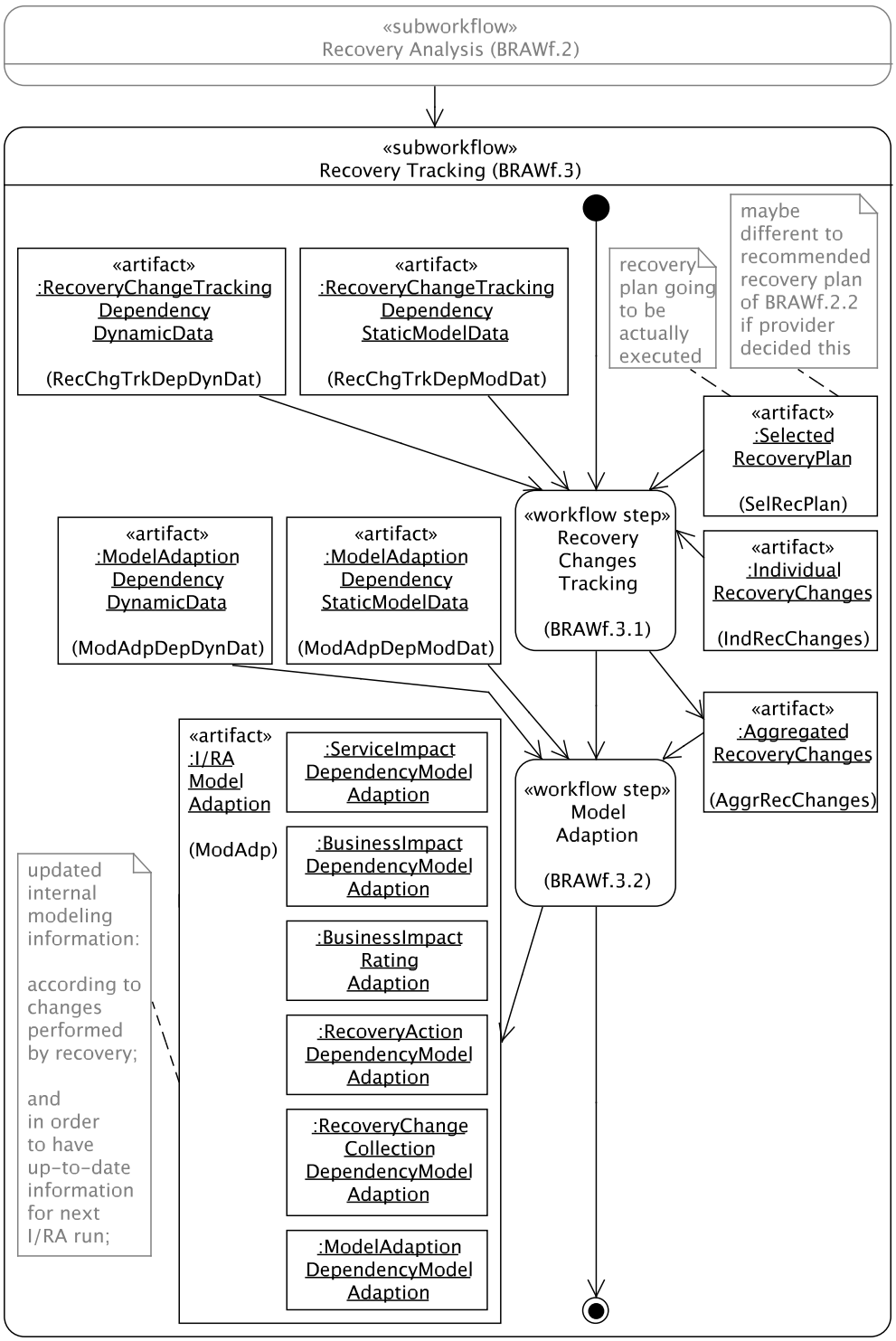

Figure 4.34: Complete basic refined abstract subworkflow of recovery tracking with all artifacts necessary (BRAWf.3, refinement of Fig. 4.32)

well as the possibility of adaption of the RT dependency models themselves. Fig. 4.34 shows this complete version of the basic refined abstract subworkflow of recovery tracking BRAWf.3, which includes all identified artifacts.

4.2.5 Basic component architecture and basic realized workflow

In the following, based on the basic refined abstract workflow (BRAWf) designed in Sect. 4.2.2 to Sect. 4.2.4, a *basic component architecture (BCArch)* for executing this workflow is introduced.

First, generic *basic external interfaces (BExtIfcs)*, which are necessary for exchanging input/output artifacts between the workflow and the external existing *IT service provisioning and management infrastructure (SP/MI)* of the service provider, are identified.

Second, the proper design of an internal component architecture (BCArch) is outlined: *Basic internal components (BIntComps)* are identified, which are required for realizing the basic refined abstract workflow by utilizing the basic external interfaces BExtIfcs.

At last, a *basic realized workflow (BRWf)* is devised as refinement and concretization of the basic refined abstract workflow (BRAWf) in order to be executed by BCArch.

4.2.5.1 Basic external interfaces

overview of BRAWf

Based on the developed basic refined abstract workflow (BRAWf) in Sect. 4.2.2 to Sect. 4.2.4 now external interfaces for the realization of this workflow are identified. Fig. 4.35 gives a complete overview of this basic refined abstract workflow, BRAWf (compare Fig. 4.14 on p. 149, Fig. 4.27 on p. 178, and Fig. 4.34 on p. 187 for details). Fig. 4.35 also presents all abstract input/output artifacts for the steps of the workflow analyzed and discussed in the previous sections.

classification of artifacts

These input/output artifacts can be classified in various ways. A particular differentiation of artifacts was used during the analysis of the specific subworkflows in the previous sections: essential artifacts, which are essential to the whole workflow, or at least essential to a particular subworkflow, in contrast to further additional artifacts for determining how to derive essential output artifacts from essential input artifacts (compare also p. 128 in Sect. 4.2.1).

external and internal artifacts

Another classification of artifacts is more relevant concerning the relationship of the workflow and the provider's existing *IT service provisioning and management infrastructure (SP/MI)*: Some of these input/output artifacts of BRAWf are external to the workflow, i.e., are exchanged with the SP/MI and therefore visible outside the workflow. Examples are the initial *list of resource degradations* representing the essential input of the workflow, and the *SI de-*

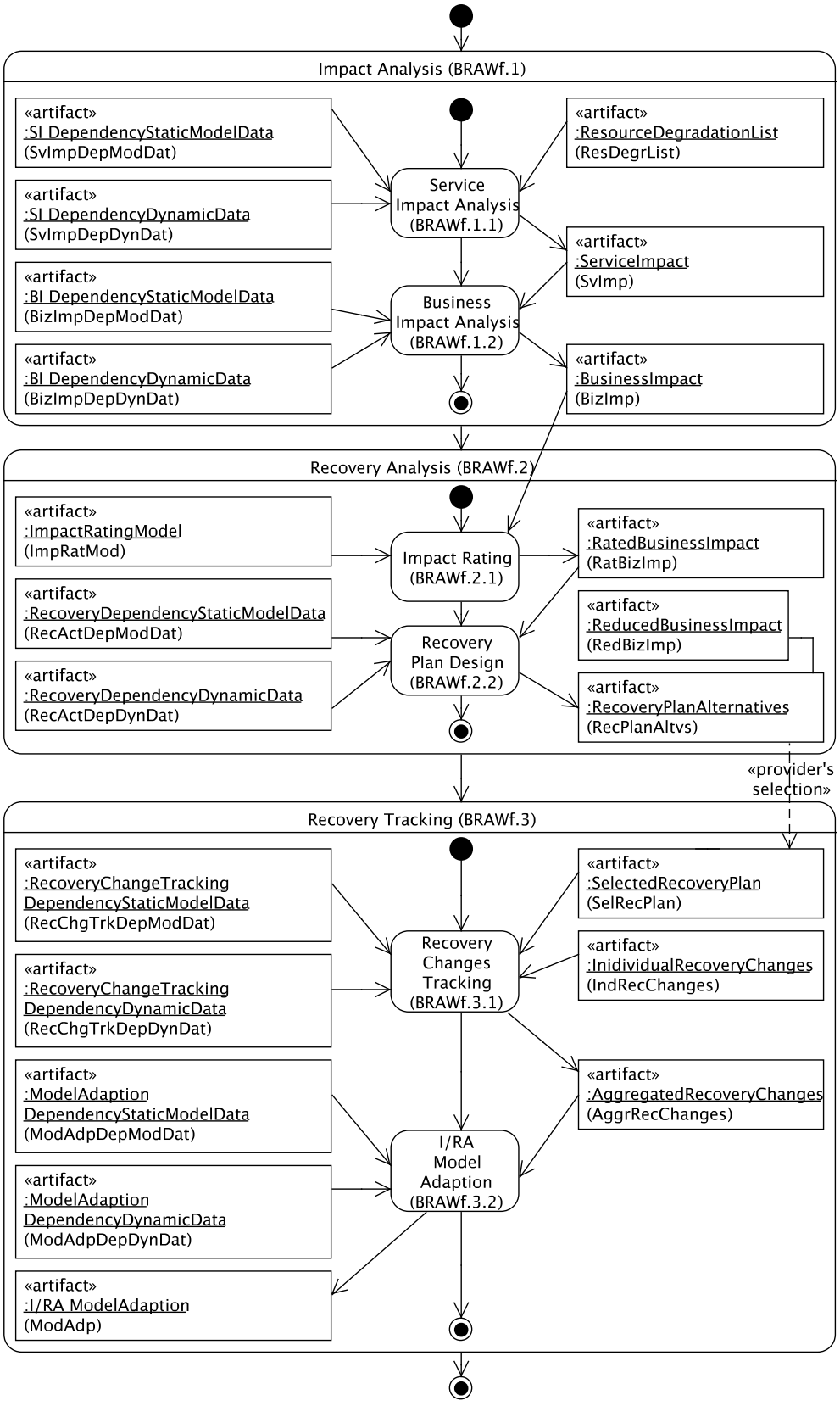

Figure 4.35: Complete overview of basic refined abstract workflow (BRAWf)

pendency dynamic additional data. Other artifacts are completely internal to the workflow, i.e., not visible outside the workflow as being only intermediate artifacts, such as the *service impact* determined by SIA and given as input to BIA. In the following, artifacts external to the workflow are further denoted tersely as *external artifacts*, whereas the others are denoted as *internal artifacts*.

identification of external artifacts

Comparing the artifacts of BRAWf, the following artifacts are clearly external ones:

- the initial resource degradation list (input)
- the recovery plan alternatives together with their respective reduced impact (output)
- the selected recovery plan (input)
- the individual recovery changes (input)
- the model data of the different dependency models (see p. 140 in Sect. 4.2.2.2, p. 171 in Sect. 4.2.3.2, and p. 186 in Sect. 4.2.4.2) as well as the impact rating model (input)
- the additional dynamic data for the different dependency models (input)
- the adaption of model data and additional dynamic data for the different dependency models (output).

role of pre-recovery business impact

All the artifacts listed above have in common that their input or output is fully required in order to correctly perform I/R analysis. In contrast, the (pre-recovery) business impact, output of IA, in the first place is an internal artifact, as it is only intermediate between IA and RA as far as I/R analysis is concerned. But especially for the purpose of immediate notification of the provider about the intermediate information derived so far, after having performed IA and before performing RA, the business impact is added as a special case of external output artifact. By this means, i.e., making the derived business impact an external artifact, the provider can get preliminary informed before actual recovery alternatives are evaluated, e.g., for using the pre-recovery business impact immediately for purposes other than recovery. Concluding, even if from the point of view of the BRAWf pre-recovery business impact is not required to be an external output of the workflow, nevertheless it gets assigned the rank of an external output artifact for the purpose of immediate notification of the provider.

external interfaces

For each external artifact a basic abstract external input/output interface (BExtIfc) is introduced and discussed in the following. Fig. 4.36 gives an overview of these external interfaces and their specific relationship to the steps of BRAWf.

classes of external artifacts

The external artifacts can be subclassified according to the manner in which their actual exchange with the SP/MI by the workflow is actually happening. This yields the following three subclasses of external artifacts: *direct external*

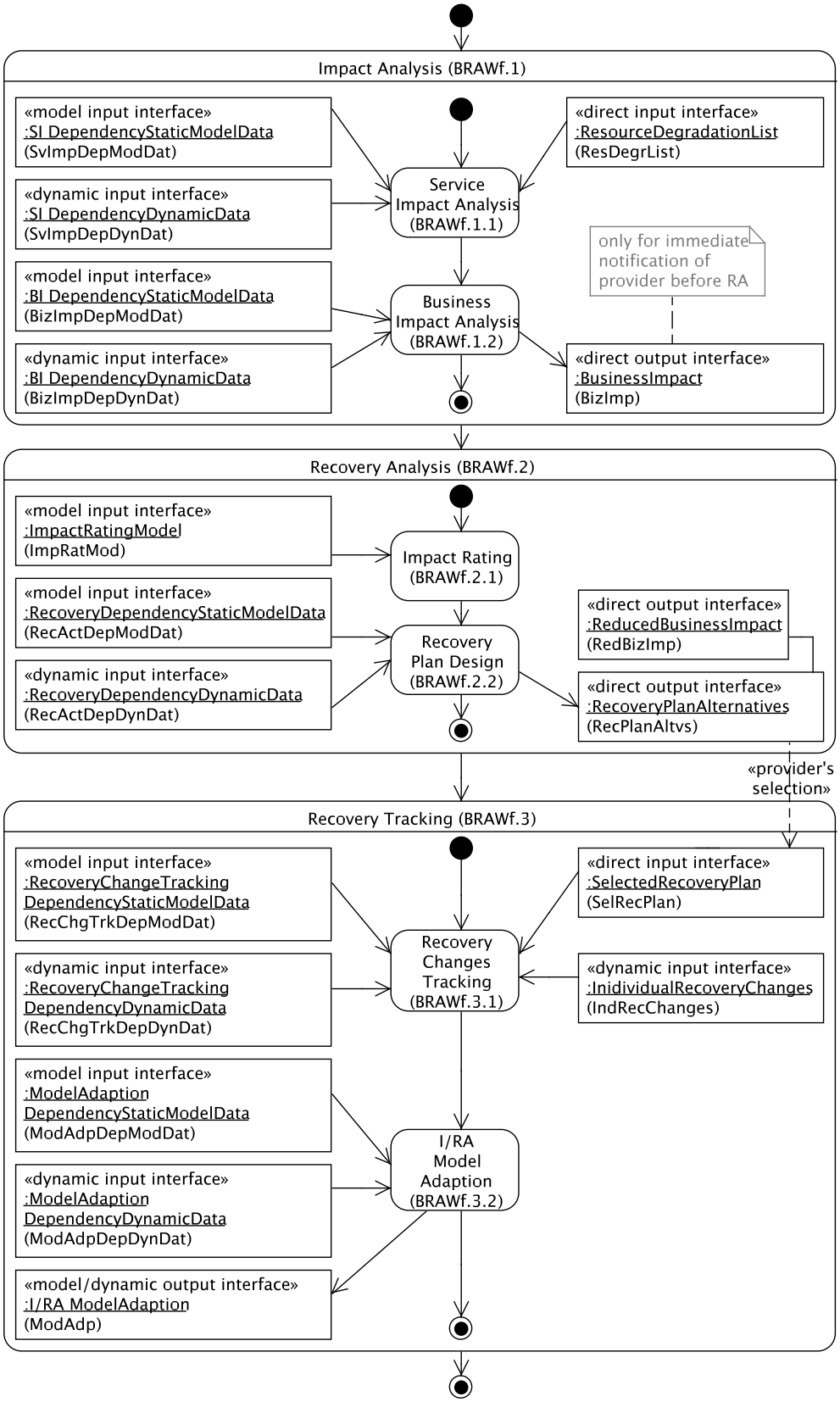

Figure 4.36: Overview of basic external interfaces (BExtIfcs) in relationship to workflow steps of BRAWf (compare Fig. 4.35)

artifacts, *dynamic external artifacts*, and *model (external) artifacts*. Each subclass of external artifacts maps to a specific class of external interface, i.e., there are *direct external interfaces*, *dynamic external interfaces*, and *model external interfaces*.

direct external artifacts

Direct external artifacts are artifacts which are exchanged once with SP/MI at a specific, defined position during the workflow execution. Thus, their single input or output exchange with SP/MI is completely determinable and predictable from the workflow description. Most artifacts which are essential to their subworkflow pertain to these subclass, i.e., the first three artifacts listed above as well as the special external output artifact business impact (see also above).

dynamic external artifacts

The subclass of dynamic external artifacts includes mainly the so-called additional dynamic data for the different dependency models. In contrast to direct external artifacts, the exchange with the SP/MI for dynamic external artifacts is more dynamic, not fully predictable, and depends on the current state of other workflow artifacts. Moreover, the exchange with the SP/MI is not restricted to single exchange instance as for direct external artifacts, rather a particular dynamic external artifact may be accessed on-demand multiple times for different parts/aspects of it, and so also cause multiple exchanges with the SP/MI. In addition to the already mentioned additional dynamic data attached to the various dependency models, the subclass of dynamic external artifacts comprises also the individual recovery changes, which are dynamically observed during recovery change tracking and therefore also involve multiple data exchanges with SP/MI.

role of static model artifacts

The static model data of the various dependency models have an intermediate role between being clearly internal and being clearly external: The data which a static model comprises is externally visible, but the integrated model is visible only internally. So, on the one hand, as being static, explicitly modeled data and being accessible always with not much effort by the workflow they can be considered as internal artifacts. On the other hand, as these model data is regularly synchronized with related external data sources of the SP/MI, in order to allow up-to-dateness of I/RA, they are also clearly external. That is why the static model data of each dependency model has to be taken into account as external artifacts, and so have to be mapped to respective basic external interfaces.

model (external) artifacts

Static model data of dependency models or the impact rating model are different from direct external artifacts, as they are typically accessed multiple times for different parts/aspects of it, actually they may be accessed very often. So, they resemble dynamic external artifacts, but do not share with them the on-demand exchange with the SP/MI for each access instance. Rather, the exchange with the SP/MI is decoupled from the actual workflow access by only periodically synchronizing/updating the model data with the information sources/sinks in the SP/MI. That is why this type of data represents an own subclass of external artifacts, termed the subclass of model (external) artifacts.

Summing up, this results in the following overview of basic external artifacts and corresponding basic external interfaces (compare Fig. 4.36):

- direct external artifacts and respective direct external interfaces
  - initial resource degradation list (input)
  - business impact (output, but solely for immediate notification of the provider about this intermediate result)
  - recovery plan alternatives together with its respective reduced business impact (output)
  - selected recovery plan (input)
- model data artifacts and respective model (external) interfaces
  - static model data of dependency models or the rating model (model input)
  - adaption of static model data of dependency models or the rating model (model output)
- dynamic external artifacts and respective dynamic external interfaces
  - dynamic data for dependency models (dynamic input)
  - adaption of dynamic data for dependency models (dynamic output)
  - individual recovery changes (dynamic input)

4.2.5.2 Basic internal components

Based on the basic abstract interfaces identified before, the proper component architecture (BCArch), being able to realize the basic refined abstract workflow (BRAWf), is introduced. Fig. 4.37 gives a rough overview of the basic component architecture (BCArch) and its relationship to the provider's SP/MI. Additionally, Fig. 4.38 shows the basic internal components (BIntComps), of which BCArch is consisting of, in greater detail.

basic structure

As the framework has to have a tight, appropriate integration with SP/MI (generic requirement R0.1 in Sect. 2.4.1), concerning the model and dynamic external interfaces, it is basically divided into two areas: a *model/dynamic input/output data area (m/d i/o data area)* concerned with the exchange of model/dynamic input/output with SP/MI and a *proper I/R analysis area (proper I/RA area)* concerned with the proper I/R analysis using the model/dynamic external artifacts provisioned by the model/dynamic input/output area.

In the following, first the inner structure of the m/d i/o area is analyzed, afterwards the proper I/R analysis area. The m/d i/o data area comprises the *model/dynamic input/output data access area (m/d i/o data access area)*, and the *model/dynamic input/output data engine (m/d i/o data engine)*, which are both treated next.

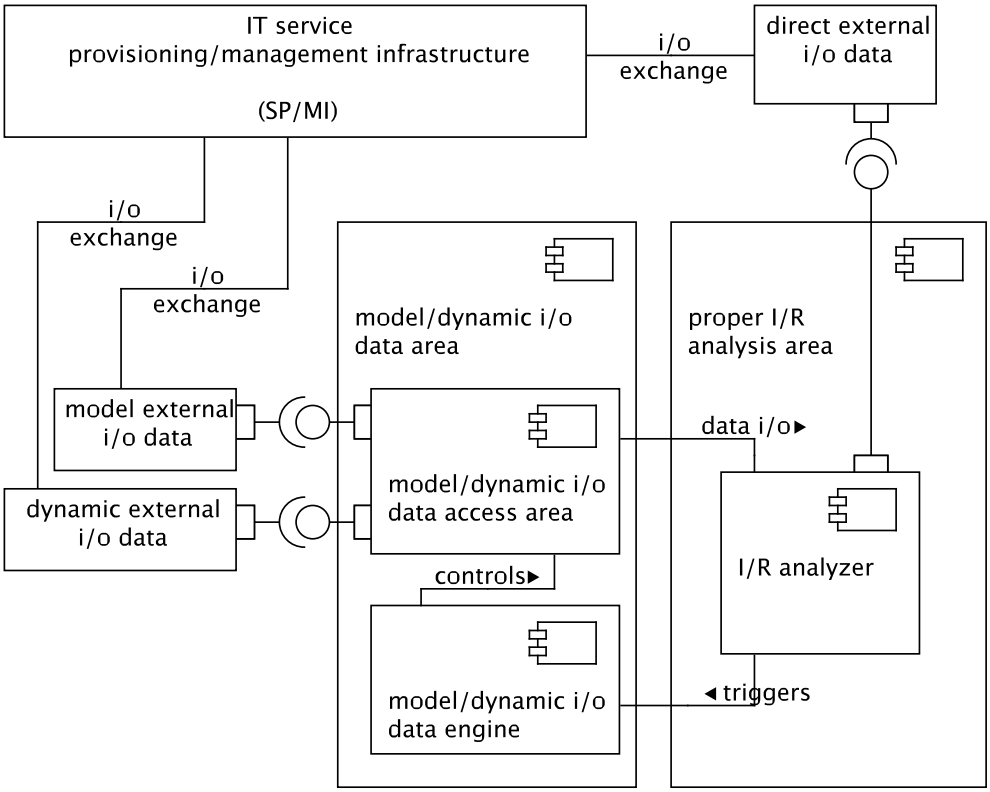

Figure 4.37: Rough overview of basic component architecture (BCArch) for the basic refined abstract workflow (BRAWf) and relationship to provider's SP/MI

model/dynamic i/o modules

An architecture for performing I/RA has to be generic, i.e., applicable to any potential IT service scenario, and so be independent of specific (management) technology and approaches concerning the SP/MI (generic requirement R0.2 in Sect. 2.4.1). That is why for realizing the interfaces for model or dynamic external artifacts a range of independent, different so-called *model/dynamic input/output modules (m/d i/o modules)* for exchanging (parts of) such artifacts with various information sources/sinks in the SP/MI are devised. In general, multiple m/d i/o modules are provided for realizing one particular of the model/dynamic external interface identified in the last section, e.g., the model external interface of SI dependency proper model data. All m/d i/o modules altogether constitute the m/d i/o data access area as a part of the m/d i/o data area.

Potential examples of (management) technologies supported and covered by different m/d i/o modules include SNMP, CORBA (Common Object Request Broker Architecture), evaluation of log files and configuration files, protocol analysis, access to specific management tools like trouble ticket systems, SLA databases or even more specifically HP OpenView Software.

m/d i/o data engine for management

The m/d i/o modules in the m/d i/o data access area are directly concerned with the particular exchange of model/dynamic external artifacts with various information sources/sinks in the SP/MI. In contrast, their overall generic control and management is handled by the m/d i/o data engine, representing the

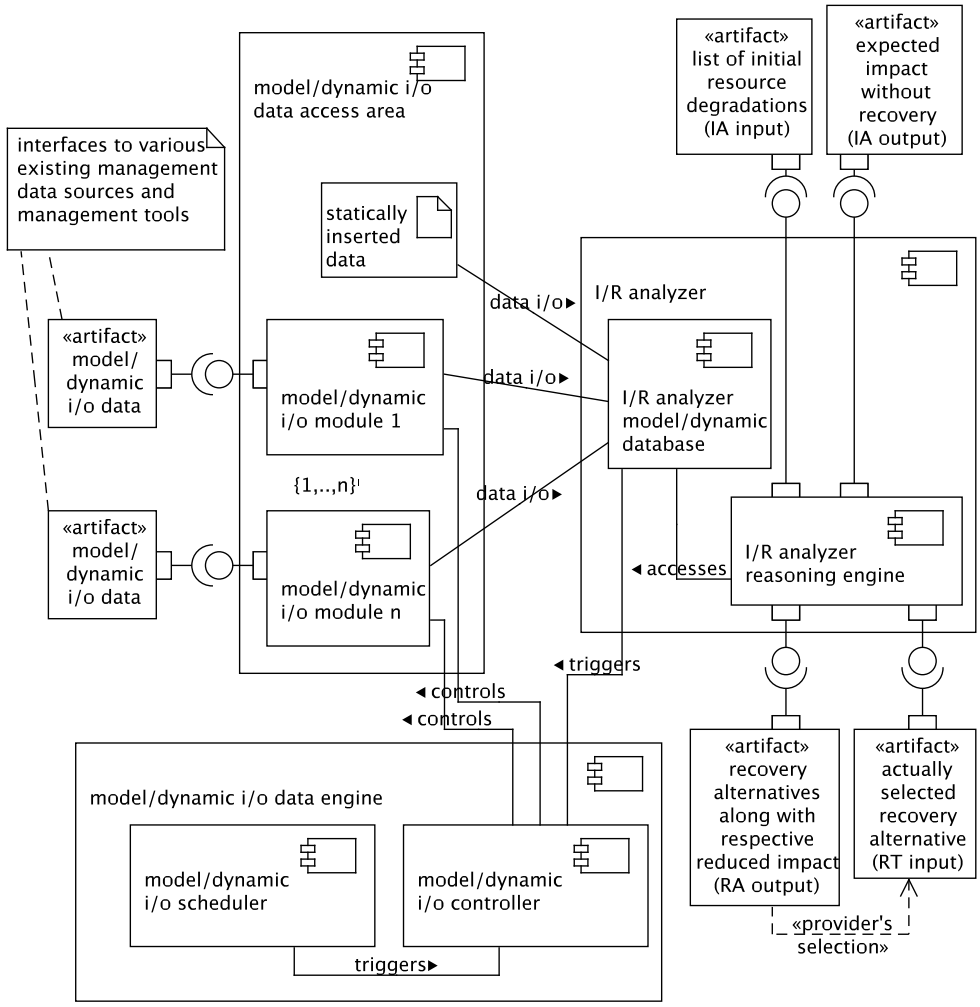

Figure 4.38: Basic component architecture (BCArch) with detailed basic internal components (BIntComps)

second part of m/d i/o data area in addition to the m/d i/o data access area. Basically, m/d i/o data engine ensures and allows for a synchronization of all internal models with the changing data in the SP/MI (generic requirement R0.3 in Sect. 2.4.1).

m/d i/o scheduler and controller

Internally the m/d i/o data engine it is further subdivided into a model/dynamic i/o scheduler responsible for scheduling of regular periodical model synchronization with the SP/MI, and a model/dynamic i/o controller responsible for coordinating regular model i/o data requests from the scheduler, and dynamic i/o requests (on-demand) from the proper I/R analysis area. So, on the one hand, the m/d i/o data engine (more exactly the m/d i/o scheduler) allows for manageability of SP/MI updates by regular synchronization with changes in the SP/MI. On the other hand, the m/d i/o data engine (more exactly the m/d i/o controller) acts as a mediator between m/d i/o access area and the proper I/RA area.

proper I/R analysis area

The *proper impact/recovery analysis area* (*proper I/RA area* or *I/R analyzer* in short) is responsible for actually performing the proper I/R analysis using

the provisioned model/dynamic external artifacts from the m/d i/o data area. It is subdivided into the *impact/recovery analysis model/dynamic database (I/RA m/d database)* and the *impact/recovery analysis reasoning engine (I/RA reasoning engine)*.

I/RA m/d database

The I/RA m/d database contains all model external artifacts and current dynamic external artifacts having been exchanged with the SP/MI by the various m/d i/o modules under control of the m/d i/o controller.

I/RA reasoning engine

The I/RA reasoning engine is responsible for performing the actual I/R analysis by utilizing the m/d database for access to the model/dynamic external artifacts. For access to direct external artifacts the I/RA reasoning engine has respective direct external interfaces which provide the direct external artifacts. As model data is updated regularly by the m/d i/o scheduler (see above) it can be accessed with low effort by the I/RA reasoning engine from the I/RA m/d database. In contrast, access to dynamic external artifacts by the I/RA reasoning engine, also via the I/RA m/d database, may cause the I/RA m/d database to trigger the m/d i/o controller for actually fetching the dynamic external data from the SP/MI via the respective m/d i/o modules.

summary of basic components

To sum it up, basic internal components (BIntComps) are the following:

- model/dynamic i/o data area: exchange of model/dynamic external artifacts with SP/MI.
  - model/dynamic i/o data access area:
    - various model/dynamic i/o modules: each specifically designed for some existing model/dynamic data source/sink in the SP/MI, possibly by utilizing specific (management) technologies.
  - model/dynamic i/o data engine:
    - model/dynamic i/o controller: actually controls the m/d i/o modules for exchanging their specific model/dynamic external data between SP/MI and I/RA model/dynamic database.
    - model/dynamic i/o scheduler: for triggering the model/dynamic i/o controller to update/synchronize particular model data with SP/MI periodically.
- proper I/R analysis area (I/R analyzer):
  - I/RA model/dynamic database: containing all model artifacts, and all dynamic external artifacts exchanged so far.
  - I/RA reasoning engine: for actually performing I/RA, i.e., actually realizing basic refined abstract workflow (BRAWf) by using model/-dynamic external artifacts in the I/RA model/dynamic database.

basic realized workflow

The above introduced design of the basic component architecture has the following consequence for the realization workflow of the basic refined abstract workflow (BRAWf), namely the basic realized workflow (BRWf): As the whole architecture is divided into two basic parts, namely m/d i/o data

area and proper I/RA area, the basic realized workflow will also consist of two respective basic parts: a *basic model/dynamic input/output support workflow (basic m/d i/o support workflow)* for managing the m/d i/o data exchange with the SP/MI, and a *basic proper impact/recover analysis workflow (basic proper I/RA workflow)*, also being called *basic I/RA reasoning workflow*. The former one is concerned with regular synchronization of model external data with SP/MI, and concerned with the intermediation of the exchange of all model/dynamic external data between the various m/d i/o modules and the I/RA m/d database. The latter is concerned actually with the proper realization of the basic refined abstract workflow by support through the former one. Thus, generally speaking, basic m/d i/o data exchange subworkflow is ensuring that all required model/dynamic external artifacts are available for the basic proper I/RA subworkflow. Moreover, basic proper I/RA subworkflow, being in essence a more concrete realization of the basic refined abstract workflow, is in turn consisting of respective subworkflows, one for each step of the basic refined abstract workflow.

In the following, further general characteristics and details of m/d i/o data engine (Sect. 4.2.5.3), the m/d i/o modules (Sect. 4.2.5.4), and the I/R analyzer (Sect. 4.2.5.5) are analyzed and treated. Based on this, eventually the basic realized workflow (BRWf) will actually be devised (Sect. 4.2.5.6) in detail.

4.2.5.3 Model/dynamic input/output data engine

The m/d i/o data engine is treated in more detail here. First some conclusions from the introduction of the m/d i/o data engine in the previous section are drawn, and afterwards some further issues based on these conclusions are introduced.

As already introduced in Sect. 4.2.5.2, the following can be said about the m/d i/o data engine: It is part of the m/d i/o data area, and in turn itself consists of two basic components, namely the m/d i/o data scheduler and m/d i/o data controller. Generally its task is to control the m/d i/o data access area, which is the other part of the m/d i/o data area. Specifically, its m/d i/o data controller has to control the performing of requests to exchange model or dynamic external data between the SP/MI and the I/RA m/d database via specific m/d i/o modules of the m/d i/o data access area. The data exchanged with the I/RA m/d database is thereby provided to the I/RA reasoning engine for actually performing the proper I/R analysis. Requests for m/d data exchange originate either directly from the I/RA m/d database or from the m/d i/o data scheduler. In the first case the request demands for on-demand exchange of dynamic external data, in the second case the requests demands the exchange of model external data for regular synchronization with SP/MI. To support the second case of request the specific task of the m/d i/o data scheduler is to regularly schedule all necessary exchanges of model external data.

data exchange triggered by I/RA m/d database or by m/d i/o data scheduler

In addition to regularly scheduled requests to update/synchronize model external data from the m/d i/o data scheduler, a specific m/d i/o data module

data exchange triggered from m/d i/o modules

may request model data update/synchronization, if it determines that important model data in its related SP/MI source has changed.

Moreover to the above mentioned types to trigger requests to exchange m/d external data with the SP/MI, two specific types have to be added: The input of model data at system startup, as well as m/d i/o triggered by a human operator for system correction purposes. These both types of exchange requests are referred to as *non-standard m/d data exchange requests*, while all former ones (occurring during normal operation of the I/R architecture) are referred to as *standard m/d data exchange requests*.

data exchange initiated at system start or by operator

In Fig. 4.39 all discussed types of triggering or requesting m/d i/o data exchange between SP/MI and IR/A model/dynamic database are illustrated. To

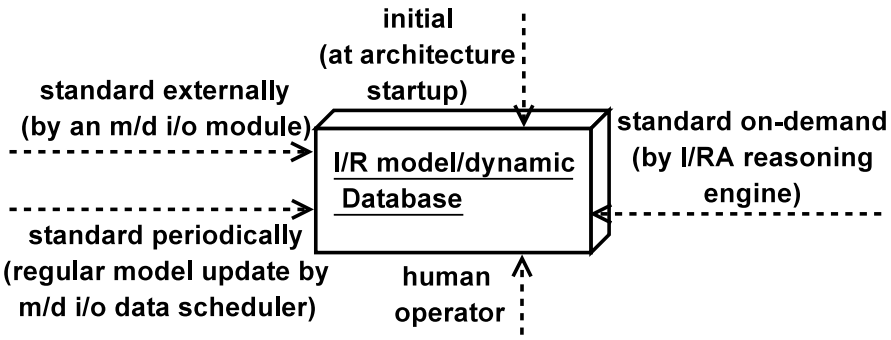

Figure 4.39: Types of triggering/requesting m/d i/o data exchange between SP/MI and I/RA m/d database

sum it up, these types of triggering/requesting are:

- statically: at system start, or later by human operator interaction (model data console).
- dynamically:
  - periodically, triggered by the model/dynamic i/o data scheduler.
  - on-demand, triggered by the I/RA reasoning engine via I/RA m/d database, when needed in its course of action.
  - externally triggered, i.e., by a model/dynamic i/o module (when a important change of model data in SP/MI is registered by the module).

4.2.5.4 Model/dynamic input/output data access area

In the following some further issues of the m/d i/o data access area, i.e., the m/d i/o modules it is consisting of, are discussed.

Each model or dynamic external artifact (compare list on p. 192) may be realized jointly by multiple m/d i/o modules, when the external artifact comprises data that is distributed over multiple information sources/sinks in the SP/MI, e.g., with respect to management technology or management paradigm.

ssifications
m/d i/o
odules

That is why m/d i/o modules can be classified in the following ways: First, m/d i/o modules can be classified by the external artifact(s) they (partially) realize (see list referenced above). So, this type of classification originates from the I/RA workflow/artifact definition, and is therefore referred to as *I/RA (artifact) oriented module classification*. Second, m/d i/o modules can be classified by external aspects and factors of its related data sources/sinks in the SP/MI, which were already in place before, independently of I/RA workflow/artifact definition. Such external, already existent aspects and factors of SP/MI data sources/sinks comprise e.g., the type of technology, management technology, management paradigm, or management architecture model. This second type of classification is referred to as *SP/MI (source/sink) oriented module classification*. Concerning SP/MI source/sink oriented module classification, there may be m/d i/o modules covering a wide range of different types of SP/MI sources/sinks, as e.g., CORBA, SNMP, simple log-files, configuration files, Excel files, SQL databases, SLA repositories, financial tools/databases, or even vendor-specific ones like Tivoli or HP OpenView.

4.2.5.5 I/R analyzer

In the following the inner structure of the I/R analyzer is treated in more detail.

basic structure

In Sect. 4.2.5.2 the basic structure and corresponding tasks of the I/R analyzer have already been introduced: The I/R analyzer basically consists of two subcomponents, namely the I/RA reasoning engine and the I/RA model/dynamic database. The I/RA model/dynamic database provides a unified access interface to all required model/dynamic artifacts (compare Fig. 4.36 on p. 191) necessary for I/R analysis. Using the unified access interface for m/d artifacts provided by the I/RA m/d database, the I/RA reasoning engine actually performs the proper I/R analysis. This way, the I/RA reasoning engine is highly decoupled from the various external m/d i/o data sources/sinks in the SP/MI, and allows for an I/R analysis to be performed in any potential given IT service scenario, e.g., independently of specific technologies. Moreover, the differentiation of model and dynamic external artifacts avoids the use of too frequent, costly exchanges of artifact data with the SP/MI (compare discussion in Sect. 4.2.5.2).

structure of I/RA reasoning engine

In the following mainly the inner structure of the I/RA reasoning engine is refined. Fig. 4.40 illustrates this refinement in a generic manner, which is later on concretized in Fig. 4.41.

generic refinement of reasoning engine

As the I/RA reasoning engine is the actual component, which is concerned with the proper I/R analysis - using a unified access to m/d artifacts, provided by the cooperation of the I/RA m/d database and the m/d i/o data area - it is faced with a range of different activities. These activities correspond directly to the various steps of the basic refined abstract workflow (compare Fig. 4.35 on p. 189) as well as its refined version, the basic realized workflow to be designed in Sect. 4.2.5.6.

That is why the structure of the I/RA reasoning engine is refined in the fol-

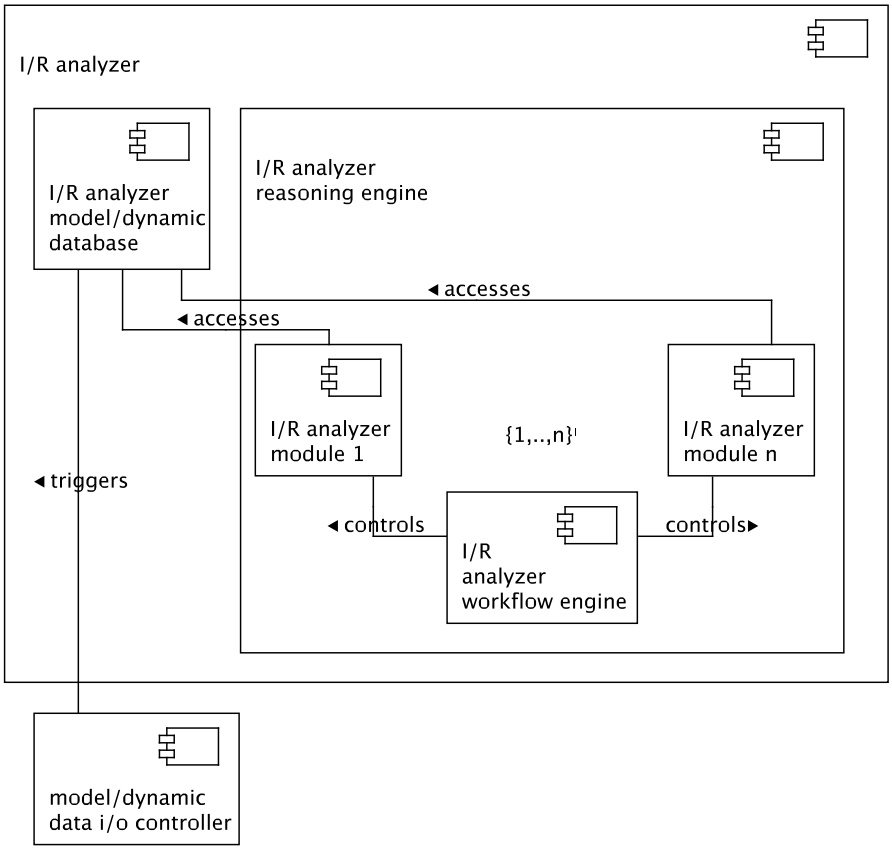

Figure 4.40: Subcomponents of the I/RA reasoning engine with generic modules

lowing way: Internally, the I/RA reasoning engine consists of a generic workflow engine (*I/RA workflow engine*), performing the coordinating all necessary steps of the proper I/R analysis, as well as a range of various workflow activity modules (*I/RA workflow modules* or tersely *I/RA modules*), one for each type of step necessary for actually performing I/R analysis. Fig. 4.40 illustrates this refinement of the I/RA reasoning engine in a generic manner, i.e., a list $1, \ldots, n$ of generic I/RA modules.

specific refinement of reasoning engine

Looking at the various steps of the basic refined abstract workflow (BRAWf, Fig. 4.35 on p. 189), the specific I/RA modules necessary are derived. So, this results in the following list of specific I/RA modules, which are corresponding to the subworkflows of BRAWf:

- *impact analyzer*: for impact analysis; actually has to two specific refinements: one for SIA (BRAWf.1.1) and one for BIA (BRAWf.1.2), i.e., resulting in a *service impact analyzer* and a *business impact analyzer*.
- *impact rater*: for impact rating (BRAWf.2.1).
- *recovery plan designer*: for recovery plan design (BRAWf.2.2).
- *recovery changes tracker*: for recovery changes tracking (BRAWf.3.1).
- *model adaption component*: for model adaption (BRAWf.3.2), will work in cooperation with m/d i/o data engine.

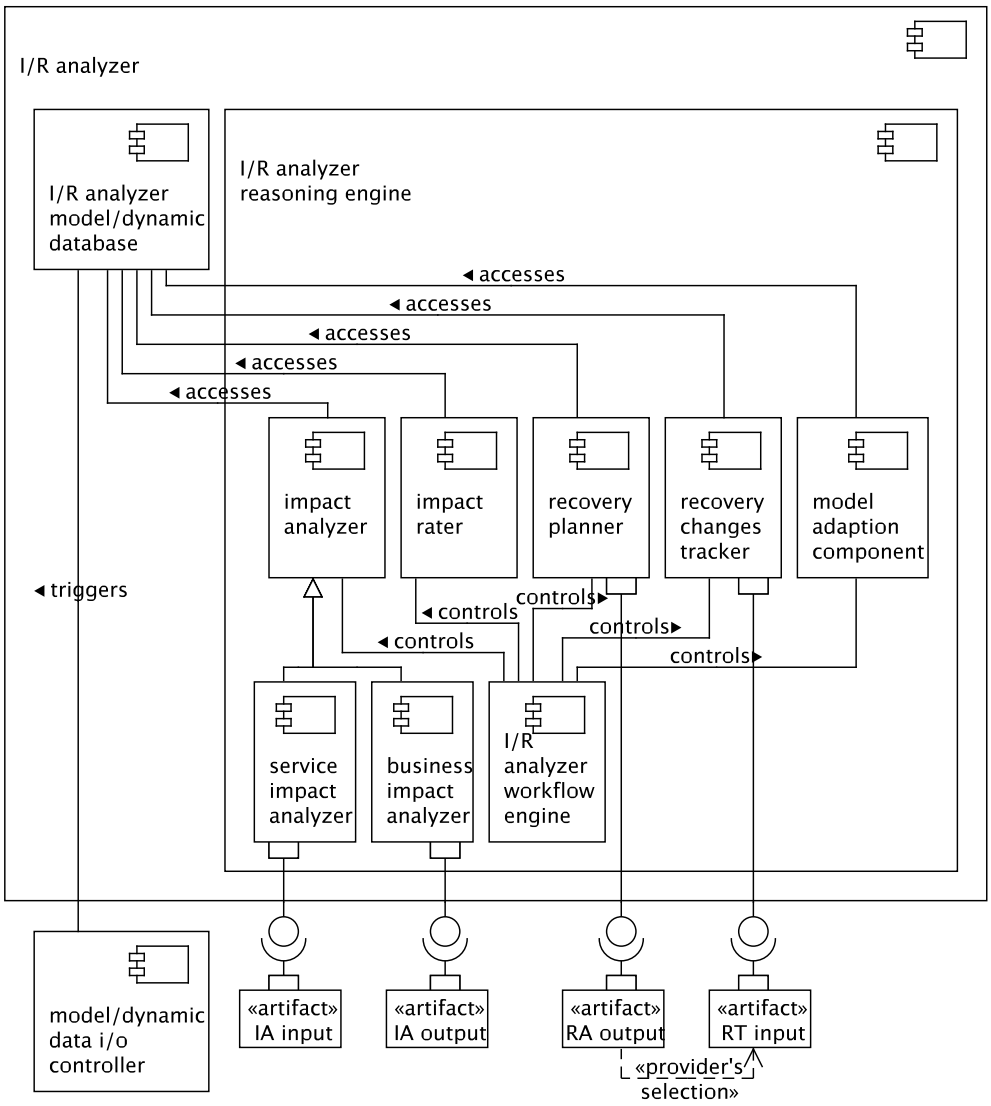

Figure 4.41: Subcomponents of the I/RA reasoning engine with specific modules (concretization of Fig. 4.40)

role of SIA and BIA

For the most part, each single step of BRAWf requires a separate I/RA module as the tasks to perform differ among each other. This is different for SIA and BIA: Both do very similar things, namely the derivation of entailed degradations, only different in the type of degradations involved, i.e., resource and service degradations in the case of SIA, and service and business degradations in the case of BIA. That is why for both - by using inheritance - as a basis a generic impact analyzer can be used as I/RA module. This generic impact analyzer is providing the functionality and the course of action for IA in general. It has two specific refinements, one for SIA and one for BIA, which actually handle the specifically involved types of degradations.

Taking into account all specific I/RA modules identified, the generic illustration of the refinement of the I/R analyzer in Fig. 4.40 can be made more concrete. The resulting concretization is illustrated in Fig. 4.41.

4.2.5.6 Basic realized workflow

Now the basic realized workflow (BRWf) is introduced as a refinement of the basic refined abstract workflow (BRAWf) and in order to actually allow to realize BRAWf by the basic component architecture introduced previously. Fig. 4.42 presents an overview of the BRWf, being discussed in the following.

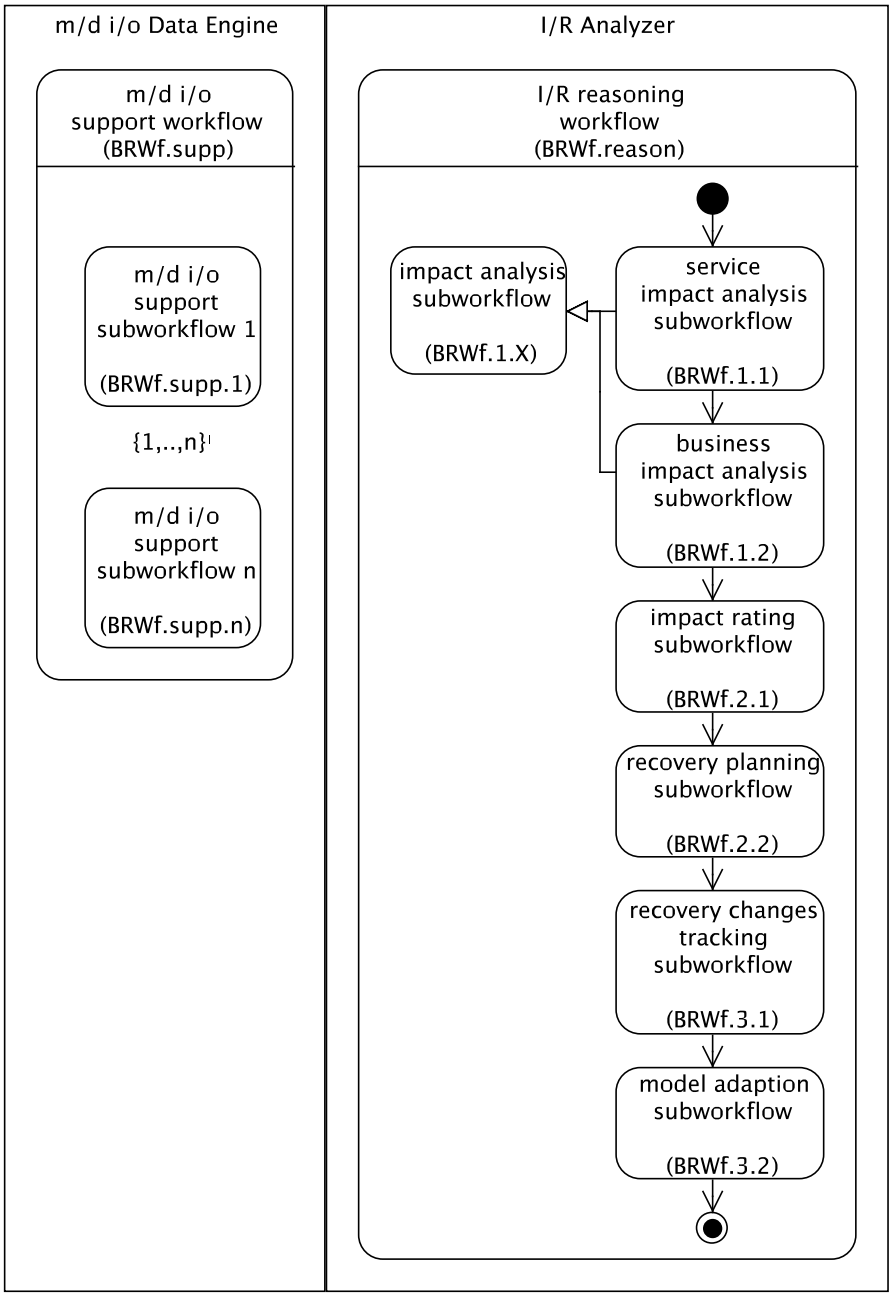

Figure 4.42: Overview of the basic realized workflow (BRWF)

subworkflows of BRWf

Similar to the basic refined abstract workflow (BRAWf), the basic refined workflow (BRWf) consists of various subworkflows (compare prior discussion on p. 196). These particular subworkflows can be grouped into two types

of subworkflows: One type of subworkflows is concerned with the management/control of the exchange of model/dynamic i/o information with SP/MI performed by the m/d i/o data engine (*BRWf m/d i/o support subworkflow*), and another type of subworkflows is involved in the actual reasoning and analysis performed by the I/R analysis area (*BRWf reasoning subworkflow*). The former type of subworkflows actually only provides support services (m/d i/o artifact exchange management) for the latter type of subworkflows.

relationship of reasoning subworkflows to BRAWf

The subworkflows of the latter type represent the proper refinement of the steps of the basic refined abstract workflow (BRAWf). That is, for each (sub)workflow step of BRAWf (compare Fig. 4.35 on p. 189), there is one subworkflow of the latter type, i.e., a BRWf reasoning subworkflow. The numbering scheme (BRWf.x.y) used for these BRWf reasoning subworkflows (as also used in Fig. 4.42) corresponds exactly with the numbering scheme (BRAWf.x.y) of the respective steps of the basic refined abstract workflow.

relationship of reasoning subworkflows to architecture

In correspondence with the design of the I/R analyzer's internal structure (compare Sect. 4.2.5.5 and specifically Fig. 4.41), each of these BRWf reasoning subworkflows is executed by one of the internal modules of the I/R analyzer. I.e., BRWf SIA subworkflow (BRWf.1.1) is executed by the service impact analyzer, BRWf BIA subworkflow (BRWf.1.2) by the business impact analyzer, BRWf impact rating subworkflow (BRWf.2.1) by the impact rater, and so forth.

Actually, the subsequent execution of the BRWf reasoning subworkflows, which corresponds to the subsequent execution of the respective workflow steps of basic refined abstract workflow, is coordinated by the I/R analyzer workflow engine, also introduced in Sect. 4.2.5.5. So, for convenience reasons, this coordinated, subsequent execution of the BRWf reasoning subworkflows is subsumed under a so-called *BRWf reasoning workflow*. Concluding, the BRWf reasoning workflow coordinates the subsequent execution of the various BRWf reasoning subworkflows performed by the respective executing I/R analyzer subcomponents under coordination of the I/R analyzer workflow engine.

Similarly, the m/d i/o support subworkflows (see above), as being the other of BRWf subworkflows in addition to BRWf reasoning subworkflows, but being executed by the m/d i/o data engine, not the I/R analysis area, are subsumed under a so-called *m/d i/o support workflow*.

BRWf = m/d i/o support + reasoning

Using both terms just introduced, the BRWf can be generally divided into BRWf m/d i/o support workflow and BRWf reasoning workflow, both comprising various subworkflows of respective type. In the following, all particular BRWf subworkflows are treated in detail. Specifically these are the impact analysis subworkflows (SIA and BIA, BRWf.1.1 and BRWf.1.2), the impact rating subworkflow (BRWf.2.1), the recovery plan design subworkflow (BRWf.2.2), the recovery changes tracking subworkflow (BRWf.3.1), the model adaption subworkflow (BRWf.3.2), as well as all additionally necessary m/d i/o support subworkflows.

IA subworkflows

Fig. 4.43 illustrates the *basic realized IA subworkflow* (BRWf.1.x), i.e., the part of the basic realized workflow performed by the impact analyzer module in the I/R analysis area.

SIA and BIA subworkflow

In fact, two instances of this workflow are performed subsequently, one for SIA and one for BIA. Only the types of degradations they are concerned with are different, i.e., resource and service degradations in the case of SIA, and service and business degradations in the case of BIA. But the general course of actions follows the same pattern for both. This inheritance relationship of BRWf IA subworkflows also corresponds to the inheritance relationship of impact analyzer components among the I/RA modules: Both specific instances of this general IA subworkflow are actually executed by the respective specific instance of the general impact analyzer module, namely the service impact analyzer or the business impact analyzer (compare Fig. 4.41 on p. 201).

entailed degradation determination

In the following, the general course of action of the general BRWf IA subworkflow, which is identical for both SIA and BIA, is discussed. The impact analyzer gets initially known degradations (resource degradations for SIA, service degradations for BIA) as essential input and has its derived degradations as essential output. All initially known degradations or ones derived in the course of action are stored in a subcomponent of the impact analyzer, the so-called *degradations working memory*. New degradations are derived from known ones by using the *degradation mappings* (see Fig. 4.11 on p. 145 and Fig. 4.13 on p. 148) defined in the specific impact dependency model (SI dependency model or BI dependency model respectively). Each degradation mapping specifies the dependency of a *target degradation* on one or multiple combined *source degradations*. Already known degradation(s) serve as source degradation(s), and by following an appropriate degradation mapping the corresponding target degradation is derived as a newly known degradation.

usage of static model data

In order to remember, degradation mappings are defined by static model data as well by additional dynamic data. So, on the one hand, degradation mappings are in fact be determined from the actual provider's service provisioning/management infrastructure (SP/MI). Moreover, the (proper) impact dependency model includes only explicitly modeled information aspects of a degradation mapping. This comprises information which is more static in nature, and it can be synchronized with the SP/MI in regular intervals without much loss of accuracy.

usage of dynamic data

On the other hand, in addition to the explicitly stored, more static information about degradation mappings in the proper model data, additional dynamic data sources/sinks in the SP/MI are referenced and known to the static model data. Similarly as for static model information, each type of dynamic information source/sink is handled by an appropriate m/d i/o module in the m/d i/o data access area (compare Fig. 4.38 on p. 195). So, the static model information of a degradation mapping is enriched on-demand by adding dynamically provided or measured information. This has the purpose of making

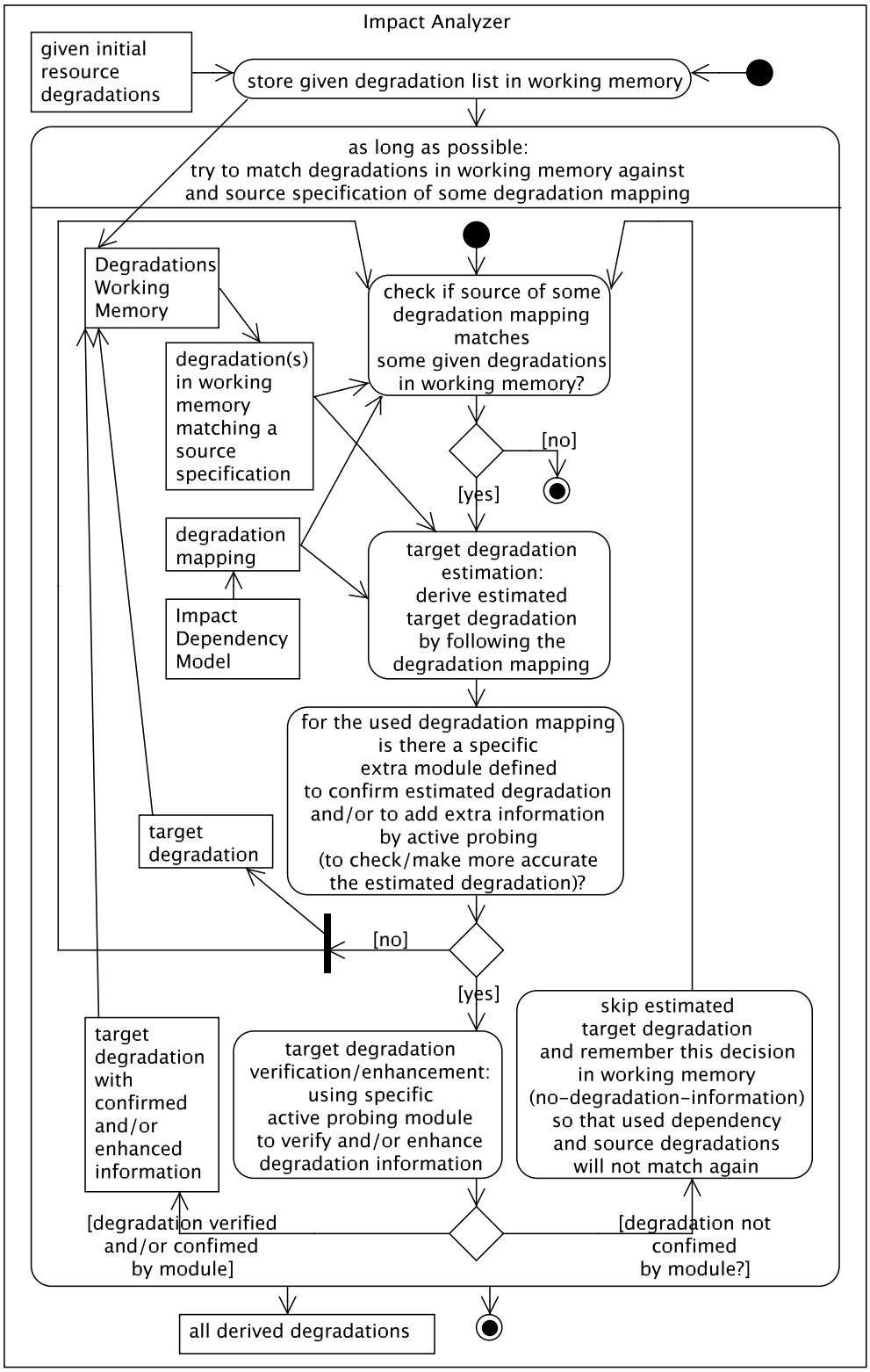

Figure 4.43: IA subworkflow (to be instantiated either for SIA or for BIA) of the basic realized workflow (BRWf.1.x, i.e., BRWf.1.1 or BRWf.1.2)

the degradation mapping information more accurate (e.g. by providing current measurement data), or of increasing its granularity, whenever necessary. Nevertheless, similar as for static model data, the impact analyzer module as part of the I/R analyzer does not interact with m/d i/o modules directly. But instead it accesses dynamic artifacts via the I/RA m/d database, which in turn cooperates with the m/d i/o data area to actually control the exchange with the SP/MI.

To sum it up, first a target degradation is derived from known sources degradations with rough, not so detailed information by using the static information of a degradation mapping in the impact dependency model. Afterwards this rough information can be refined, i.e., made more accurate or increased in granularity, by integrating once or multiple times additional dynamic data, which are eventually exchanged by m/d i/o modules on-demand with the SP/MI.

impact rating subworkflow

In the following, the *basic realized impact rating subworkflow* (BRWf.2.1), illustrated in Fig. 4.44, is treated. It is actually executed by the impact rating module of the I/R analyzer.

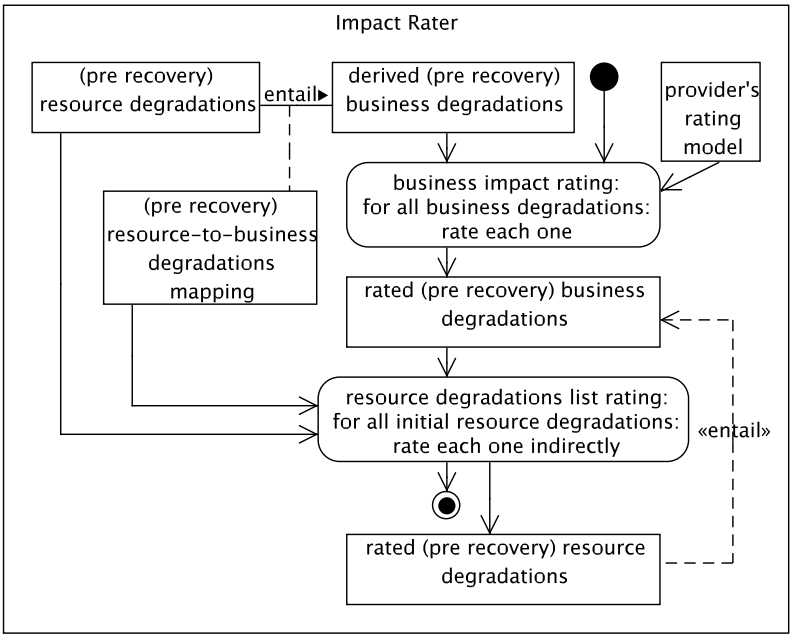

Figure 4.44: Impact rating subworkflow of the basic realized workflow (BRWf.2.1)

direct business impact rating

The impact rating subworkflow, consists basically of two steps: the (direct) business impact rating, and the following indirect resource degradations rating (compare p. 168). First, the business degradations having been derived eventually by IA are directly rated (prioritized, ordered, evaluated) using the provider's rating model. Second, an indirect rating of the initially given resource degradations (input of I/R analysis) is performed. This is actually done by starting from rated business degradations and using both impact dependency models (SI and BI), i.e., the described resource-to-service and service-

indirect rating of resource degradations

business degradation mappings, in reverse direction (in comparison to IA usage) for eventually rating/prioritizing the initial resource degradations. The resulting rating of the initial resource degradations provides a basic requirements analysis for the following recovery design subworkflow.

Fig. 4.45 visualizes the *basic realized recovery (plan) design subworkflow* (BRWf.2.2), which is actually executed by the recovery planner module of the I/R analyzer.

recovery plan design subworkflow

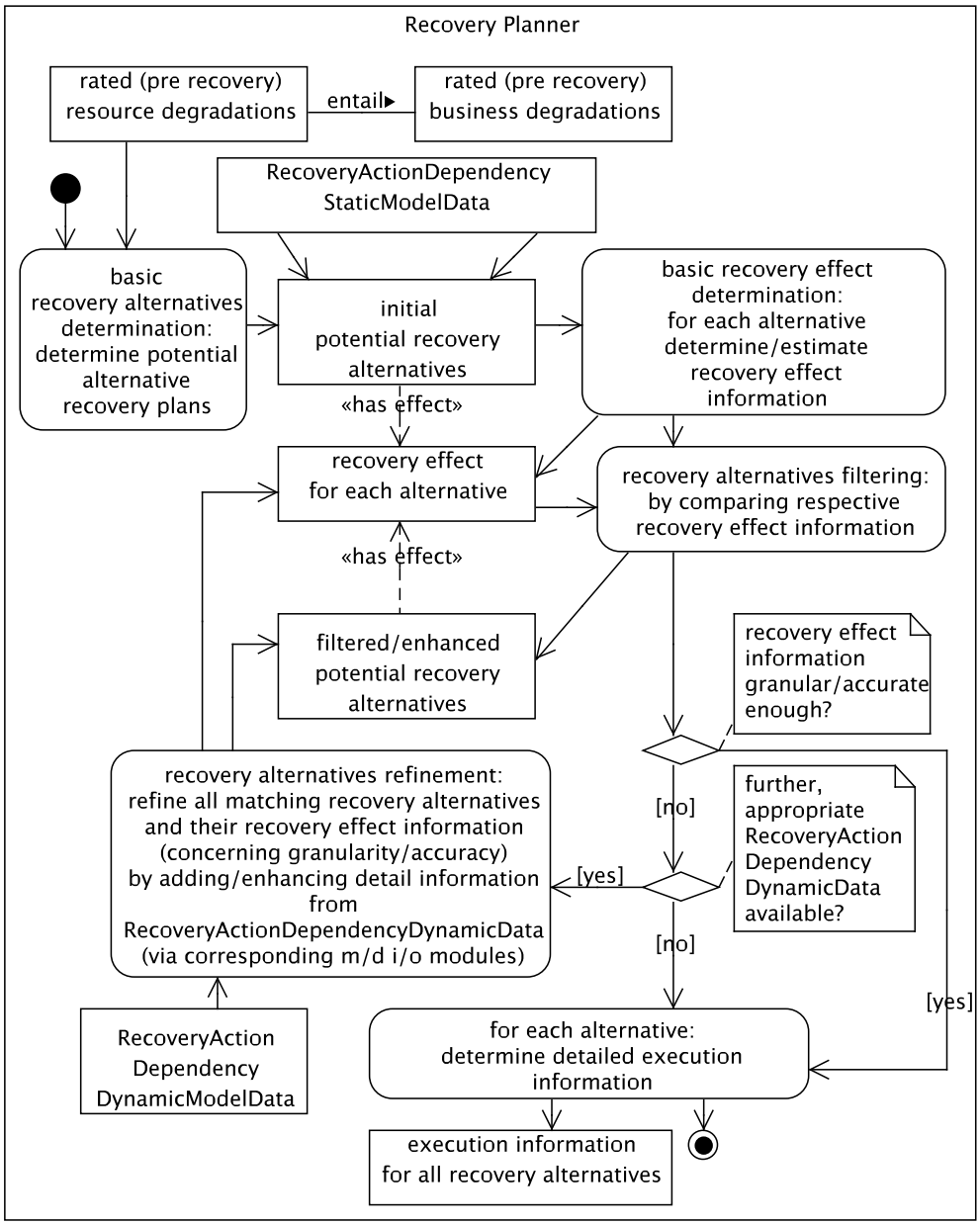

Figure 4.45: Recovery plan design subworkflow of the basic realized workflow (BRWf.2.2)

The recovery plan design subworkflow is performed in a loop of (potentially) multiple iterations for finding an optimal or an approximately optimal recovery recommendation. In each new iteration the result of the previous iteration,

multiple iterations

i.e., the determined recovery alternatives, are once more enhanced and optimized concerning their estimated effect on the business impact. In each iteration, the recovery action dependency model (possibly static and/or dynamic data of it) is used.

loop initialization

The loop initialization, i.e., the initialization of the potential recovery alternatives, is starting from rated initial resource degradations, using for the above described two steps only static model data of recovery action dependency model, resulting in first rough design of possible recovery plan alternatives. This is actually done in three steps: the potential recovery alternatives determination, the recovery alternatives effect determination, and the recovery alternatives filtering. The first step selects the initial or enhances/optimizes the existing list of potential recovery plan alternatives, each described as multiple recovery actions. The second step for each potential alternative determines or estimates its recovery effect information (influence and result information, p. 157). In the third step the current list of potential alternatives is filtered by comparing the alternatives by their actually determined/estimated recovery effect. That is why only those alternatives remain in the list which have an optimal or at least approximately good recovery effect, i.e., high mitigation of business impact over time.

loop iterations

Each loop iteration consists of the following two steps: recovery alternatives refinement by adding further additional dynamic data, and again recovery alternative filtering (same as for the loop initialization). In the first step the actual refinement and enhancement of the current list of recovery alternatives and their respective effect information is performed. This is done by adding specific dynamic additional data of the recovery action dependency model which has not been added and evaluated in previous iterations. Using this refined information, the list of potential recovery alternatives is updated accordingly, depending on all now available recovery action dependency model data in the I/RA m/d database, be it static model data or any dynamic data added so far. So, also new alternatives may be added to the list. The second step, the filtering of the current list of recovery alternatives by comparison of their actual effect information, is the same as for the loop initialization.

Further iterations are performed, as long as further additional dynamic data of the recovery actions dependency can be added, or until the effect information of the determined recovery alternatives reach a certain level of good effect over time with appropriate granularity and detail.

determination of execution information

At last, after the last performed loop iteration, for the resulting refined/optimized recovery plan alternatives, the specific execution information (see p. 167) is determined in order to make it possible to actually realize any of the particular recovery alternatives.

provider's selection of alternative

After the recovery planning subworkflow, the actual selection and the potential adaption (see p. 180) of the recovery alternative to be realized, selected and adapted by the provider, is taking place. Nevertheless, for any potential adaption made, the corresponding adaption to the execution information should be also performed. The reason for this is that actually the recovery ex-

ecution information of the selected recovery alternative to be realized is used as input for following recovery changes tracking subworkflow.

recovery changes tracking subworkflow

In Fig. 4.46 the *basic realized recovery changes tracking subworkflow* (BRWf.3.1), is illustrated. The recovery changes tracker module of the I/R analyzer is used for actually performing it.

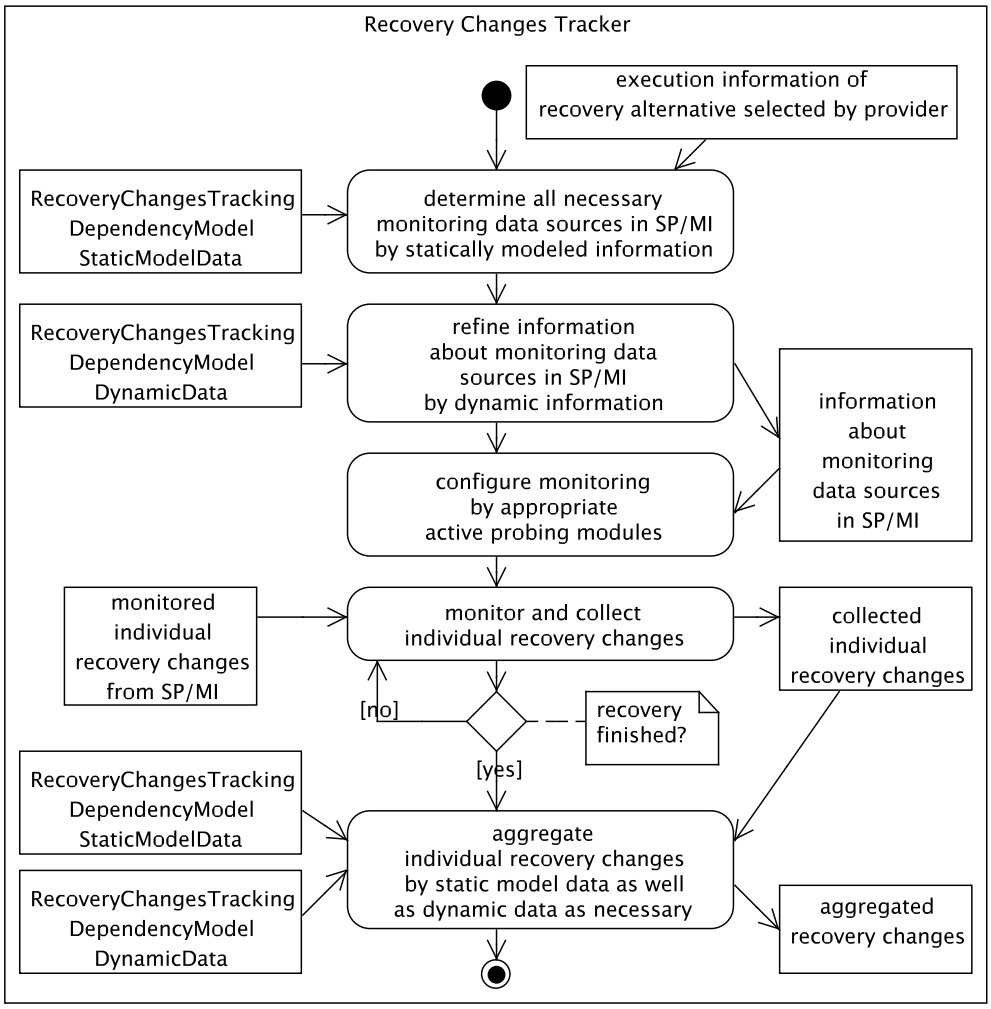

Figure 4.46: Recovery changes tracking subworkflow of the basic realized workflow (BRWf.3.1)

In general, recovery changes tracking as described previously (compare p. 185) comprises three basic steps. First, the determination and monitoring configuration of types of individual recovery changes to be monitored during recovery is performed, by using the respective mapping described by the recovery change tracking dependency model and illustrated in Fig. 4.33 on p. 185. Second, the actual monitoring and collecting of such individual recovery changes takes place. Third, all collected individual recovery changes are appropriately aggregated, by using the respective mapping also described by recovery changes tracking dependency model (see also Fig. 4.33 on p. 185).

Specifically, the first step can be further refined in the following way: First, by using the execution information of recovery alternative actually selected

recovery change monitoring configuration

by the provider, the types of recovery changes (and so indirectly the appropriate m/d i/o modules) to be monitored are determined. Moreover, this is first done roughly by using only static model data of recovery changes tracking dependency model, and secondly be refined by consecutively use of additional dynamic data of the recovery changes tracking dependency model, as far as needed. Afterwards, the actual recovery changes data source in the SP/MI have to be configured for the monitoring, via the appropriate m/d i/o modules.

model adaption subworkflow

As the last I/RA reasoning subworkflow, the *basic realized model adaption subworkflow* (BRWf.3.2), presented by Fig. 4.47, is treated. It is performed mainly by the model adaption module of the I/R analyzer, with cooperation of the m/d i/o data area, i.e., by cooperation of respective m/d i/o modules.

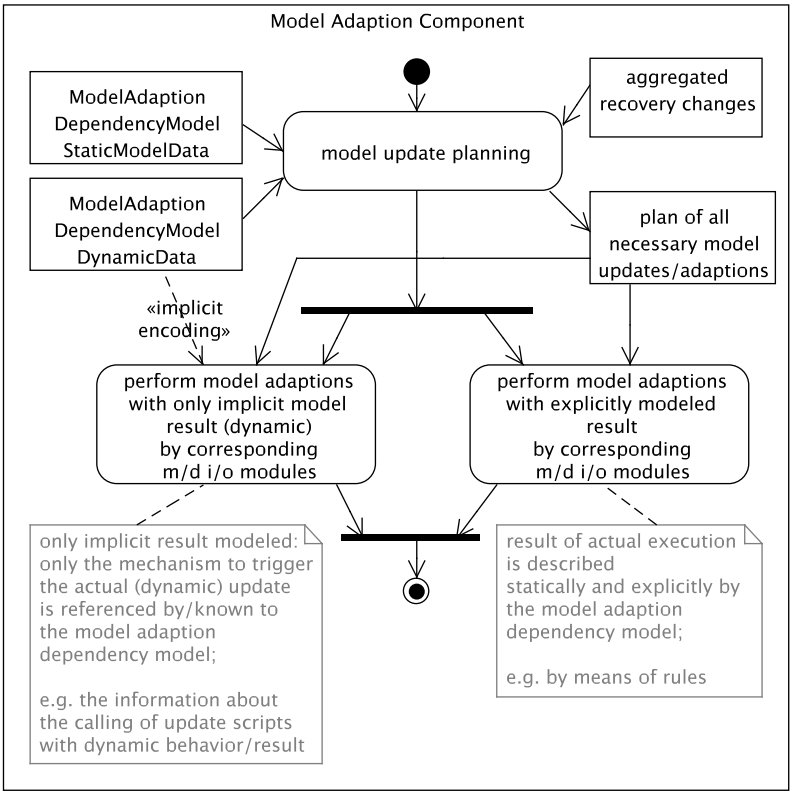

Figure 4.47: Model adaption subworkflow of the basic realized workflow (BRWf.3.2)

The model adaption can be basically subdivided into two steps: the planning of the model adaption, and the actual performing of the model adaption (realization of the planning).

model adaption planning

The planning is done by deriving from the aggregated recovery changes the actual model adaptions/updates for synchronization with SP/MI to be actually done. For this purpose so-called *model adaption planning dependencies* in the model adaption dependency model (comprising static and dynamic data) are utilized. The result is a plan for all coordinated steps of the model update/adaption/synchronization to be performed.

classification of model updates

The model adaptions may be itself classified in the various ways: On the one hand, static model data and/or dynamic data may be updated, i.e., as far as the differentiation of static/dynamic model data for I/RA workflow model artifacts to be used in further I/RA runs is concerned.

On the other hand, it matters whether the model adaption dependency model explicitly specifies the update of some model data for synchronization with recovery changes: A model adaption (of static or dynamic model data; compare above) may be described explicitly, i.e., specifying explicitly the changed values of the updated model parts, or only implicitly, i.e., only referencing dynamic behavior with dynamic results with respect to updated model data. An example of the latter case is the information about some management tool, script or similar to call, whose dynamic behavior and correspondingly whose dynamic output is not explicitly described by the model adaption dependency model. Only the necessity of calling this management tool, script or similar is described by the model. In contrast, an explicitly described model update/adaption is the explicit change of one or multiple entries in database, e.g., describing configuration information, e.g., explicit change of the IP address which is used by some service resource. This way, (actual realization of) model adaptions with only implicitly modeled result encodes implicit *model adaption execution dependencies*, which are not explicitly covered by the planning dependencies described above.

Model updates of either type introduced above are covered by the model adaption planning, and are actually performed by the respective m/d i/o modules.

m/d i/o support subworkflows

At last, after having discussed all I/RA reasoning subworkflows, the m/d i/o support subworkflows (BRWf.supp.x), which are concerned with the external data exchange with the SP/MI by the m/d i/o data area, are presented. Fig. 4.48 presents all these subworkflow at once, as they are very interrelated. Actually three different m/d i/o support subworkflows can be identified. One is performed by the m/d i/o data scheduler (BRWf.supp.1), and the two other ones are performed by the m/d i/o data controller (BRWf.supp.2 and BRWf.supp.3).

support subworkflow 1

BRWf.supp.1 is concerned with the scheduling of regular static model updates/synchronizations to be done with the SP/MI. There may be multiple instances for this subworkflow, concerning different parts of static model data, each potentially with different scheduling (e.g., different interval), as far as necessary.

m/d i/o exchange requests

The m/d i/o data controller actually manages and coordinates all m/d i/o data exchange with the SP/MI via respective m/d i/o modules (compare introduction of Sect. 4.2.5.3). Primarily, such a data exchange is initiated by the m/d i/o data scheduler for regular static model updates, or by the I/RA m/d database of the I/R analyzer for updating dynamic data. Moreover, the requests for m/d i/o data exchanges coordinated by the m/d i/o data controller include the following ones: external requests initiated by an m/d i/o module when an important model data change in the SP/MI is discovered, potential, external requests from human operators, or initial requests at startup of the

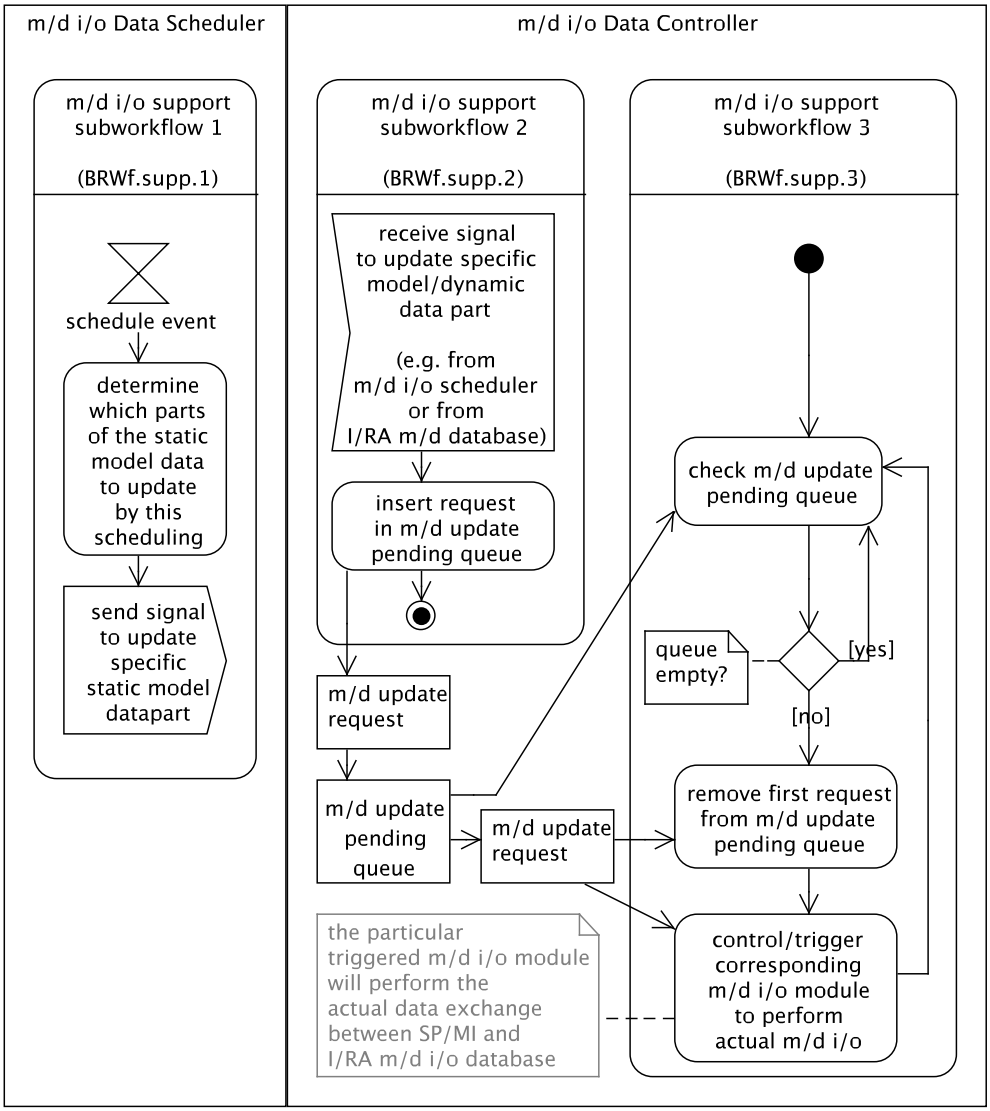

Figure 4.48: Model/dynamic i/o support subworkflows of the basic realized workflow (BRWf.supp.x)

I/RA architecture to initialize the static model data. Any such requests for exchanging m/d artifact data with the SP/MI, initiated by either source, may be designated as *m/d update request* in the following. For managing the handling of such m/d update requests, m/d i/o data controller uses an *m/d update request queue* for remembering requested, but not already completed m/d i/o updates.

support subworkflow 2 and 3

On the one hand, BRWf.supp.2 is concerned with the filling of the m/d update request queue with new m/d update requests. On the other hand, BRWf.supp.3 actually works in an endless loop to actually perform and supervise m/d update requests, including the forwarding/routing the request to the appropriate m/d i/o module.

4.3 Impact Analysis Framework

Here the *impact analysis framework* (*IAFw*) as the first of the three extension frameworks to the *basic framework* (*BFw*) is developed. In order to remind of Fig. 4.1, Fig. 4.49 depicts the basic structure of the whole I/RA framework specifically marking the impact analysis framework. The basic framework provides a generic framework, introducing basic terms, concepts and workflow steps for I/R analysis in general. The IAFw extends, refines and highly details the basic notions, concepts, and workflow steps introduced particularly for impact analysis (compare Fig. 4.1 on p. 122). More specifically, the IAFw is concerned with the design and realization of actual data structures and their specific usage in the respective workflow steps.

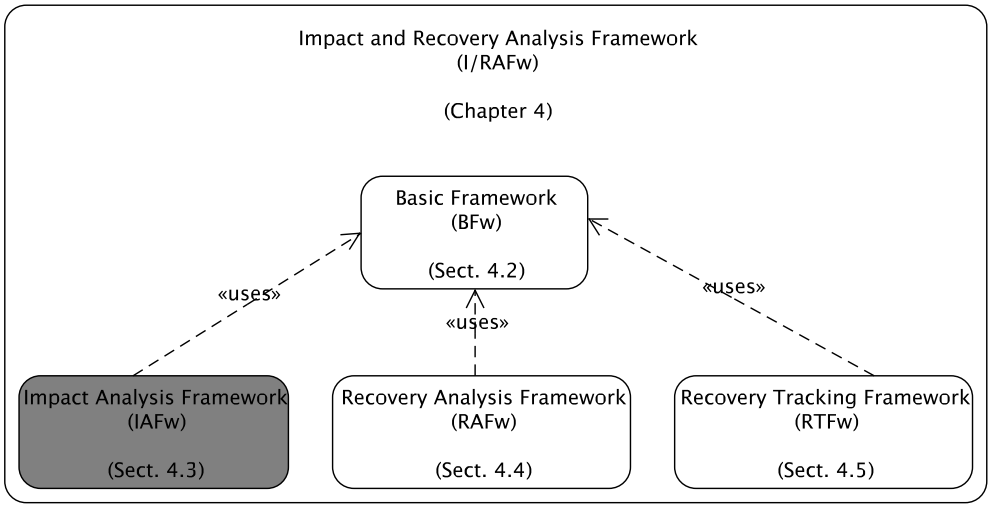

Figure 4.49: Impact analysis framework (IAFw) as first extension framework of the basic framework (reminder of Fig. 4.1)

IAFw = SIAFw + BIAFw

In general, as basically introduced in Sect. 4.2.2.1 on p. 138, impact analysis (IA) is subdivided into service impact analysis (*SIA*) and business impact analysis (*BIA*). Consequently, the IAFw can be correspondingly subdivided into a *service impact analysis framework (SIAFw)* and a *business impact analysis framework (BIAFw)*.

IA dependency models

For IA in general *impact dependency models (IDepMods)*, and specifically for SIA and for BIA *service impact dependency models (SIDepMods)* and *business impact dependency models (BIDepMods)* respectively have been introduced in the basic framework (compare Sect. 4.2.2.2 on p. 140 and p. 144).

basic IA subworkflows

The usage of these impact dependency models basically has been covered in Sect. 4.2.2.2 (basic refined abstract impact analysis subworkflow, BRAWf.1) as well as in Sect. 4.2.5.6 on p. 203 (basic realized impact analysis subworkflows, BRWf.1.1 and BRWf.1.2).

Impact dependency models describe dependencies between various types of degradations. The general information parts which are necessary to describe these degradations were generically introduced in Sect. 4.2.2.1 on p. 134. This

introduction stayed on a very rough and conceptual level, and did not include specific data structures for actually specifying particular degradations. Similarly, the actual data structures for impact dependency models were not covered. Therefore, the IAFw is mainly concerned with the concretization - in terms of data structures and their actual usage - of both impact dependency models, i.e., the SIDepMod and BIDepMod. This also includes - as far as necessary - the concretization of data structures for the various types of degradations involved.

concretization of impact dependency models

The concretization of data structures for IDepMods is actually performed iteratively in various refinement steps, mainly for SIA. Different notions of IDepMod pertaining to different refinement steps will be suitable and appropriate for different types of service scenarios, concerning the level of granularity and accuracy as far as necessary for the determination of business impact in a scenario.

iterative concretization of IDepMods

Fig. 4.50 gives an overview of the approach for the development of the impact analysis framework (IAFwAppr). Initially, general characteristics and

approach for IAFw development

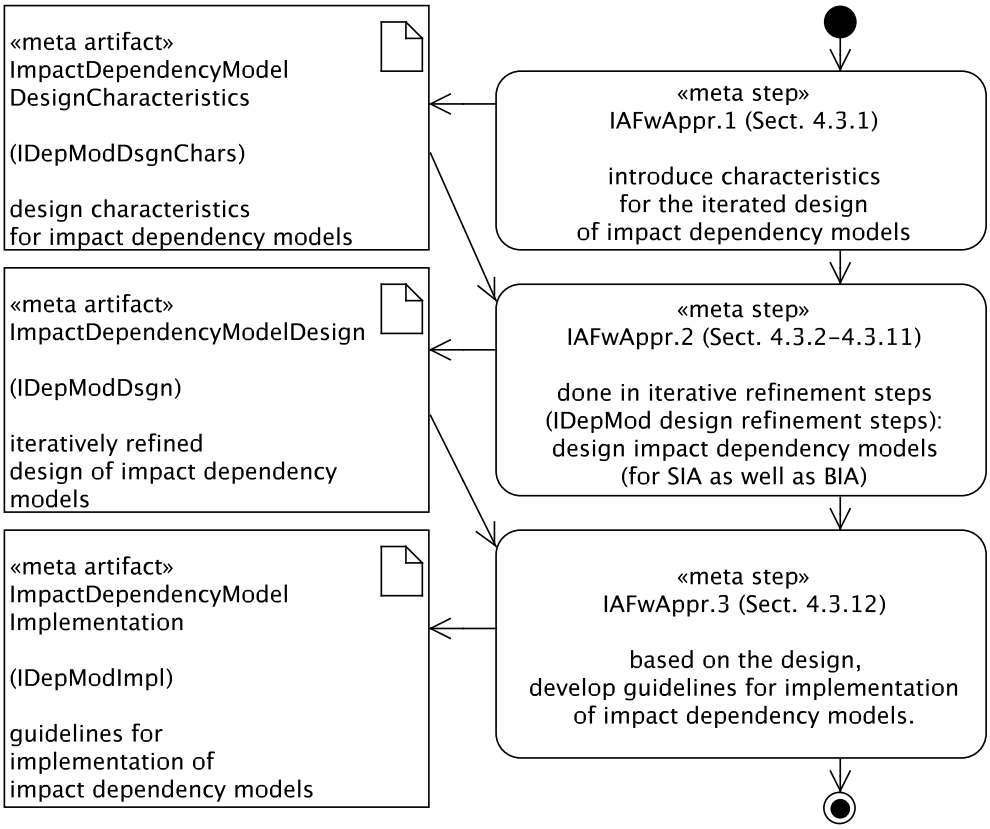

Figure 4.50: Approach (IAFwAppr) for the development of the impact analysis framework (IAFw)

aspects for a specific design of impact dependency models are introduced. Based on this, in various iteration/refinement steps, the design of BI and SI dependency models is actually done: First, a generic SI dependency model (SIDepMod(Gen)) and then a generic BI dependency model (BIDepMod(Gen)) is introduced. Afterwards, multiple iteration/refinement steps will

follow to refine these generic impact dependency models - mainly the SIDepMod. After all these iteration steps for the design of impact dependency models, guidelines for an implementation of the resulting refinement hierarchy of impact dependency models are introduced. These guidelines cover also the usage of the implemented impact dependency models as part of the basic component architecture introduced for the basic realized workflow BRWf (see p. 203 for BRWf.1.1 and BRWf.1.2).

The specific refinement steps for the design of the SI dependency model are the following:

- SIAFw(Gen): abstract base for service impact dependency models.
- SIAFw(Subj:Sv): considering only resources and services as a whole and degradation dependencies between them.
- SIAFw(Subj:SvInst): considering only resources and service instances and corresponding degradation dependencies between them.
- SIAFw(Subj:Fcty): decomposing service functionality: considering service functionalities organized as an inheritance hierarchy, and corresponding degradation dependencies between them.
- SIAFw(Subj:FctyInst): further decomposing functionalities: considering instantiations and sets of instantiations of functionality classes, and corresponding degradation dependency instantiations (links) between them.
- SIAFw(QoX)/SIAFw(QoXInst): considering gradable degradations, i.e., how (which specific QoR/QoS parameters) and to what degree (metric of QoR/QoS parameter) is a service, a functionality class, functionality instantiation degraded, and corresponding refined degradation dependencies.
- SIAFw(Coop): considering degradation dependencies with a specific instantaneous cooperation pattern of the source degradations, on which a target degradation is depending jointly (at a specific instance of time: dynamics at a specific instance of time), i.e., degradation dependencies (to services/functionality classes/functionality (instantiations)) with multiplicity > 1, and with additional attributes describing the cooperation pattern between the multiple source degradations as far as necessary for service impact determination: For instance, the cooperation pattern between multiple resources/subfunctionalities on which another one depends jointly: redundancy, load-balancing, aggregation of QoR/QoS parameters (e.g., various delays adding up to a high-level delay), but also considering complex time dependencies (e.g. one degradation of a short degradation causing another one with much longer degradation), dependence on a resource/functionality which is used multiple times in one single service interaction (e.g., multiple DNS queries for sending one mail, i.e., DNS delay propagating multiple times to mail sending delay).

- SIAFw(DynOT): considering dynamics of degradation dependencies over time: dynamics of the existence (validness) of the degradation dependency at all (at different instances of time), the dynamics of its dependent degradations (sources or targets), or the dynamics of its instantaneous cooperation pattern of its source degradations (compare SIAFw(Coop) above concerning this).

The specific refinement steps for the design of the BI dependency model are only the two following:

- BIAFw(Gen): abstract base for business impact dependency models.
- BIAFw(Char): using so-called business degradation metrics (functions of time/duration) for specifying/comparing business degradations and corresponding degradation dependencies between them.

For each refinement step, an appropriate measure of time/duration has to be defined. This has to be done with respect to the least time unit in and on which an I/R analysis run which is utilizing the respective notion of IDepMod of the particular refinement step operates. This requirement is mainly necessary for the last two refinement steps which specifically deal with dynamics at a specific instance of time (SIAFw(Coop)) or dynamics over a (longer) time range (SIAFw(DynOT)).

In the following, in Sect. 4.3.1, the general characteristics for the design of impact dependency models are introduced.

4.3.1 General characteristics/aspects for design of impact dependency models

In the following general characteristics and aspects for the (iterative) design of the two impact dependency models (IDepMod), namely service impact dependency (SIDepMod) and business impact dependency model (BIDepMod) are introduced. These characteristics and aspects are used in the following sections for the actual design of the IDepMods.

IDepMods have been basically introduced in Sect. 4.2.2.2 on p. 140 and p. 144. The usage of IDepMods has been basically covered in Sect. 4.2.2.2 (basic abstract impact analysis workflow subworkflow, BAWf.1, and basic refined abstract impact analysis subworkflow, BRAWf.1) as well as in Sect. 4.2.5.6 on p. 203 (basic realized impact analysis subworkflows, BRWf.1.1 and BRWf.1.2).

degradation dependency

IDepMods comprise and describe various directed relationships between degradations, so-called *degradation dependencies* (*DegDeps*) or *degradation mappings*.

directed relationship

In general, a *directed relationship* is a relationship between two types of *dependent objects*, namely from one or multiple *source objects* (tersely sources)

to one or multiple *target objects* (tersely targets) (compare notion of directed relationship in UML, p. 80). Fig. 4.51 illustrates the information parts gen-

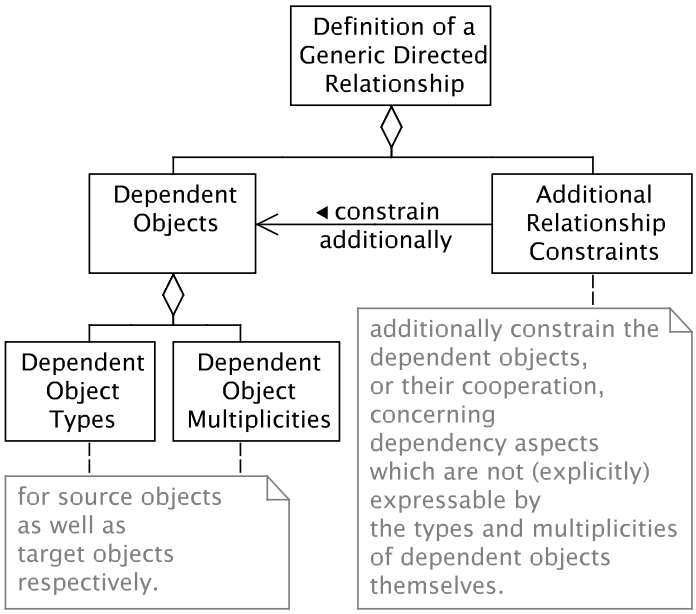

Figure 4.51: Definition of a directed relationship in general

erally necessary for the definition of a directed dependency. These parts are treated in the following using the example of degradation dependencies. For this purpose, Fig. 4.52 specifically illustrates the information parts necessary for the definition of a degradation dependency. So, Fig. 4.52 also illustrates the generic definition of an impact dependency model, as it is consisting of degradation dependency definitions.

dependent degradations

In the case of DegDeps, the dependent objects are *dependent degradations*. Consequently, a DegDep is a directed relationship from (potentially one or multiple) *source degradations* to (potentially one or multiple) *target degradations*.

degradation dependency definition

The definition of a DegDep can be subdivided into two parts: the definition of i.e., the source/target objects (source/target degradations) of the DegDep (*DegDep objects definition*), as well as any further specific constraints (*DegDep additional constraints*) of the DegDep necessary in addition to its particular source(s) and its target(s) (*DegDep additional constraint definition*). An example of the latter case are time constraints, e.g., restricting the validness of a DegDep to various time instances or time ranges. A second example are cooperation patterns (composition types) of multiple source objects, e.g., concerned with questions of how multiple source degradations of redundant resources entail corresponding service degradations. Furthermore the latter includes any further aspect which is necessary for DegDep definition but not expressed/not differentiated in the definition of dependent objects itself. Of course, an appropriate refinement of the space of potential dependent objects (dependent degradations), i.e., an appropriate refined subdivision of them, might eliminate the need for the definition of DegDep additional

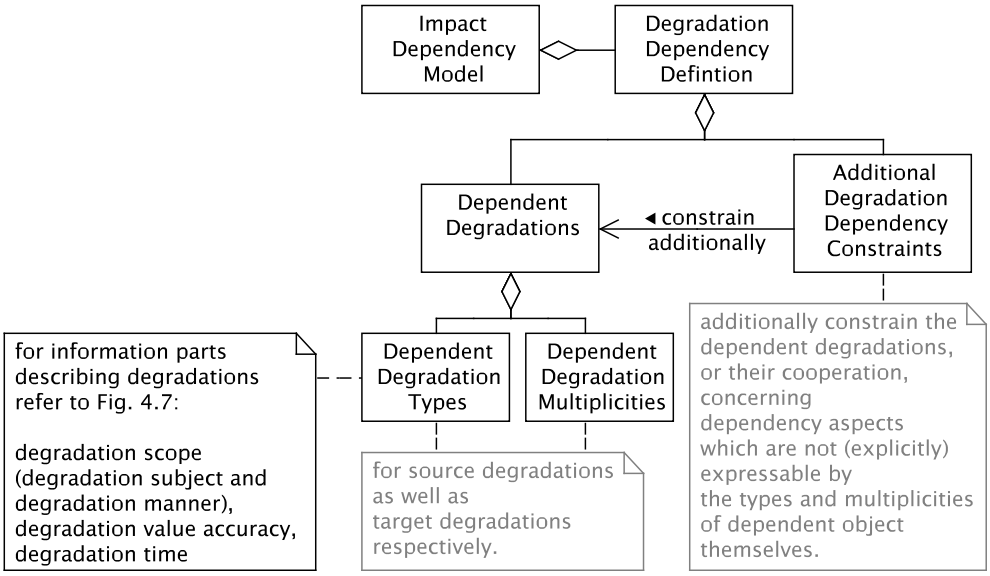

Figure 4.52: Generic definition of a impact dependency model consisting of degradation dependency definitions

constraints. Concerning the former of the two aspects, the *DegDep object definition*, the potential types of objects themselves, i.e., potential types of degradations used as sources or targets (*DegDep object types*), and also their potential multiplicities (*DegDep object multiplicities*) for a particular DegDep are relevant. The abstract information parts necessary for describing the potential types of degradations were already basically treated in Sect. 4.2.2.1 (see Fig. 4.7 on p. 135).

That is why a thorough design and implementation of IDepMods has to cover the *DegDep objects definition* and any necessary *DegDep additional constraint definition*.

naming of iterative refinement steps

The naming of the specific iteration of the SI dependency model design being used in the following is *SIDepMod(X)* where *X* identifies the specific iteration. It correlates directly to the naming convention *SIAFw(X)* labeling the design of the impact analysis framework up to this refinement step of SIA as a whole (compare list of refinement steps in Sect. 4.3 on p. 215). Analogous, the designation BIDepMod(X) is used for the design of BI dependency models in correspondence to the designation BIAFw(X).

requirements for the iterative refinement

compatibility of DegDep notions of different refinement steps

Each specific iteration/refinement step of the design of IDepMods (IDepMod design refinement step) comprises a particular refinement of the notion of DegDeps. Moreover, for compatibility and comfortability concerning the definition of particular DegDeps later-on at the instantiation to a specific service scenario, each refinement step should have the possibility to easily also express the cases of the (non-refined) steps preceding it. Stating it alternatively, for I/RA applied to a specific service scenario it should be easily possible to mix degradation dependencies defined using different IDepMod design refinement steps. So, for each degradation dependency present in a specific scenario it should be possible to use the most refined DegDep notion as nec-

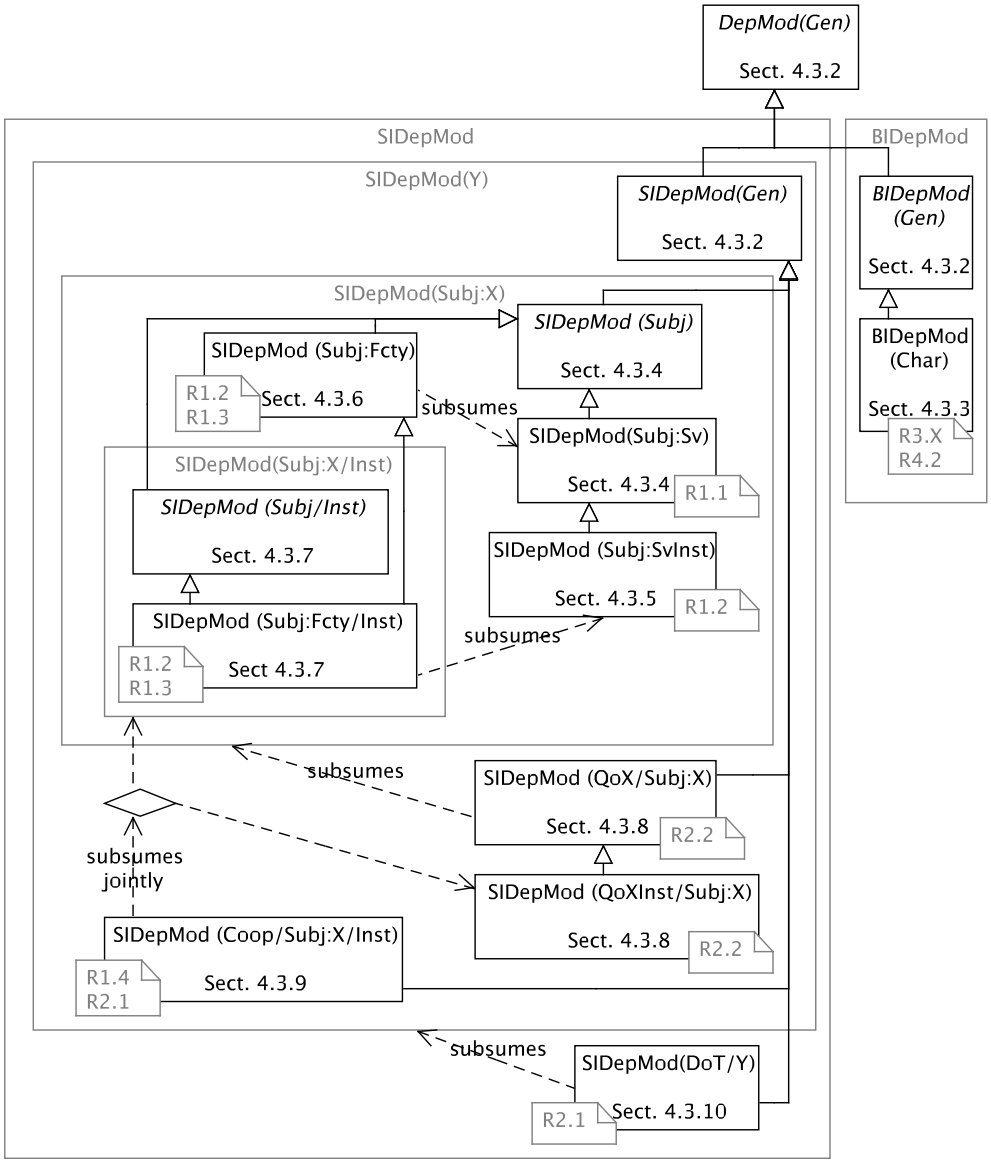

Figure 4.53: Refinement dependencies between the impact dependency models of the different steps for the iterated design (IAFwAppr.2)

essary to define it in appropriate granularity/accuracy, but on the other hand it should also be possible to use the least complex (least refined) notion of DegDep necessary independent from the choice of notion for other degradation dependencies in the scenario.

Fig. 4.53 illustrates the dependencies of this iterated design for impact dependency models (step IAFwAppr.2 of Fig. 4.50), and so gives an overview of the different impact dependency models developed in the following sections Sect. 4.3.2-4.3.10. Basically, in the figure, the naming convention as explained above, i.e., BIDepMod(X) for BIFw(X) and SIDepMod(X) for SIFw(X), are utilized. However, as the different design steps, at least for SI dependency models are based on each other, that is subsequent design steps reuse (parts of) the design of preceding steps, the concrete val-

ues for the design index variable X can be in a short form (simply X, with X $\in \{Sv, SvInst, Fcty, FctyInst, QoX, QoXInst, Coop, DynOT\}$ corresponding to the respective framework design step SIAFw(X)), or in a long, more elaborated form, mostly for referencing the reused, preceding design steps explicitly: First, SI dependency model design steps concerned only with degradation subject specification (Sect. 4.3.4-4.3.7), are in their long form designated as SIDepMod(Subj:X) (instead only SIDepMod(X)) with concrete X, marking them explicitly as related to subject specification. E.g., SIDepMod(Subj:Sv) is the long form of SIDepMod(Sv), and SIDepMod(Subj:Fcty) the one of SIDepMod(Fcty). So, all these subject design steps can be subsumed as whole by SIDepMod(Subj:X) with unbound variable X. Reuse of the specification means of SIDepMod(Subj:X) in general or with a specific X can so in following design steps be referenced in the long form of the SIDepMod(.) index designation, e.g., SIDepMod(QoX) has as elaborated, long form SIDepMod(QoX/Subj:X), to indicate the general or specific reuse of SIDepMod(Subj:X) in general or for a specific X (e.g., X=Sv, SIDepMod(Subj:Sv) or X=Fcty, SIDepMod(Subj:Fcty)) to specify degradation subjects. Reuse of preceding design steps takes place also among the SIDepMod(Subj:X) steps, namely in case of the modeled instantiation of a preceding design step (see 4.3.7). This particular specification reuse is indicated by a further elaboration of the long index designation: For instance, the full long, elaborated form of SIDepMod(FctyInst) is SIDepMod(Subj:Fcty/Inst), to mark it as the instantiated design of SIDepMod(Subj:Fcty). This notation allows to explicitly designate the reuse of subject instantiation design steps (with general or specific X), e.g., in SIDepMod(QoXInst/Subj:X/Inst) as the elaborated, long form of SIDepMod(QoXInst), which can only reuse SIDepMod(Subj:X/Inst), not the more general SIDepMod(Subj:X).

4.3.2 SIDepMod(Gen) and BIDepMod(Gen): Generic SI and BI dependency models

Here generic, abstract base classes for all following design steps of impact dependency models are introduced. That is, these classes are used as basis for further refinement in the following sections. This comprises abstract classes for the particular types of degradations as well as abstract classes for the respective degradation dependencies between the degradations (compare Fig. 4.13 on p. 148).

The abstract base classes for SI dependency model design are subsumed under the generic dependency model SIDepMod(Gen), while the abstract base classes for BI dependency model design are subsumed under the generic dependency model BIDepMod(Gen). These dependency models are actually derived from a fully generic top base dependency model IDepMod(Gen). IDepMod(Gen) consists only of one generic class representing degradation and one association class for representing the respective degradation dependencies. Fig. 4.54 illustrates the abstract top base classes for degradation

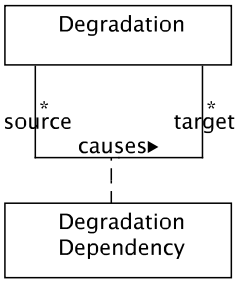

Figure 4.54: Abstract top base classes for degradations and degradation dependencies in general: IDepMod(Gen)

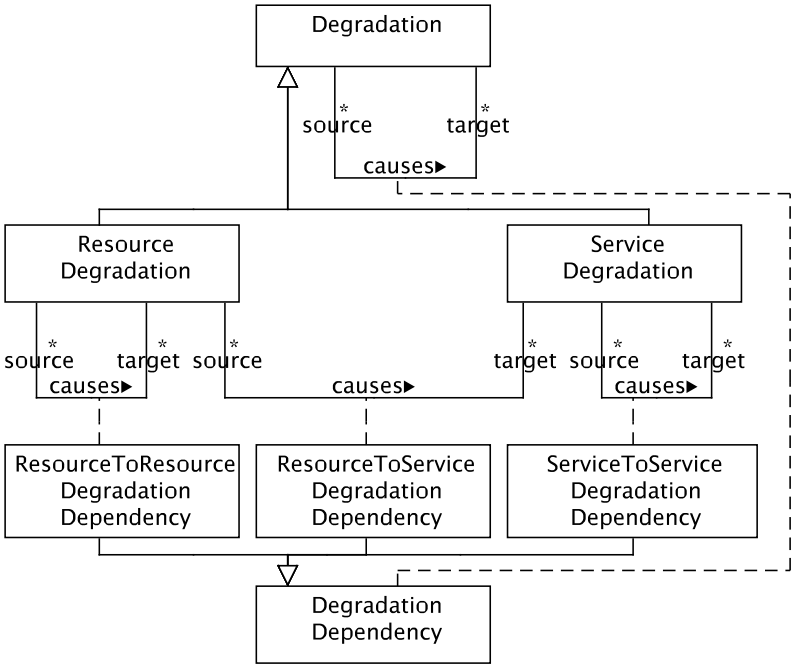

Figure 4.55: Abstract base classes for degradations and degradation dependencies related to SIA, as the constituents of SIDepMod(Gen), refinement of Fig. 4.54

and degradation dependencies as parts of IDepMod(Top). Restrictions on the multiplicity for source and target degradations are left open here, allowing multiple source and targets in general for a single degradation dependency.

Based on this, Fig. 4.55 further derives abstract base classes for degradations and degradation dependencies related to SIA. Types of degradations related to SIA are *resource degradation* and *service degradation*. Moreover, the degradation dependencies between these types of degradations - as far as related to SIA - are generally subdivided into *resource-to-resource degradation dependencies*, *resource-to-service degradation dependencies*, and *service-to-service degradation dependencies* (compare Fig. 4.13 on p. 148).

Similarly as done for SIA, Fig. 4.56 further derives abstract base classes for degradations and corresponding degradation dependencies related to BIA.

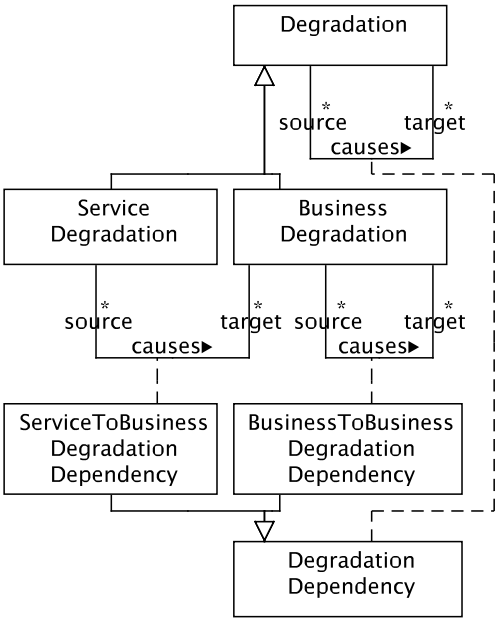

Figure 4.56: Abstract base classes for degradations and degradation dependencies related to BIA, as the constituents of BIDepMod(Gen), refinement of Fig. 4.54

The types of degradations related to BIA are *service degradations*, as well as *business degradations*. There are only two types of corresponding degradation dependencies related to BIA, namely *service-to-business degradation dependencies* and *business-to-business degradation dependencies* (again compare Fig. 4.13 on p. 148).

To provide a summary, Table 4.5 gives an overview of the degradation dependency specification by the above discussed generic dependency models, i.e., IDepMod(Gen), SIDepMod(Gen), BIDepMod(Gen),

All further refinements of the generic base classes introduced above, for SIA as well as for BIA, will be concerned with their particular specification of the specific information details of degradations, which were already identified in Sect. 4.2.2.1 (see Fig. 4.7 on p. 135).

IDepMod(Gen):	
types of dependent degradations:	described generically by an abstract base class;
	no explicit definition of specific information parts of a degradation (degradation scope = subject + manner, degradation value accuracy, degradation time);
associations of dependent degradations:	totally generic;
	multiplicities left open;
definition of additional dependency constraints:	none

SIDepMod(Gen):	
types of dependent degradations:	described generically by abstract classes for resource and service degradations;
	no explicit definition of specific information parts of a degradation;
associations of dependent degradations:	generically classified as resource-to-resource, resource-to-service, or service-to-service;
	multiplicities left open;
definition of additional dependency constraints:	none

BIDepMod(Gen):	
types of dependent degradations:	described generically by an abstract class for business degradations;
	no explicit definition of specific information parts of a degradation;
associations of dependent degradations:	generically classified as service-to-business, or business-to-business;
	multiplicities left open;
definition of additional dependency constraints:	none

Table 4.5: Overview of DegDep specification with IDepMod(Gen), SIDepMod(Gen), and BIDepMod(Gen) (compare Fig. 4.54-4.56)

4.3.3 BIDepMod(Char): BI dependency model using business degradation metrics

Here the abstract, generic BI dependency model BIDepMod(Gen), introduced in the previous section, is refined to BIDepMod(Char) in order to provide a BI dependency model which can actually be instantiated for a specific scenario.

BIDepMod(Char) is designed to be compatible with the abstract generic SI dependency model SIDepMod(Gen) also introduced previously. However BIDepMod(Char) will be actually also compatible, i.e., applicable together, with any refinement of SIDepMod(Gen) (Sect. 4.3.4 to 4.3.10).

As the general purpose of I/RA is to evaluate and rate business degradations

in order to recommend appropriate recovery alternatives, BIDepMod(Char) as a particular BI dependency model which can be instantiated for a real-world scenario, is discussed here before discussing particular SI dependency models.

different kinds of business degradations

In general business impact entailed by resource degradations may comprise a set of different business degradations of different type.

In order to remind - each of the business degradations is concerned with a specific financial or reputational factor influencing the business. Different business degradations may be caused by the same or different service degradations, which in turn may be caused by the same or different resource degradations.

In any case SLA penalty costs are considered as a standard example for business degradations (requirement R3.1), as these penalties are already defined and described in a formal and measurable manner. Moreover, it should be also possible to integrate any further resulting financial/reputational impact on the business entailed by resource degradations as some kind of business degradation (requirement R3.3). The only restriction concerning R3.3 is that these further business degradations and their dependencies from resource/service degradations are formalizable and measurable in a similar way as SLA penalty costs. As they mostly represent financial impact, they may be measured in money or any other appropriate unit. Often they may only represent some kind of estimation, e.g., for revenue loss resulting from an interruption of a dynamically subscribed service. For such kind of estimations additional information, such as the current and estimated future service usage (e.g., of a dynamically subscribed service), has to be used (requirement R3.2), if necessary.

BIDepMod(Char) provides a generic way to specify any type of business degradation, in a unique and consistent formalization. But actually it is the task of the service provider to specify these formalizations, because only he has the complete knowledge about all potential business degradations and their dependence on his resources and services.

also recovery costs as business degradation

I/R analysis in general has also to take into account potential recovery costs (requirement R4.2). Based on this requirement, in Sect. 4.2.3.1 on p. 158, potential recovery costs were identified as a specific kind of business degradation (so-called Type II business degradations). Of course, they are necessary only for describing post-recovery impact, not pre-recovery impact. That is, they are only taken into account in the recovery analysis performed later on, after the (pre-recovery) impact analysis.

Concluding, a BI dependency model being usable for actual instantiation to a concrete scenario (as BIDepMod(Char)) should be able to cover SLA penalty costs, any further (measurable or estimable) business degradation types, especially potential recovery costs.

Fig. 4.57 presents the classes used for the specification of degradations and corresponding degradation dependencies by BIDepMod(Char) (compare

Fig. 4.56), which are discussed in the following. In order to comply with

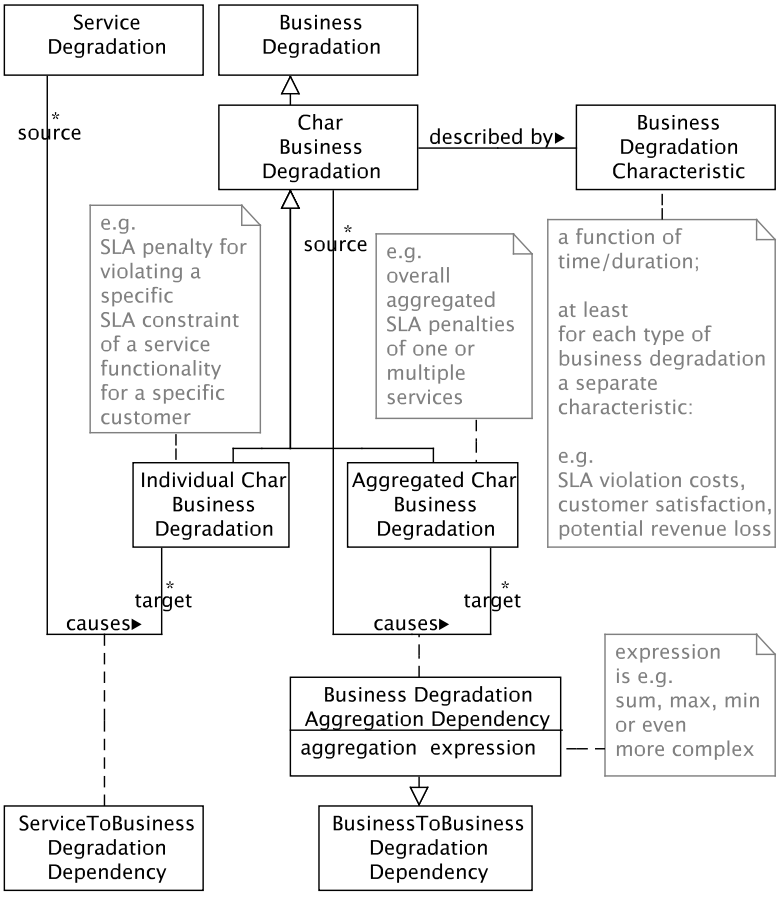

Figure 4.57: Individual and aggregated business degradations described by business degradation characteristics, as the constituents of BIDepMod(Char), refinement of Fig. 4.56

business degradation specification

the above discussed requirements, the specification of business degradations and related degradation dependencies with BIDepMod(Char) allows for the definition of all above discussed types of business degradations.

business degradation characteristics

Each type of business degradation defined with BIDepMod(Char) is particularly described by a so-called *business degradation characteristic (BDgr-Char)* or *business degradation metric*. A business degradation characteristic is an appropriately declared and defined function of time or duration, which particularly specifies the business degradation over time in a formal manner. A business degradation characteristic comprises an appropriate function signature (or function declaration, such as $duration \rightarrow costs$), and a corresponding function definition (composed of time/value pairs) complying to this signature.

Such a formal functional specification can be later on during recovery analysis used directly for rating/prioritizing the determined business degradation (and indirectly their originally entailing resource degradations), depending on the elapsed degradation duration. That is why this formal functional specification

by business degradation characteristics is a vital key factor for an appropriate and accurate recovery alternative recommendation.

For the standard example of business degradations, i.e., SLA penalty costs, the business degradation characteristic can normally directly be extracted from the SLA, which was agreed upon between the customer and provider.

The concept of a business degradation characteristic is similar to the concept of a Key Risk Indicator (KRI) in financial risk analysis (see Sect. 3.5.3). Potentially business degradation characteristics may also be directly derived from existing definitions of KRIs.

individual and aggregated business degradations

BIDepMod(Char) introduces two separate kinds of business degradations: so-called *individual business degradations*, which are being directly entailed from respective service degradations, as well as *aggregated business degradations*, which are a combination of some more basic business degradations (individual ones or aggregated ones).

In the case of SLA penalty costs, e.g., the individual SLA penalties (concerning specific violations of QoS parameter constraints) entailed from particular service degradations can be aggregated to the sum of all SLA penalty costs for a whole service, or even to the sum of all SLA penalties for all services.

Using these newly introduced terms in BIDepMod(Char), individual business degradations are derived by using service-to-business degradation dependencies, while aggregated business degradations are derived by using business-to-business degradations. (compare also Fig. 4.56).

degradation dependency specification

service-to-business degradation dependencies

In the following the specification of both kinds of degradation dependencies is treated: For service-to-business degradation dependencies, the dependent individual business degradations, and the dependence on the service degradations entailing them have to be specified. This is not covered completely here for all types of business degradations. But for the case of SLA violation costs, a proposal can be given here: e.g., in terms of SLA constraints and SLA penalty definitions for the respective SLA violation costs. In general, the mapping of (top-most) service degradations (i.e., described by classes, instantiations, or template instantiations as a specification of a set of instantiations) to some sort of *business degradation characteristic calculation algorithm* (or *business degradation characteristic expression*) has to be specified. More specifically, this algorithm or expression is used for calculating the particular business degradation characteristic (as a function of time/duration) of the individual business degradation entailed by previously determined (top-most) service degradations. In a simple case it may be assumed that the current state of a service degradation in question stays without change (without recovery), and the entailed business degradation described by a function of time (business degradation characteristic) can be directly specified by the business degradation characteristic expression. In a more complex case, i.e., the service degradation specification comprising itself some information about changing temporal course of its degradation values (e.g., QoR/QoS metric values, specified as a function of time), the business degradation characteristic calculation

algorithm has to accurately transform this information from the service degradation into the corresponding business degradation characteristic.

business-to-business degradation dependencies

For business-to-business degradation dependencies, the business degradations to be combined, as well as an appropriate *business degradation characteristic aggregation expression* (or *business degradation characteristic aggregation algorithm*) referencing the business degradations to be combined has to be specified, e.g., as a simple case the expression $sum\_of(.)$.

Table 4.6 summarizes the details of the DegDep specification with BIDepMod(Char). Moreover, Table 6.13 on p. 380 in Appendix A introduces a set-theoretic, formal notation for degradations and degradation dependencies of BIDepMod(Char).

BIDepMod(Char):	
types of dependent degradations:	business degradations described by specific business degradation characteristics (as functions of time/duration);
associations of dependent degradations:	firstly, generic specification of dependencies from service degradations to individual business degradations, i.e., to their specific business degradation characteristics (multiplicities left open);
	secondly, aggregation dependencies among business degradations;
definition of additional dependency constraints:	aggregation expression/algorithm in the case of business-to-business degradation dependencies;

Table 4.6: Overview of DegDep specification for BIDepMod(Char) (compare Fig. 4.57)

example

In the example situation $ExSit1$ of Sect. 4.2.2.1 (illustrated in Fig. 4.6 on p. 131, and detailed in Table 4.3 on p. 136) various individual business degradations were discussed, namely g_{b1-1}, g_{b2-1}, g_{b2-2-1}, g_{b2-2-2}: All of them are SLA violation costs, each one specified as a function of time, a so-called service level penalty (slp) function. So, the slp function of each business degradation represents the business degradation characteristic used to specify the business degradation in detail.

Actually these service level penalty functions were already derived in Sect. 2.3.4 from the actual QoS values of affected QoS parameters, the corresponding SLA constraints and SLA penalty definitions. E.g., for the derivation of $slp_{\text{mail3}}(t)$ (business degradation characteristic of g_{b1-1}) the degraded avg. mail sending delay value of $6\ \text{min}$, and the corresponding SLA constraint sla_cnstr_{mail3} and the SLA penalty definition sla_pnlty_{mail3} were used.

An example for an aggregated business degradation is the sum of g_{b2-2-1} and g_{b2-2-2}, describing the overall service level penalties caused by the original resource degradation g_2 for the web hosting service, denoted g_{b2-2} in the fol-

lowing. The service level penalty function for g_{b2-2} (as its business degradation characteristic) can be specifically defined as the sum of the service level penalty functions of g_{b2-2-1} and g_{b2-2-2}.

As seen from the example, the general term *business degradation characteristic* introduced here is only a generalization of the specific term *SLA penalty function* (slp) used for SLA violation costs.

actual instantiation of BIDep-Mod(Char)

The classes of BIDepMod(Char) (in Fig. 4.57) are to be considered as meta classes (e.g., Business Degradation Char). Their (meta) instances are classes themselves, defined for a specific service scenario, and which have actual instances themselves. Especially the meta instances of the meta class *business degradation characteristic* are specific business degradation characteristic types defined for a concrete scenario. Each such business degradation characteristic type corresponds to a declaration of a function of time/duration. Its specific instances are actual definitions of functions of time with concrete values.

In the following sections (Sect. 4.3.4-4.3.10) SI impact dependency models are devised, which can actually be instantiated for concrete scenarios and can be used in combination with BIDepMod(Char).

4.3.4 SIDepMod(Subj:Sv): SI dependency model with degradation dependencies of services

Here, a first possibility for the concrete specification of an SI dependency model is introduced, namely the SI dependency model SIDepMod(Subj:Sv) or SIDepMod(Sv) in short. SIDepMod(Sv) corresponds to the typical form of SI dependency model used and realized in today's industry products concerned with SIA, and is therefore introduced here first. All following SI dependency models (Sect. 4.3.5 to 4.3.10) can be regarded conceptually (not structurally, i.e., concerning class structure) as refinements of SIDepMod(Sv).

only consideration of resources and services

For the specification of degradation dependencies in SIDepMod(Sv), only dependencies between resources and/or (whole) services are considered. That is, the dependent degradations of SIDepMod(Sv) are actually (mainly) determined by their degradation subjects, which are in a coarse-grained manner only to be specified as the degraded resources or the degraded (whole) services. Consequently, aspects such as degradation manner, degradation value accuracy, degradation time (compare Fig. 4.7 on p. 135) are not explicitly specified and so actually neglected in SIDepMod(Sv).

simple example

The following simple example provides a motivation for the use of SIDepMod(Sv): In example scenario of Sect. 2.3 a degradation of the DNS server ($r_{\mathrm{dns\_sv1}}$) entails a degradation of the DNS service (s_{dns}), which is based on this resource. In turn, the latter degradation entails further degradations of the dependent services s_{mail} and s_{web}, which both use s_{dns} as subservice (compare Fig. 2.8 and Fig. 2.6 in Sect. 2.3).

subservice dependencies

As seen from this example, dependencies of services on subservices are taken into account for SIDepMod(Sv). Subservices may be provider-internal or provider-external. So, the meeting of the requirement R1.1 (number of provider domains) is basically approached.

limitations of SIDepMod(Sv)

As the specification of degradation dependencies with SIDepMod(Sv) is only concerned with degradation subjects (here only roughly specified as resources or services), this specification has many limitations: Many information parts (see Fig. 4.7 on p. 135) of a degradation are not (explicitly) covered - as will be discussed below. But, even the subject specification is only very basic. For many appropriate applications various refinements of this specification are necessary, which will be covered in the next sections (Sect. 4.3.5-4.3.7). Here only the basic case, i.e., resources and whole services as subjects, are treated.

generic approach for degradation subject dependencies

In order to allow an appropriate refinement of the subject specification later on in the following sections, the following approach is taken: First, the specification of degradations and degradation dependencies only by means of the respective degradation subjects is discussed on a general basis, and a generic subject-induced SI dependency model, termed SIDepMod(Subj), or SIDepMod(Subj:X) with variable subject specification method X, is introduced. Second, this generic dependency model SIDepMod(Subj:X) is instantiated for the case of simple subject specification, i.e., resources or services, expressed by setting X=Sv and yielding SIDepMod(Subj:Sv). In the following sections (Sect. 4.3.5-4.3.7) further instantiations of SIDepMod(Subj:X) will be made.

So, degradation dependencies derived from degradation subject dependencies in general are discussed first: Fig. 4.58 presents the abstract, generic classes used for the specification of degradations and corresponding degradation dependencies by SIDepMod(Subj) in general (compare Fig. 4.55).

specification of degradation dependencies by subject dependencies

Degradations are primarily specified by their degradation subject. Based on this, degradation dependencies are specified only as dependencies between the degradation subjects (*degradation subject dependencies*). Any further information part for degradation specification (see Fig. 4.7 on p. 135) is not covered at all - as far as degradation dependencies are concerned. But degradations itself may be in addition to the degradation subject (primary part of specification) further described by additional information. This additional information may cover the missing degradation specification parts (other than degradation subject), e.g., mainly degradation time. But as these others information parts are not taken into account for the respective dependencies explicitly, they can only be determined or derived as being equal for source and target of a degradation. This makes mostly sense only for degradation time, less for e.g., degradation manner (“which QoX parameters of the subject are affected?”), as different subjects most often have distinct QoX parameters sets. But even, for degradation time, only the very basic relationship, i.e., “source degradation time is equal to target degradation time” is expressible. Complex temporal relationships are not expressible.

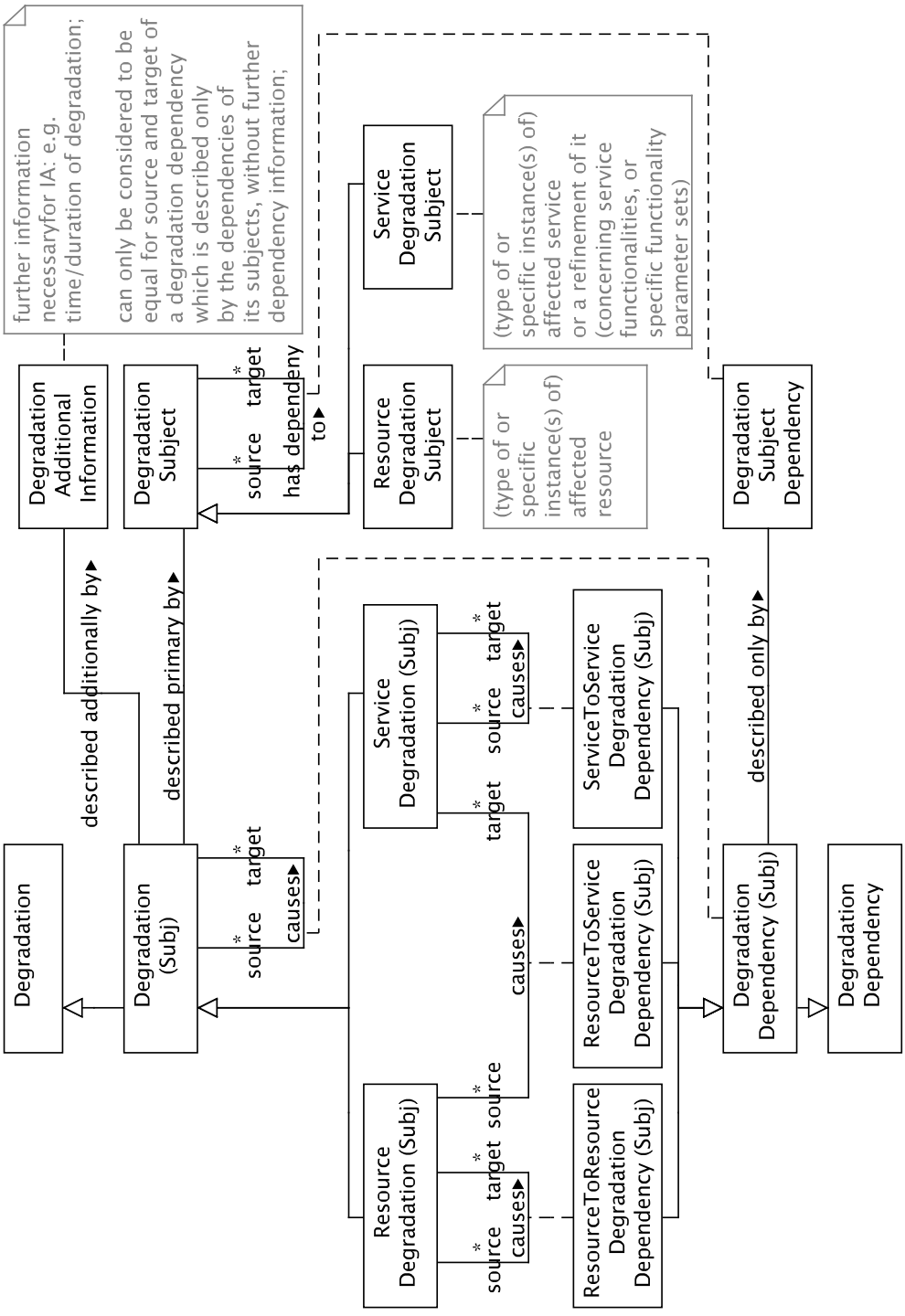

Figure 4.58: Degradations primarily described by their subjects and degradation dependencies only described by dependencies of their subjects, as the constituents of SIDepMod(Subj), refinement of Fig. 4.55

instantiation of abstract SIDepMod(Subj)

Fig. 4.59 extends the abstract class structure of Fig. 4.58 in order to provide a template for actually instantiating SIDepMod(Subj) with a particular specification for degradation subjects, designated as SIDepMod(Subj:X). The variable X denotes an actual particular degradation subject specification type. That is, here degradation subjects and their respective degradation subject de-

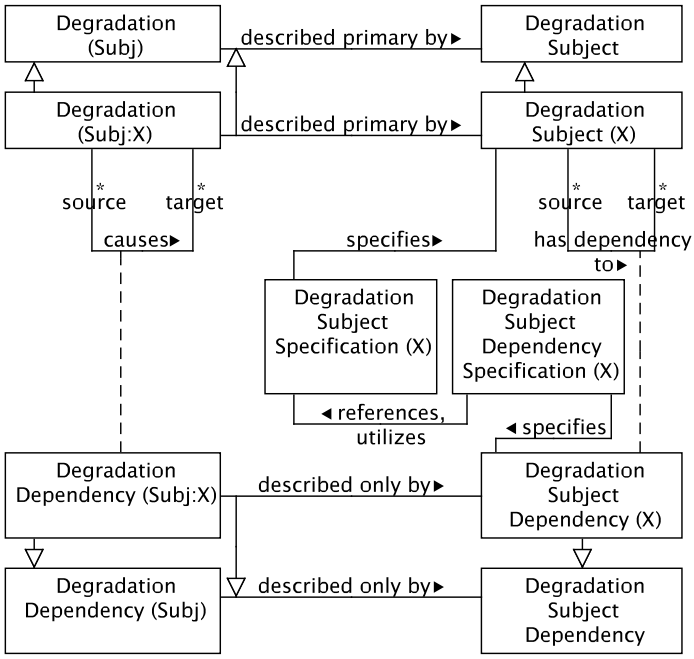

Figure 4.59: Template for a particular refinement of SIDepMod(Subj), refinement of Fig. 4.58

pendencies are specified by particular inter-mediate specifications (denoted by X). For instance, one such specification is introduced in the following for the case of subject described only as resources/services, i.e., the X of SIDepMod(Subj:X) will be instantiated with X=Sv. Other more refined subject (dependency) specification types will be treated in the following sections (Sect. 4.3.5-4.3.7).

Concluding the generic subject-induced SI dependency model, Table 4.7 summarizes the details of the DegDep specification with SIDepMod(Subj)/SIDepMod(Subj:X), i.e., the DegDep specification by using only degradation subjects and their respective dependencies (compare Table 4.5 on p. 223).

SIDepMod(Sv) as instantiation of SIDepMod(Subj)

In the following, a specific instantiation of the abstract SIDepMod(Subj) is made with X=Sv, i.e., the actual SI dependency model SIDepMod(Subj:Sv)=SIDepMod(Sv) is presented. Fig. 4.60 presents the classes used for the specification of degradation subjects and degradation subject dependencies by SIDepMod(Subj:Sv) as a particular case (the simplest one) to instantiate SIDepMod(Subj) (compare Fig. 4.58 and Fig. 4.59 above). As already mentioned above - the degradation subjects are specified either by resource classes or service classes only, with no further refinement.

SIDepMod(Subj:X), with subject specification type X:	
types of dependent degradations:	degradations described mainly by their degradation subjects (specified with some **subject specification type X),**
	secondly described by additional information (other than the subject, e.g., degradation time), which is not considered explicitly for dependencies, and so is ignored or at best only considered/derived to be equal for dependent degradations;
associations of dependent degradations:	degradation dependencies totally determined/derived by the dependencies of their respective degradation subjects;
	$\Longrightarrow$ for dependency specification, most information parts of a degradation (e.g., degradation manner, degradation value accuracy) are not expressible, or only in a very primitive way (e.g., degradation time) as additional information (see above);
	multiplicities left open;
definition of additional dependency constraints:	none;
	especially complex relationship of additional information parts (other than subject) of related degradations cannot be expressed (being ignored or only considered to be equal);

Table 4.7: Overview of DegDep specification for SIDepMod(Subj:X) with generic degradation subjects of type X (compare Fig. 4.58-4.59)

limitations of SIDep-Mod(Subj:Sv) specifically

No further refinement, or subdivision into e.g., service instances of distinct customers/users, or into particular service functionalities of a service, or even a differentiation of service functionality parameter values is performed here (will be addressed in Sect. 4.3.5-4.3.7).

limitations of SIDep-Mod(Subj) in general

But as discussed above and valid for any subject-induced degradation dependency specification (SIDepMod(Subj)), no complex time relationships are expressible. The only simple temporal relationship being expressible or derivable is "source degradation time/duration is equal to target degradation time/-duration". Furthermore, also valid for any subject-induced degradation dependency specification, no degradation manner, degradation value accuracy, or even dynamics in/over time are taken into account (will be covered in Sect. 4.3.8-4.3.10).

Table 4.8 summarizes the details of the DegDep specification with SIDep-Mod(Subj:Sv) as the basic possibility to actually instantiate SIDepMod(Subj) (compare Table 4.7). Moreover, Table 6.2 on p. 370 in Appendix A introduces a corresponding, set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:Sv).

complex example

Following the discussion of the example situation $ExSit1$ from Sect. 4.2.2.1 (illustrated in Fig. 4.6 on p. 131, as well as Fig. 4.12 on p. 146, and de-

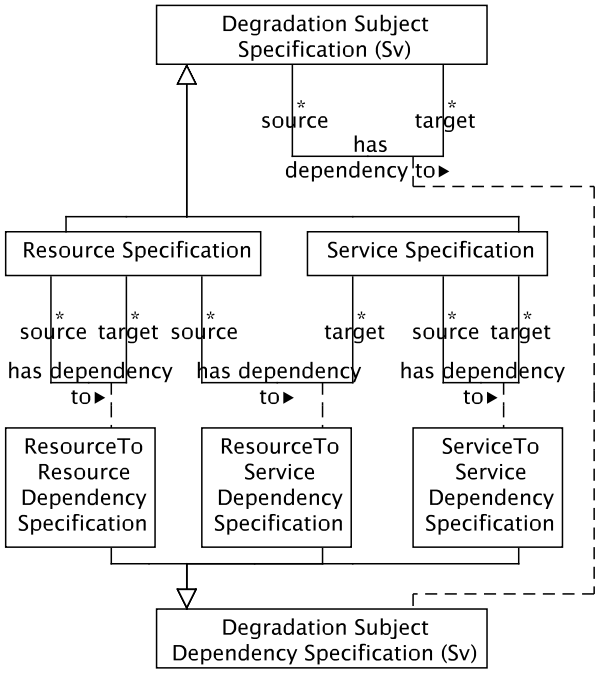

Figure 4.60: SIDepMod(Subj:Sv), refining Fig. 4.59 with X=Sv

tailed in Table 4.3 on p. 136) is continued regarding SIDepMod(Subj:Sv). For $ExSit1$ some resource degradations and service degradations were introduced, namely g_{r1}, g_{r1b}, g_{r2}, as well as g_{s1-1}, g_{s1-2}, g_{s2-1}, g_{s2-2}. If all these degradations and their corresponding dependencies (see Fig. 4.12) are specified by using SIDepMod(Subj:Sv), this results in the following examples of degradation specification and degradation dependency specification: E.g., g_{r1} is described only as a degradation of resource $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$, g_{s2-2} is described only as a degradation of s_{web}. Examples of degradation dependencies utilized for SIA of $ExSit1$ are e.g., $r_{\mathrm{dns\_sv1}} \rightarrow s_{\mathrm{dns}}$ (used for SIA of g_{r1b}) or $r_{\mathrm{afs\_sv1}}(\mathit{path} \in \mathrm{PathListAfs}) \rightarrow s_{\mathrm{store}}$ (used for SIA of g_{r2}), $s_{\mathrm{dns}} \rightarrow s_{\mathrm{mail}}$ (also used for SIA of g_{r1b}).

But many information parts of the resource/service degradation of $ExSit$ (compare Table 4.3 on p. 136) are neglected by using only SIDepMod(Subj:Sv):

- no consideration of service instances (different customers/users), i.e., instances of a service class, i.e., no specification of affected users; e.g., the particular affected user groups of g_{s2-1} or g_{s2-2}.

- no consideration of service/functionality access parameters (refinement of class instantiation); e.g., for g_{r2} the actually affected part of the AFS filesystem is not specified.

- no consideration of multiple service functionalities (refinement of service class); e.g., for g_{s1-1} the specific functionality $f_{\mathrm{mail/use/send}}$ is not specified.

SIDepMod(Subj:Sv) = SIDepMod(Subj:X) with X=Sv	
with resources and (total) services as subjects	
types of dependent degradations:	degradations described mainly by their respective degradation subjects, which are considered to be **resources or (total) services,**
	secondly described by additional information (other than the subject, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived by the dependencies of their respective degradation subjects (see above);
	multiplicities left open;
additional dependency constraints:	none

Table 4.8: Overview of DegDep specification for SIDepMod(Subj:Sv) (compare Table 4.7 with X=Sv, and Fig. 4.60)

- no consideration of specific QoR/QoS, only full or no degradation: i.e., not which QoX parameter set and not at all its specific QoX metric value range; e.g., for g_{r1} the affected QoR parameter link utilization and its concrete value specification $> 60\%$ (permanently in affected time interval) is neglected.

- no consideration of cooperation patterns of dependent degradation (subjects), i.e., no consideration of performance redundancy/load-balancing; e.g., load-balancing of $r_{\mathrm{dns\_sv1}}$ and $r_{\mathrm{dns\_sv2}}$.

- no consideration of temporal change of degradation dependencies or some of their defining aspects (dynamics over time); e.g., redundancy switching on failure.

relationship to current related work

Concluding, it can be said that SIDepMod(Subj:Sv) is a very basic possibility for degradation dependency specification only. Nevertheless, it was introduced as a basis, because it is the de-facto practice utilized and realized in today's industry products for performing SIA. Even more general, most of today's service models are concerned only with this type of degradation dependencies and mostly neglect further degradation details. But for appropriate I/R analysis - concerning granularity and accuracy - these neglected degradation details are often vital.

The aspects covered by the extensions of following sections (Sect. 4.3.5 to 4.3.10) are almost anywhere in today's products missing or not optimally integrated with each other. An exception to this are simple concepts for cooperation patterns (compare SIDepMod(Coop) of Sect. 4.3.9), e.g., using very simple weighting mechanisms (by percent values) for summing up multiple source degradations, i.e., not accurate for e.g., expressing more complex cooperation patterns, or dynamics over time of dependencies.

In the following sections (Sect. 4.3.5 to 4.3.7) refined alternatives for instantiating SIDepMod(Subj:X), each by taking into account further details, are treated (concerning the requirements R1.2 and R1.3). Later on, in Sect. 4.3.8-4.3.10, degradation specification information parts other than degradation subject are covered.

4.3.5 SIDepMod(Subj:SvInst): SI dependency model with degradation dependencies of service instances

In this section, a refined SI dependency model SIDepMod(Subj:SvInst) (or SIDepMod(SvInst) in short) is discussed, which takes into account individual service instances for particular customer/user (groups) of a service.

The consideration of service instances, i.e., different customers/users of a service, is demanded by the requirement R1.2 (granularity of service/functionality definition), as far as this distinction is necessary for accurate I/R analysis.

examples

Examples for such a differentiation of service instances are the following: In general, in the example scenario of Sect. 2.3, there are e.g., the (subject) dependencies of the mail service $r_{\text{mailin\_lrz}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$ or $r_{\text{mailin\_studlmu}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LMU} \cup \text{stud})$. In the case of the specific example situation $ExSit1$, there are the particular affected user groups $GrpMail$ and $GrpWeb$ of the service degradations g_{s2-1} or g_{s2-2} (see Table 4.3 on p. 136).

refinement of SIDepMod(Sv)

SIDepMod(Subj:Sv) is actually designed as a refinement of SIDepMod(Sv) (see Fig. 4.60 in previous section). It represents an alternative, refined way to specify degradation subjects. Thus, basically it also instantiates the abstract dependency model SIDepMod(Subj:X) for specifying degradation dependencies via subject dependencies (see previous section), in this case with X=SvInst. Fig. 4.61 presents the classes used for the specification of degradations and degradation dependencies by SIDepMod(Subj:SvInst) as a possibility to instantiate SIDepMod(Subj:X) (compare the abstract models of Fig. 4.58 and Fig. 4.59, as well as the specific model of Fig. 4.60).

SIDepMod(Subj:SvInst) refines SIDepMod(Subj:Sv) in the following way: The service specification class is refined into two subclasses: a complete service specification covering a service as whole, and service instance specification which pertains to a complete service specification. In addition to that, the service instance specification subclass contains an attribute determining its particular customer/user or group of customers/users.

example application

Consequently, degradation dependencies for specific customer/user (groups) can be specified with SIDepMod(Subj:SvInst), e.g., the dependencies of the mail service scenario mentioned above:
$r_{\text{mailin\_lrz}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$ as well as
$r_{\text{mailin\_studlmu}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LMU} \cup \text{stud})$.
From such specified dependencies the specific user group affected by a service

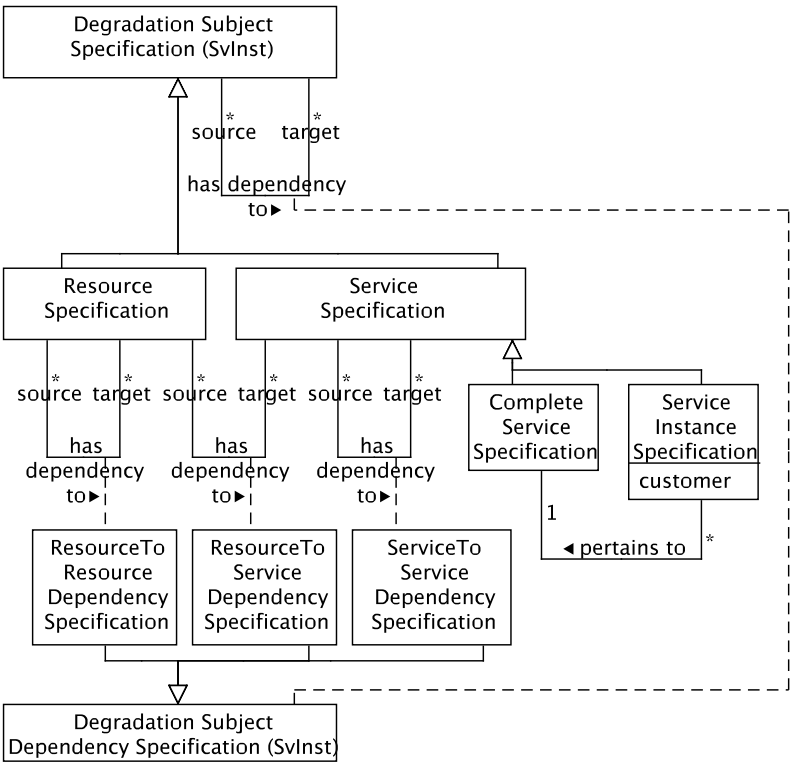

Figure 4.61: SIDepMod(Subj:SvInst), refining Fig. 4.59 with X=SvInst, refining Fig. 4.60

SIDepMod(Subj:SvInst) = SIDepMod(Subj:X) with X=SvInst	
with resources and service instances as subjects	
types of dependent degradations:	degradations described mainly by their respective degradation subjects, which are considered to be **resources or service instances,**
	secondly described by additional information (other than the subject, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived by the dependencies of their respective degradation subjects (see above);
	multiplicities left open;
additional dependency constraints:	none

Table 4.9: Overview of DegDep specification for SIDepMod(Subj:SvInst) (compare Table 4.7 with X=SvInst, and Fig. 4.61)

degradation, e.g., user group $GrpMail$ for degradation g_{s2-1} in $ExSit1$, can be derived.

Table 4.9 summarizes the details of the DegDep specification with SIDepMod(Subj:SvInst), being an instantiation of SIDepMod(Subj). Correspondingly, Table 6.3 on p. 370 in Appendix A introduces a set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:SvInst).

Limitations concerning subject-only degradation dependency model (see previous section) also hold for SIDepMod(Subj:SvInst). Moreover, the granularity of service/functionality specification (requirement R1.2) is still only weakly covered. In the next section another refinement of SIDepMod(Subj:Sv), i.e., a refinement of the service specification, namely the subdivision of service functionalities of a service, is treated. In Sect. 4.3.7 this other refinement is further extended to treat so-called functionality instantiations, a concept which will also subsume the specification possibilities of SIDepMod(Subj:SvInst), i.e., taking into account particular customers/users or groups of them. limitations

4.3.6 SIDepMod(Subj:Fcty): SI dependency model with degradation dependencies of functionalities

Here the SI dependency model, SIDepMod(Subj:Fcty) (or tersely SIDepMod(Fcty)), is introduced which refines the notion of a service into its particular (service) functionalities. That is the subject specification described as resource or service of SIDepMod(Subj:Sv) (Sect. 4.3.4) is again refined, but concerning another aspect as for SIDepMod(Subj:SvInst) (Sect. 4.3.5).

Using only (sub)services as a whole for the specification of degradation subjects is often too general and not specific enough for the purpose of degradation dependency specification. For example, in the example mail service of Sect. 2.3 mostly specific single functionalities are depending on specific resources or specific single functionalities of subservices, e.g., $r_{\text{mailin}} \rightarrow f_{\text{mail/use/recv}}$ or $f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}}$. In particular, e.g., the degradation subject of the service degradation g_{s1-1} (originally entailed by resource degradations g_{r1}nd g_{r1b}) of the example situation $ExSit1$ (Fig. 4.6 on p. 131) is affecting only the mail sending functionality $f_{\text{mail/use/send}}$ (high mail sending delay) and not the whole mail service. For example, the mail receiving functionality is not affected by g_{s1-1} (and its originally entailing resource degradations). example

The refinement of service specifications as a whole to particular functionalities is related to the requirements R1.2 (granularity of functionality definition) and R1.3 (service view/resource view consideration).

Requirement R1.2 demands for an enough detailed specification of service

service functionalities instead of services as degradation subjects

(functionality), mainly in order to specify the corresponding degradation dependencies in an enough detailed manner, and so in order to determine the service degradations as precisely as needed. That is why in general not (sub)services (i.e., the overall functionality of a (sub)service) are taken as the subject of dependent degradations, but instead the different service functionalities of the respective services are considered as the subjects of the dependent degradations. Expressing this fact with the notion of degradation subject dependencies (see Sect. 4.3.4), the specification of the dependent subjects are refined.

Nevertheless, this subdivision into particular functionalities will also subsume the general case, of a whole service being a degradation subject, as it will be discussed below.

resource usage instead of resources as degradation subjects

Similarly as for services, it is useful to allow also to refine the notion of resources, as being the other general kind of service degradation subjects. The same resource may be used in different ways or aspects by a service (functionality), which may result in different refined degradation dependencies taking into account these particular aspects. These ways or aspects to rely on a resource are called the different *resource usages* of the specific resource in the following.

examples of resource usages

For one resource usage the resource may be degraded, while in the same time it may be not degraded for another one. For example, in the example scenario of Sect. 2.3 the mail server r_{mailin} has two different subject dependencies to the two different functionalities $f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LRZ})$ and $f_{\text{mail/use/recv}}$, i.e., $r_{\text{mailin}} \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LRZ})$ and $r_{\text{mailin}} \rightarrow f_{\text{mail/use/recv}}$. With respect to degradation dependencies, this e.g., means that the availability of r_{mailin} in general affects the availability of both functionalities. But in fact, r_{mailin} as whole resource is consisting of subcomponents, e.g., processes running on this machine, especially one/multiple processes related specifically to mail receiving, and one/multiple processes related specifically to mail box access. If only one of these two mentioned classes of processes becomes unavailable, only a degradation for single respective functionalities, either $f_{\text{mail/use/mbox\_access}}(\textit{user} \in \text{LRZ})$ or $f_{\text{mail/use/recv}}$, will be affected concerning availability. In order to differentiate these two types of partial unavailability of r_{mailin}, two particular resource usages $r_{\text{mailin/mail\_recv}}$ and $r_{\text{mailin/mailbox\_access}}$ can be introduced with corresponding refined degradation dependencies to the respective functionalities. This example also illustrates that the subdivision into different resource usages may correspond to the subdivision of a resource into multiple subcomponents. But the former subdivision is more abstract as it only considers the accessing/usage of the respective subcomponent(s) by functionalities and not the whole structure of a resource concerning the (potentially hierarchical organized) interdependencies of its subcomponents.

Similar as for services, the refinement of the resource notion will also allow to express the general case of the whole resource being a degradation subject of service degradation.

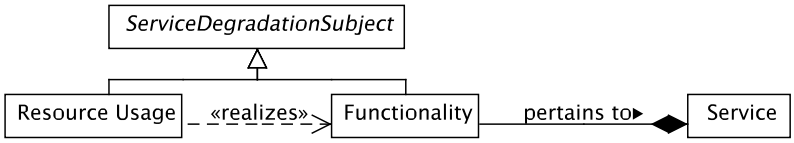

Figure 4.62: Refined kinds of service degradation subjects

refinement of service degradation subject

Fig. 4.62 illustrates the refinement of the two general kinds of service degradation subjects (resources and services, compare Fig. 4.55 on p. 221): The class *service* is replaced by the class *service functionality*, and the class *resource* is replaced by the class *resource usage*. Nevertheless, the refinement is mainly needed for functionalities, and so mainly done for resources only for completeness.

Whereas functionalities as integral part of their service are regarded as being directly and exclusively associated with the service, resource (usages) are not regarded as belonging exclusively to a service. The reasons for this are on the one hand that resource (usages) of a service may be changed or be replaced without changing the service from the customer's point of view, and on the other hand that a single resource (usage) might be used for realizing multiple independent services, so in fact it may not pertain to exactly one service.

overall functionality/resource usage

For each service at least one particular functionality is introduced, one which covers the whole service, i.e., all other functionalities, and so represents the service as a whole. Similarly, for each resource, an overall resource usage is introduced, which covers all other resource usages, and so represents the resource as a whole. Using these overall functionality/resource usage the general case of a whole service or a whole resource being a degradation subject of a service degradation can be expressed. So the new notion of refined degradation subjects is also able to subsume the general cases which are already expressible with SIDepMod(Subj:Sv).

Concerning resources in the examples used in the following, the notion resource usage will often be neglected, i.e., an overall resource usage per particular resource will be usually assumed only. This overall resource usage will be normally designated by the particular resource it represents, instead of using the explicit term resource usage. The refinement made here is mainly useful for functionalities, and maybe used for resource (usages) if appropriate for the specific service scenario.

examples from the example scenario

Table 4.10 contains some examples for resources and functionalities from the example scenario of Sect. 2.3 given in a short notation already introduced in Sect. 2.3.

instantiating SIDepMod(Subj)

After generally introducing the notion of refined service degradation subjects, the dependency model SIDepMod(Subj:Fcty) itself is treated. Similar as SIDepMod(Subj:Sv) of Sect. 4.3.4, SIDepMod(Subj:Fcty) is a particular instantiation of SIDepMod(Subj:X) (see Sect. 4.3.4), with X=Fcty. I.e., it is an alternative for SIDepMod(Subj:Sv), and actually as already indicated above, it is also subsuming this one, by allowing overall service functionalities and

Examples of resources:

- $r_{\mathrm{ldap\_sv1}}$: LDAP server
- $r_{\mathrm{sw\_6}}$: a network switch (compare to Fig. 2.9)
- $r_{\mathrm{iplink}}(r_{\mathrm{sw\_6}}, r_{\mathrm{ipaccess}}(r_{\mathrm{ldap\_sv1}}))$: an IP link connecting the LDAP server to the network switch (compare to Fig. 2.9)

Examples of functionalities:

- f_{web}: overall functionality of the web hosting service
- $f_{\mathrm{web/use}}$: usage functionality of the web hosting service
- $f_{\mathrm{web/mgmt}}$: management functionality of the web hosting service
- $f_{\mathrm{web/use/apage/static}}$: accessing of static web pages as a specific functionality of the web hosting service

Table 4.10: Refined kinds of service degradation subjects from the example scenario in Sect. 2.3

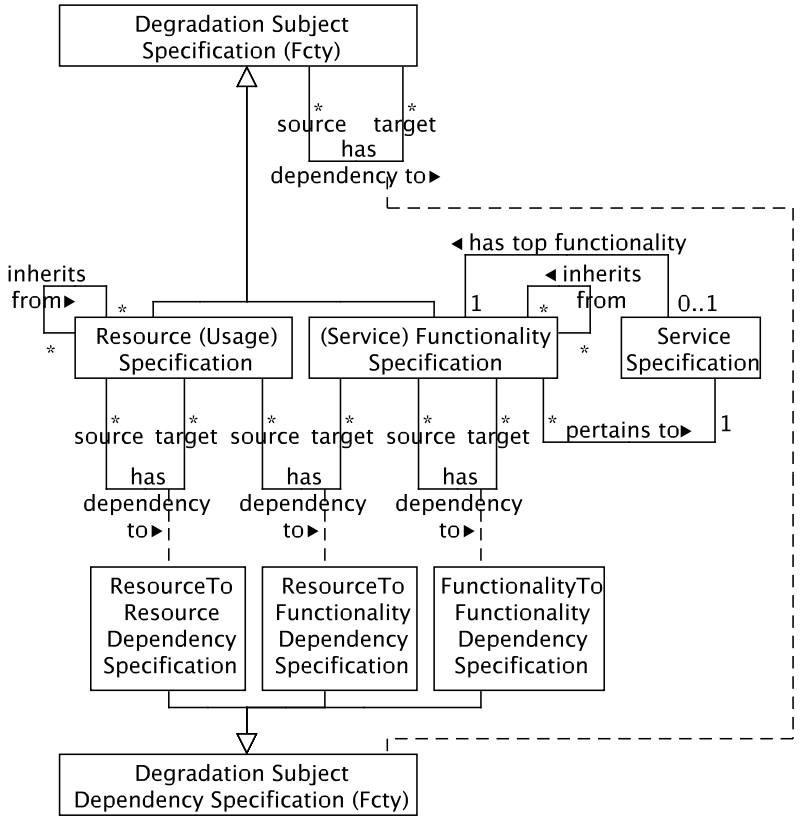

Figure 4.63: SIDepMod(Subj:Fcty), refining Fig. 4.59 with X=Fcty, and also subsuming Fig. 4.60

overall resource usages. Fig. 4.63 presents the classes used for the specification of degradations and degradation dependencies by SIDepMod(Subj:Fcty) as an instantiation of SIDepMod(Subj:X) (compare Fig. 4.58 and 4.59).

In comparison to SIDepMod(Sv) (Fig. 4.60) there are some changes reflecting the refined degradation subjects: *resource specification* and *service specification* of SIDepMod(Sv) are replaced by *resource usage specification* and *functionality specification*. A *functionality specification* pertains to the *service specification* of its particular service.

The class *service specification* retains in the model, but it is itself not considered a degradation subject any more. Instead for each service a specific overall functionality (or top functionality) is defined representing the service as whole as degradation subject. This top functionality is also subsuming all other functionalities of its service. Actually, this becomes possible by allowing the functionalities to be arranged in an inheritance hierarchy (relation *inherits from*) where the top functionality of a service is always the single, top-most functionality in this hierarchy. Using this concept not only one layer of refinement of a whole service into functionalities is possible. Instead also non-top functionalities in the inheritance hierarchy can have children, allowing recursive refinement into service functionalities. Similarly, the class *resource usage specification* allows a refinement (via a corresponding relationship *inherits from*).

SIDepMod(Subj:Fcty) = SIDepMod(Subj:X) with X=Fcty	
with resources and service functionalities as subjects	
types of dependent degradations:	degradations described mainly by their respective degradation subjects, which are considered to be **resources or service functionalities,**
	secondly described by additional information (other than the subject, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived by the dependencies of their respective degradation subjects (see above);
	multiplicities left open;
additional dependency constraints:	none

Table 4.11: Overview of DegDep specification for SIDepMod(Subj:Fcty) (compare Table 4.7 with X=Fcty, and Fig. 4.63)

Particular examples for the subdivision of a service into its functionalities by this inheritance hierarchy are discussed in Sect. 4.3.6.1. Based on this, Sect. 4.3.6.2 is further introducing refined kinds of functionalities in order to tackle requirement R1.3 (functionalities in service-view as well as in resource-view). Table 4.11 gives a summary of the DegDep specification with SIDepMod(Subj:Fcty). Furthermore, Table 6.4 on p. 371 in Appendix A introduces

a set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:Fcty).

limitations

Limitations concerning subj-induced degradation dependency models (see Sect. 4.3.4) in general are valid also for SIDepMod(Subj:Fcty). The possibility to express granularity of service/functionality specification (requirement R1.2) is relatively powerful here compared to the previous SI dependency models. Nevertheless, sets of service interactions (instantiations of service functionalities) are only expressible via inheritance hierarchies. More complex restrictions to particular subsets (if necessary) are not possible. To remove also this limitation, in Sect. 4.3.7, a further refinement is discussed, allowing to express any subset of functionality instantiations, as far as needed for degradation subject specification.

4.3.6.1 Subdivision of services into functionalities: concepts and notations

In the following, the notion of functionality as a subdivision of a whole service is covered in detail.

A service depends mainly on its (service) functionality. Its service functionality comprises all interactions with the roles customer or user, being concerned with the service usage and its management.

Roughly speaking, the service functionality can be divided into usage functionality and management functionality. But also these generic functionalities can be further subdivided into smaller, more specific functionalities, e.g., the e-mail example service's (see Sect. 2.3.1) usage functionality can be roughly subdivided into mail sending and mail receiving.

service functionality as a concept subsuming specific functionality interactions

Moreover, depending on the given service, each of its functionalities represents a specific type of interactions, which are described by terms like service request, service invocation, service session. That is, a single interaction taking place in some period of time between a specific customer or user and the provider. In the following, the general term *service (functionality) interaction* will be used instead of other notions as mentioned above. In general it can be said that each functionality describes some subset of service interactions of the whole service.

functionality classes and their inheritance to differentiate service functionality interactions of different services and within a service

First, a generic concept and notion to describe functionalities (as well as their corresponding service interactions represented by them) as degradation subjects is treated: A type of functionality represents a *class of service interactions* between provider side and customer side with characteristics specific to these type of interaction, e.g., such as specific types of parameters necessary for a specific one of the represented service interactions. Therefore for the purpose of degradation dependency specification, a functionality will be seen as a class, a so-called *functionality class*, of possible service interactions which are represented by this functionality class. On the one hand, this allows for distinguishing between different types of service interactions by different

functionality classes. On the other hand, an *inheritance relationship between functionality classes* can be used to have the notion of generic top functionality classes as well as more specific functionality classes which are subclasses of generic top ones, and to represent only a subset of service interactions of a generic class. For example, the service functionality of any service as a whole can be represented by the service's main functionality class, which can have e.g., as subclasses the more specific usage functionality class and management functionality class of the service (see Fig. 4.64). In turn, both refined functionality classes, can have own subclasses to distinguish between various types of usage or management interactions of the service in question.

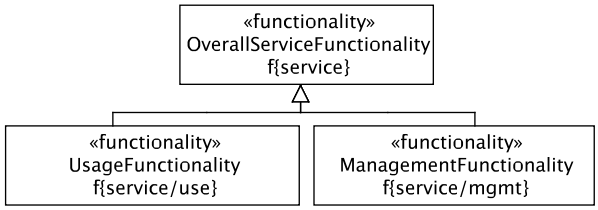

Figure 4.64: Service functionality class hierarchy (top part) of a generic service

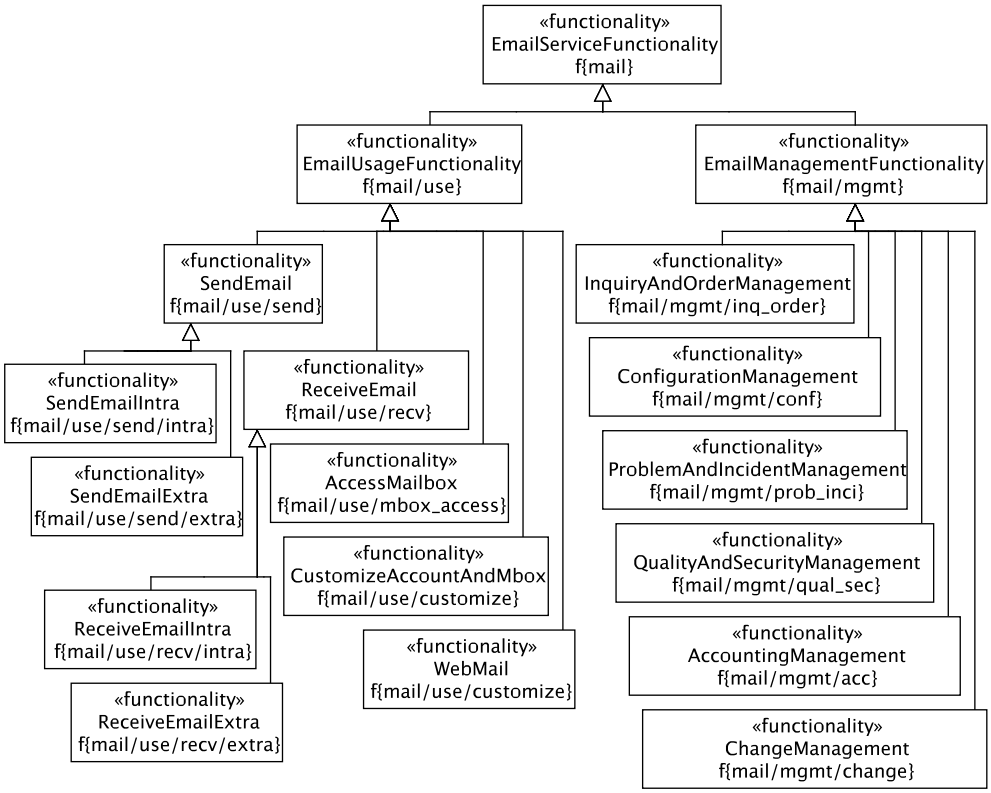

Figure 4.65: Service functionality class hierarchy for the example e-mail service

To sum it up, the functionality of a service is represented by a class hierarchy of functionality classes where each class represents a specific subset of service interactions of the service, i.e., an actual service interaction is an instance of its corresponding functionality class.

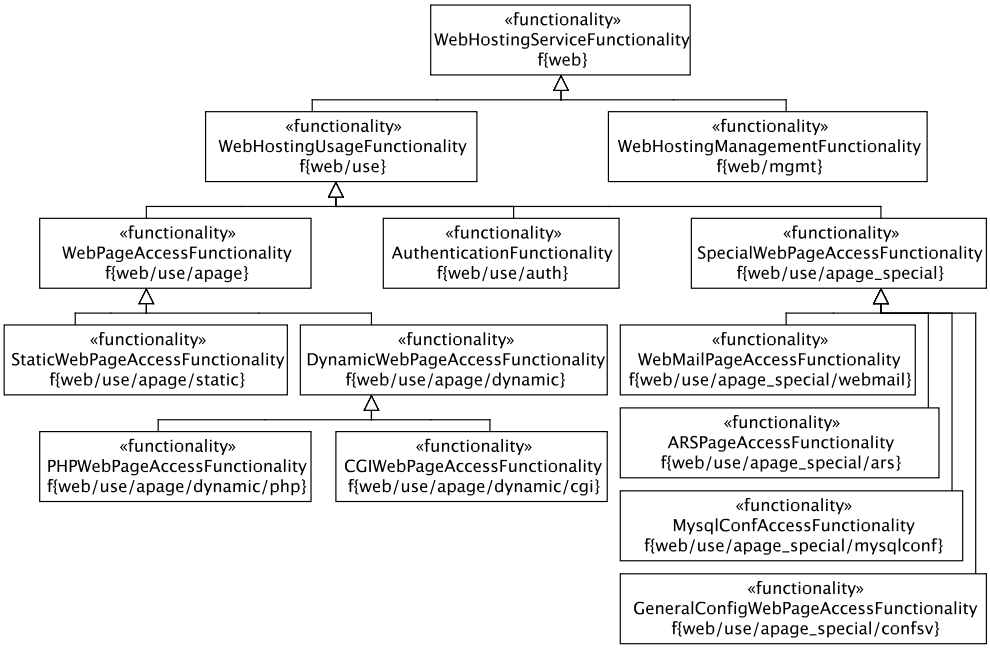

Figure 4.66: Service functionality class hierarchy for the example web hosting service

The specific inheritance hierarchy chosen for a specific given service scenario is not further restricted and specified here. Instead, it can be chosen as detailed as necessary, i.e., with as many hierarchy steps as appropriate. Thereby each service scenario given can be modeled with a specific granularity concerning the differentiation of functionality classes. This approach allows for different specification granularity regarding the term functionality: In an extreme case a service's functionality might be represented by one single functionality class without further subclasses. Alternatively, the functionality can be further subdivided by a detailed inheritance hierarchy of functionality classes with necessary granularity. Fig. 4.65 shows as an example the functionality class hierarchy for the e-mail service of Sect. 2.3.1, while Fig. 4.66 shows the functionality hierarchy of the web hosting service of Sect. 2.3.2.

4.3.6.2 Refined types of functionality

In this section functionality as a kind of service degradation subject is further refined in order to allow an easier transition from the resource view to the service view regarding degradation dependencies.

In the instantiation methodology of the MNM service model for the transition from service view to realization view (or the other way round) service functionalities (as agreed with customers) are provider-internally refined by decomposition as appropriate and afterwards mapped to resources and subservice functionalities.

Therefore regarding the degradation dependencies among functionalities, there are in fact two different types of functionalities on which another func-

tionality depends: First, a functionality of a service can depend on the functionality of a subservice (via a special internal resource, the subservice client, at a specific subservice access point). Second, in order to allow the provider for a more fine-grained decomposition of functionalities than defined together with the customer before mapping to the resources, a service functionality (known to and defined together with the customer) might be decomposed into more specific partial functionalities before mapping to the concrete resources implementing it. I.e., the customer-visible functionality is dependent on some other functionalities, but being internal to the service and normally unknown to the customer side.

For the example mail service of Sect. 2.3.1 such a decomposition of functionalities is useful e.g., for the functionality $f_{\text{mail/use/recv}}$, i.e., the receiving of e-mails for an user account from other users or from outside the mail domain. Actually, $f_{\text{mail/use/recv}}$ can be decomposed into various functionalities like $f_{\text{mail/rsrc/greylist}}$ (graylisting of incoming e-mail), $f_{\text{mail/rsrc/blacklistcheck}}$ (blacklist checking of sender domain), $f_{\text{mail/rsrc/spamcheck}}$ (spam checking of incoming e-mail), $f_{\text{mail/rsrc/viruscheck}}$ (virus checking of incoming e-mail), $f_{\text{mail/rsrc/queue\_rcvd\_mail}}$ (inserting of received e-mail in the right mail queue, as there are different mail queues which differ in their final target mail incoming server for eventually storing them: i.e., TUM, non-student users of LMU, students of LMU, other users), $f_{\text{mail/rsrc/relay\_to\_inbox}}$ (relaying to final incoming mail server), $f_{\text{mail/rsrc/store\_in\_inbox}}$ (storing in inbox). Instead of making $f_{\text{mail/use/recv}}$ to have direct dependencies on the resources $r_{\text{greylistsv}}$, $r_{\text{blacklistsv}}$, $r_{\text{spamchecksv}}$, $r_{\text{viruschecksv}}$, r_{mailin}, $r_{\text{mailin\_tum}}$, $r_{\text{mailin\_lmu}}$, and $r_{\text{mailin\_studlmu}}$, it can be modeled to have dependencies on the given resource functionalities which in turn have the direct dependencies on the respective resources: E.g., $f_{\text{mail/rsrc/greylist}}$ depends on $r_{\text{greylistsv}}$, $f_{\text{mail/rsrc/queue\_rcvd\_mail}}(\textit{customer} \in \text{LMU})$ depends on r_{mailin}, $f_{\text{mail/rsrc/queue\_store\_in\_inbox}}(\textit{customer} \in \text{LMU})$ depends on $r_{\text{mailin\_lmu}}$, and $f_{\text{mail/rsrc/queue\_store\_in\_inbox}}(\textit{customer} \notin \text{LMU} \cup \text{TUM})$ depends on r_{mailin} (compare resource dependencies described in Sect. 2.3.1 for further information).

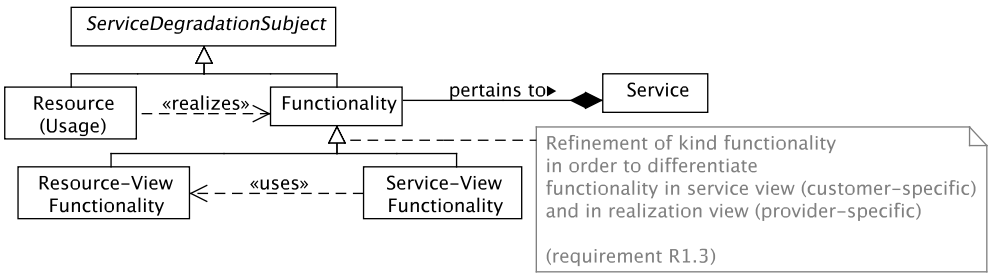

Figure 4.67: Kinds of service degradation subjects with refined functionality kind (refinement of Fig. 4.62)

In general, this refinement by decomposition relates also to requirement R1.3 which demands for different levels of abstractions in the realization view. Concluding, two sub-kinds of functionalities of a service can be distinguished:

- so-called *service-view functionality* for describing a common view between provider and customer about the functionality of the service and
- so-called *resource-view functionality* as a refinement of (more-detailed, but same level of abstraction) of the service functionality; it is only known provider-internally.

The names are given in reflecting the notions of service-view and realization-view of the MNM service model (see Sect. 2.2). Fig. 4.67 shows the resulting refined hierarchy for kinds of service degradation subjects, which is a refinement of Fig. 4.62.

The usage of resource-view functionalities for modeling of a service is optional and can be applied where it is appropriate.

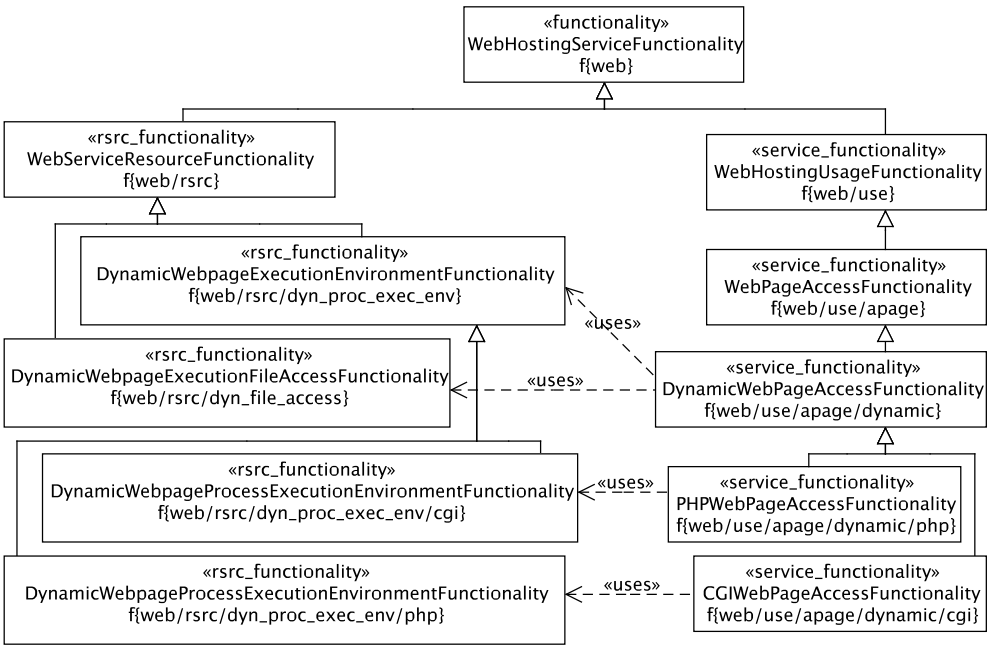

Figure 4.68: Refined functionality class hierarchy as UML class hierarchy

Refined functionality classes can be represented as UML classes, too. Fig. 4.68 shows in UML notation the hierarchy of some resource-view functionality classes of the example web hosting service, namely some resource-view functionalities used for the service functionality $f_{\text{web/use/apage/dynamic}}$ respectively its refined functionalities $f_{\text{web/use/apage/dynamic/cgi}}$ and $f_{\text{web/use/apage/dynamic/php}}$. Moreover, Fig. 4.69 illustrates in UML notation the example already mentioned above - the functionality class declarations of the resource-view functionalities used by the service-view functionality $f_{\text{mail/use/recv}}$ of example mail service. In Table 4.12 some examples for resource-view functionality classes from the example scenario of Sect. 2.3 are presented.

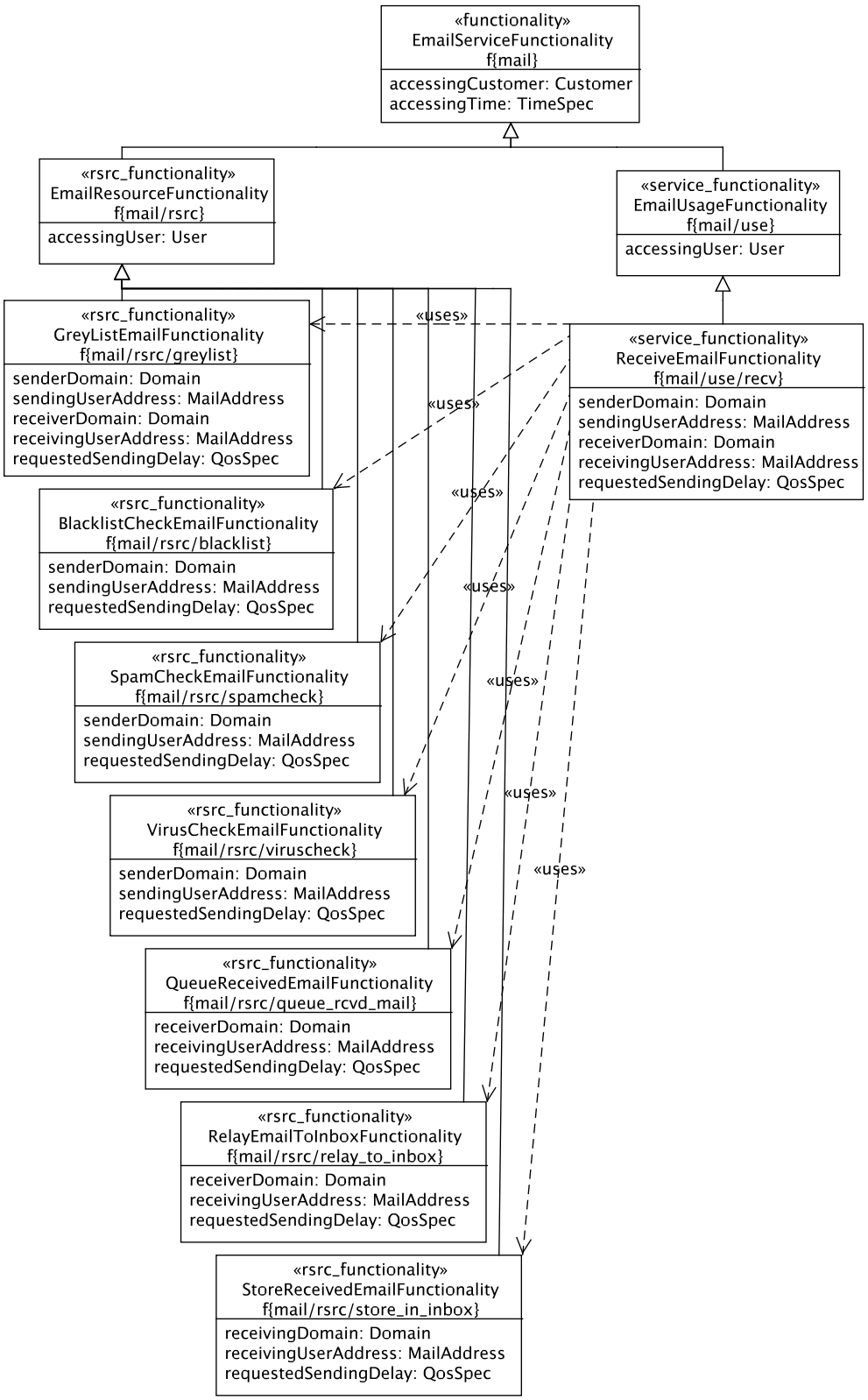

Figure 4.69: Functionality class declaration hierarchy as UML class hierarchy

Examples of some resource-view functionalities:

- resource-view functionalities for refinement of service-view functionality $f_{\text{mail/use/rcv}}$ of the mail service:
  - $f_{\text{mail/rsrc/greylist}}$: graylisting of incoming e-mails
  - $f_{\text{mail/rsrc/blacklistcheck}}$: blacklist checking of the sender domain addresses in incoming e-mails
  - $f_{\text{mail/rsrc/spamcheck}}$: spam checking of incoming e-mail
  - $f_{\text{mail/rsrc/viruscheck}}$: virus checking of incoming e-mail
  - $f_{\text{mail/rsrc/queue\_rcvd\_mail}}$: inserting of received e-mail in the right mail queue (different queues for mail which differ in their final target mail incoming server for eventually storing them, i.e., TUM, non-student users of LMU, students of LMU, other users)
  - $f_{\text{mail/rsrc/relay\_to\_inbox}}$: relaying to final incoming mail server (for receivers not in LMU or TUM there is nothing to do here)
  - $f_{\text{mail/rsrc/store\_in\_inbox}}$: storing of received mails on the respective incoming mail server depending on the receiver of the mail
- refinement of the $f_{\text{web/use/apage/dynamic}}$ and its subfunctionalities $f_{\text{web/use/apage/dynamic/cgi}}$ and $f_{\text{web/use/apage/danymic/php}}$:
  - $f_{\text{web/rsrc/dyn\_proc\_exec\_env}}$: process execution environment for creating dynamic web pages.
  - $f_{\text{web/rsrc/dyn\_file\_access}}$: file system access for creating dynamic web pages (realized by local filesystem on web server machine as well as subservice functionality $f_{\text{nfs/use}}$ in case of CGI scripts.
  - $f_{\text{web/rsrc/dyn\_proc\_exec\_env/cgi}}$
  - $f_{\text{web/rsrc/dyn\_proc\_exec\_env/php}}$

Table 4.12: Examples of resource-view functionalities and service-view functionalities

4.3.7 SIDepMod(Subj:Fcty/Inst): SI dependency model with degradation dependencies of functionality instantiations

Here the previously developed SI dependency model SIDepMod(Subj:Fcty), which allows to differentiate between different service functionalities as degradation subjects, is further refined. The notion of functionality as a degradation subject was in Sect. 4.3.6.1 specifically defined as a class covering some subset of similar service interactions of a service. The SI dependency model developed here, SIDepMod(Subj:Fcty/Inst) (or SIDepMod(FctyInst) in short), will allow complex restrictions of degradation subject specifications to particular subsets of service interactions. This complex restriction goes beyond the simple restriction to the subset of all service interactions pertaining to a functionality class. Requirement R1.2 (granularity of functionality definition) will hereby completely be covered.

examples

For the example scenario of Sect. 2.3.1 restrictions on some functionalities as subjects of target or source degradation of a degradation dependency are necessary in order to specify these degradation dependency in detail, e.g., $f_{\text{mail/use/send}}(\textit{authentication} = \text{yes})$, $f_{\text{mail/use/recv}}(\textit{receiver}$ is mailinglist$)$, $f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$. Especially the last one of these examples, which is a restriction to a service instance (set), is already expressible with SIDepMod(Subj:SvInst) (Sect. 4.3.5), but not the other ones, which are concerned with functionality parameters other than specification of customer/user (group). The here discussed DepMod(Subj:Fcty/Inst) provides a generic possibility to express any functionality parameter (set) restriction, comprising also such parameters concerned with customer/user specification. Further examples, concerned with the example situation $ExSit1$, for restriction to particular functionality parameter sets of a functionality as a degradation subject are (Table 4.3 on p. 136 and Fig. 4.12 on p. 146): $r_{\text{afs\_sv1}}(\textit{path} \in PathListAfs)$ for degradation g_{r2}, as well as $f_{\text{ip/use/connect}}(\textit{path} = r_{\text{mailout}}, \cdots, anywhere)$ for degradation g_{s1-0} (compare especially p. 145).

A similar refinement for resource (usage) classes is possible. Examples are $r_{\text{afs\_sv1}}(\textit{path} \in PathListAfs)$ and its respective dependency above, and more generally from the example scenario of Sect. 2.3:
$r_{\text{dns\_sv1/resolve\_domain}}(\textit{domain} \in List1) \rightarrow f_{\text{dns/use/resovle\_domain}}(\textit{domain} \in List1)$,
and $r_{\text{mailin/mail\_recv}}(\textit{sender\_domain} \in List1) \rightarrow$
$f_{\text{mail/use/mail\_recv}}(\textit{sender\_domain} \in List1)$.
The latter two examples of refined dependencies can be utilized in combination, when the failure to resolve a particular set of domain names entails a failure to receive mails from these domains.

Chiefly, with respect to degradation dependencies, the refinement to subject instantiations allows to refine single, particular degradation dependencies, whose specification is too rough by using subject classes alone. In

contrast, the differentiation of different subject classes (functionalities or resource usages) allowed basically to differentiate different degradation dependencies, i.e., with similar source subjects. The differentiation of similar source subjects (by classes with inheritance, $f_{\text{mail/use}}$ differentiated into e.g., $f_{\text{mail/use/send\_mail}}$ and $f_{\text{mail/use/recv\_mail}}$) makes it possible to actually follow only the relevant degradation dependencies to the relevant target degradations. But subject instantiation can also serve the purpose of differentiation of multiple degradation dependencies, as the examples from Sect. 2.3 $r_{\text{mailin\_lrz}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{LRZ})$ and $r_{\text{mailin\_tum}} \rightarrow f_{\text{mail/use/recv}}(\textit{receiver} \in \text{TUM})$ show.

Subj:FctyInst most elaborated subject specification

The refinement to instantiations (service interactions here) of classes will actually complete the degradation subject specification possibilities developed in this impact analysis framework. As being a refinement of SIDepMod(Subj:Fcty) (Fig. 4.63), SIDepMod(Subj:FctyInst) is also an instantiation of the generic SIDepMod(Subj) (see Fig. 4.58) for specifying degradations dependencies mainly by their corresponding degradation subject dependencies. So, SIDepMod(Subj:FctyInst) represents an alternative to SIDepMod(Subj:Sv), SIDepMod(Subj:SvInst) as well as its predecessor SIDepMod(Subj:Fcty). It actually also subsumes every one of these other possibilities concerning the power of expression, which is directly clear for SIDepMod(Subj:Fcty) and SIDepMod(Subj:Sv). For SIDepMod(Subj:SvInst) this subsumption of expression power will be shown later in Sect. 4.3.7.1.

generic subject instantiation

Nevertheless, the refinement to a SI dependency model for specification of particular subject instantiations (e.g., of SIDepMod(Subj:FctyInst)) of a subject class, which is itself priorly specified with prior existing SI dependency model (e.g., of SIDepMod(Subj:Fcty)), is a general approach. This approach might be applied also to other subject-induced SI dependency models, e.g., SIDepMod(Subj:Sv). In this thesis this general refinement approach is specifically applied to SIDepMod(Subj:Fcty) only, as this results in the most complex subject-induced dependency model necessary here. Thus, as it is a general approach, it is first done generically for any existing subject-induced SI dependency model (SIDepMod(Subj:X)), and afterwards specifically for SIDepMod(Subj:Fcty). The generic approach, presented in the following, shows how the approach can be applied to any subject-induced SI dependency models other than SIDepMod(Subj:Fcty).

Fig. 4.70 illustrates in a class structure how the subject specification of a prior existing subject-based SI dependency model (e.g., SIDepMod(Subj:Fcty)) is refined to express individual instantiations of the prior subject specification.

A subject specification class $SubjSpecClass$ (e.g., *functionality specification* in Fig. 4.63 on p. 240) of a prior existing SIDepMod(Subj:X) (e.g., X=Fcty) is taken to be a meta class. That is, the instances of $SubjSpecClass$ (e.g., functionality specifications, such as $f_{\text{mail/use/send}}$) are classes themselves (e.g., functionality classes), termed $SubjSpecClass$-classes. Each one has particular instantiations (e.g., functionality instantiations, i.e., service interactions),

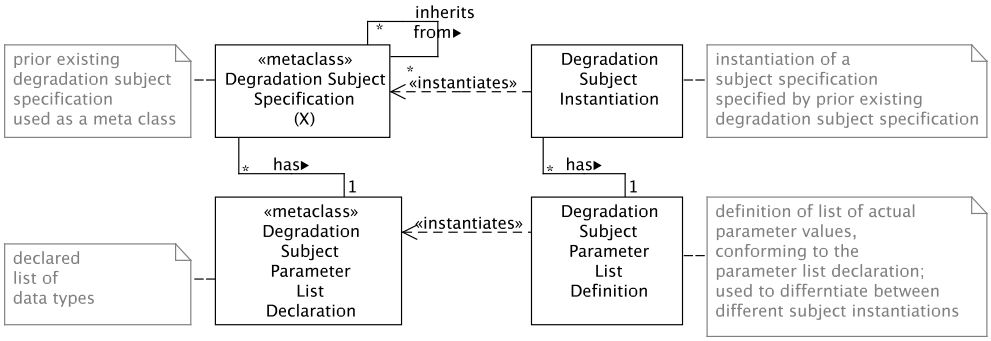

Figure 4.70: Generic approach of reusing an existing degradation subject specification as a meta class with multiple instantiations differentiated by parameter value list

being called $SubjSpecClass$-instantiations. These instantiations can be specified particularly in SIDepMod(Subj:X/Inst).

For this purpose, $SubjSpecClass$-classes have a *subject parameter list declaration*, which is a list of named parameters with type declarations. Corresponding $SubjSpecClass$-instantiations have a *subject parameter list definition*, which is a list of parameter definitions (name/value pairs) corresponding to the parameter type declarations of its $SubjSpecClass$-class.

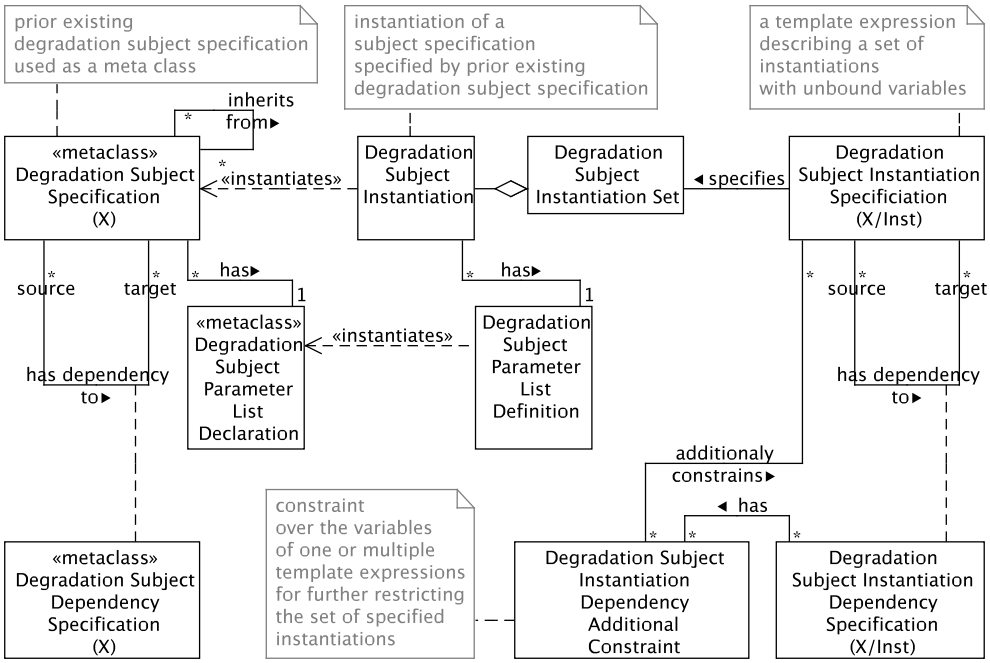

Figure 4.71: SIDepMod(Subj:X/Inst), refining prior existing SIDepMod(Subj:X) reusing its degradation subject specifications as meta class with multiple instantiations

refined SIDepMod for instantiations

Fig. 4.71, based on Fig. 4.70, introduces the complete class structure for a refined SI dependency model SIDepMod(Subj:X/Inst), which refines the subject specification of a prior existing subject-induced SI dependency model SIDepMod(Subj:X) (with some X, e.g., X=Fcty) to a subject specification restricted

to particular subject instantiation set. For example, applied to X=Fcty (will be done in detail below), the degradation subjects in SIDepMod(Subj:Fcty/Inst) (or SIDepMod(FctyInst) in short) are subsets of instantiations of the subjects in SIDepMod(Subj:Fcty), i.e., instantiations of functionality classes, i.e., service interactions in the case of X=Fcty.

Degradation subject specifications in SIDepMod(Subj:X/Inst) are actually template specifications which describe subsets of instantiations of a $SubjSpecClass$-class specified prior in SIDepMod(Subj:X). A degradation dependency in SIDepMod(Subj:X/Inst) basically specifies a relationship between a set of source subjects and a set of target subjects, each described by such a template specification. But in addition to that, these templates may contain (common) variables, which can be additionally constrained, to restrict further the dependent subsets of degradation subject instantiations. Concerning such additional dependency constraints compare especially Fig. 4.52 on p. 218.

example

A very basic example for such a degradation dependency is $subject\_type_{source}(attribute_1 = X) \rightarrow subject\_type_{target}(attribute_2 = Y)$ with the additional constraint $X == Y$.

SIDepMod(Subj:X/Inst)	
with instantiations of a prior existing subject specification as refined subjects	
types of dependent degradations:	degradations described mainly by their respective degradation subjects, which are considered to be **subsets of instantiations**
	secondly described by additional information (other than the subject, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived by the dependencies of their respective degradation subjects (see above);
	multiplicities left open;
additional dependency constraints:	possibility to constrain the dependent sets of subject instantiations by the use of templates with variables;

Table 4.13: Overview of DegDep specification for SIDepMod(Subj:X/Inst) (compare Table 4.7)

Concluding, the resulting dependency model SIDepMod(Subj:X/Inst) is a subject-induced one, i.e., a particular instantiation of SIDepMod(Subj) (see Sect. 4.3.4), the same as its predecessor SIDepMod(Subj:X). Table 4.13 gives a summary of the DegDep specification with SIDepMod(Subj:X/Inst) in general.

specific instantiation for SIDep-Mod(FctyInst)

Following, the specific instantiation of SIDepMod(Subj:X/Inst) for X=Fcty is performed. Thus, SIDepMod(Subj:Fcty/Inst) (or tersely SIDepMod(FctyInst)) is a particular instantiation of SIDepMod(Subj:X/Inst) and so also one of SIDepMod(Subj). Fig. 4.72 illustrates the class structure used

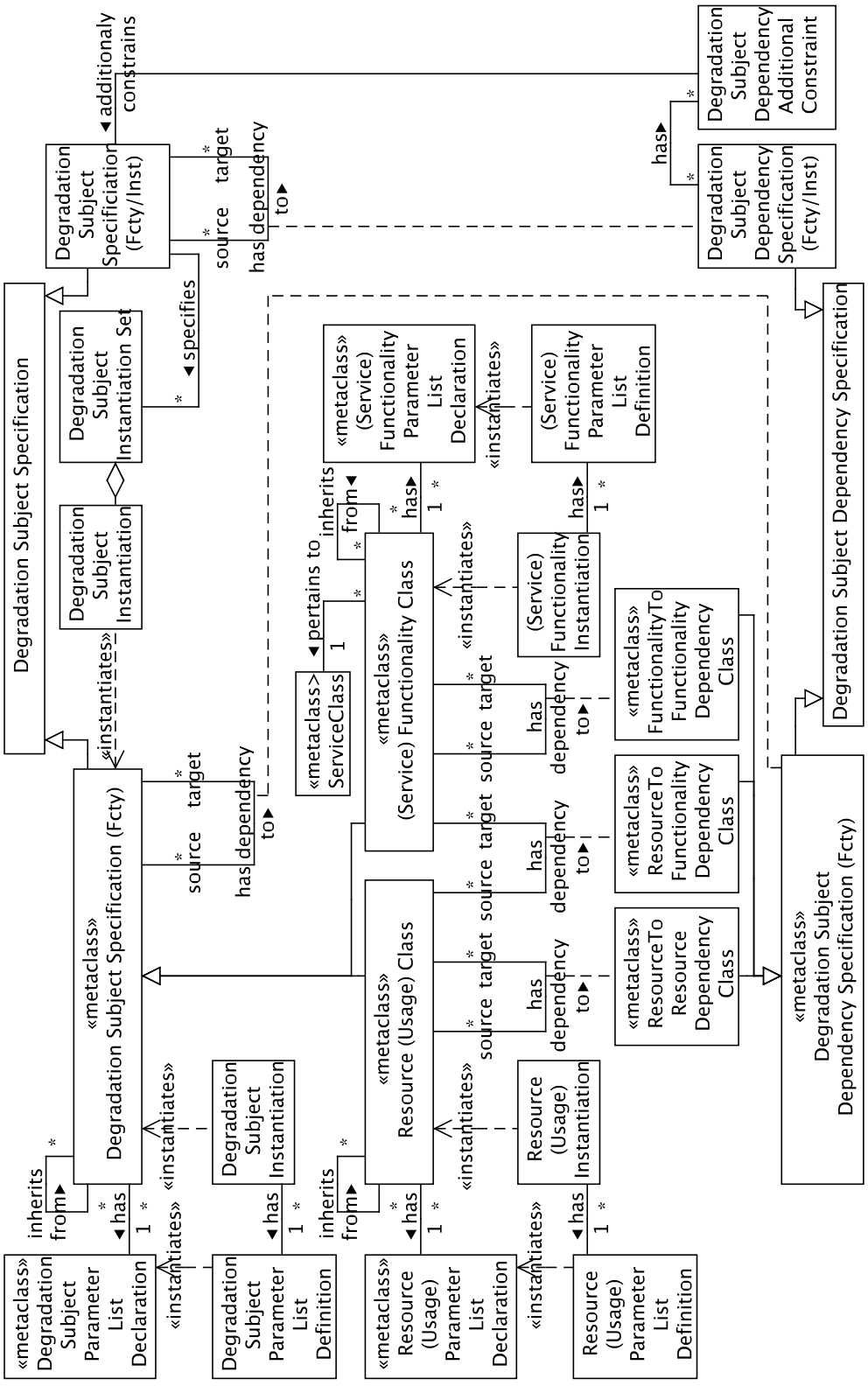

Figure 4.72: SIDepMod(Subj:Fcty/Inst), refining Fig. 4.59 with X=FctyInst, refining Fig. 4.63 as well as Fig. 4.61

for the specification of degradations and degradation dependencies by SIDepMod(Subj:Fcty/Inst), as an instantiation of SIDepMod(Subj:X/Inst) (compare Fig. 4.71, 4.63). Basically Fig. 4.72 is based on Fig. 4.63 on p. 240 (SIDepMod(Subj:Fcty)). Now, the subject specification classes (now becoming meta classes) have appropriate classes for functionality/resource parameter declarations. Accordingly, classes for subject instantiations are added with conforming subject parameter definitions. Moreover, subject specifications in SIDepMod(Subj:Fcty/Inst) are templates restricting to a particular subset of functionality/resource instantiations. Degradation dependencies relate sources/-targets specified by such template specifications. The actual subset of dependent functionality/resource instantiations are further restricted by constraints over common variables in the template specifications.

examples

The motivating examples given at the beginning of this section, can be specified in this way: For instance, the dependency $r_{\text{dnssv1/resolve\_domain}}(\textit{domain} \in List1) \rightarrow f_{\text{dns/use/resovle\_domain}}(\textit{domain} \in List1)$, can be more explicitly specified as $r_{\text{dnssv1/resolve\_domain}}(\textit{Domain: domain} = \text{X}) \rightarrow f_{\text{dns/use/resovle\_domain}}(\textit{Domain: domain} = \text{Y})$, with additional dependency constraint $X == YandX \in List1$. Furthermore, as an example from $ExSit1$, $r_{\text{afs\_sv1}}(\textit{path} \in PathListAfs) \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{user} \in GrpMail)$, used for deriving g_{s2-1} from g_{r2}, can be more explictly specified as $r_{\text{afssv1}}(\textit{FilePath: path} = \text{X}) \rightarrow f_{\text{mail/use/mbox\_access}}(\textit{UserSpec: user} = \text{Y})$ with additional constraint $X \in PathListAfs$ and $Y.mailbox \in X$.

For the last example the constraint uses the function evaluation or attribute access $Y.mailbox$. In general a dependency constraint may use additional helper functions, attribute accessors, relationships (such as $==$, $<$, $>$, but not limited to these ones) as necessary.

SIDepMod(Subj:Fcty/Inst) = SIDepMod(Subj:X) with X=FctyInst	
with resource (usage) and service functionality instantiations as subjects	
types of dependent degradations:	degradations described mainly by their respective degradation subjects, which are considered to be **sets of resource (usage) instantiations or, sets of service functionality instantiations,**
	secondly described by additional information (other than the subject, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived by the dependencies of their respective degradation subjects (see above);
	multiplicities left open;
additional dependency constraints:	possibility to constrain the dependent sets of subject instantiations by the use of templates with variables;

Table 4.14: Overview of DegDep specification for SIDepMod(Subj:Fcty/Inst) (compare Table 4.7 with X=FctyInst and Fig. 4.72)

Sect. 4.3.7.1 is discussing functionality instantiations and functionality parameters in detail. Similarly, Sect. 4.3.7.2 treats resource instantiations and resource (usage) parameters. Table 4.14 gives a summary of the DegDep specification with SIDepMod(Subj:Fcty/Inst) (compare Table 4.13). Table 6.9 on p. 376 in Appendix A introduces a set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:Fcty/Inst).

limitations

Limitations concerning subj-based degradation dependency model (see Sect. 4.3.4) in general are valid also for SIDepMod(Subj:Fcty/Inst), i.e., aspects of degradation specification beyond degradation subject. But the specification of degradation subjects is very powerful and allows to cover the targeted requirements R1.2 (granularity of functionality definition) and R1.3 (service-view/resource-view coverage) as far as it is needed for degradation dependency specification. Sect. 4.3.8-4.3.10 are concerned with the missing degradation specification beyond degradation subject.

4.3.7.1 Functionality instantiations and functionality parameters

In the following, as a continuation of Sect. 4.3.6.1, refinement of a service specification as a degradation subject is further extended. In Sect. 4.3.6.1 the service specification was refined into particular functionalities (or functionality classes) which are for each service organized in an inheritance hierarchy. Each functionality class represents some subset of similar functionality instantiations (or specifically called service interactions) of its service. Here, the specification of particularly restricted subsets of service interactions by means of functionality parameter declarations and definitions is treated. This restriction goes beyond the simple restriction to all instantiations of a functionality class.

As discussed previously - in accordance with requirement R1.2, it should be allowed to refine the notion of functionality classes as a degradation subject specification to sets of functionality instantiations. Such a refinement is concerned with specific details to allow differentiation and restriction of specific instantiation subsets as far as necessary for degradation subject specification.

Here generic examples for such a refinement by introducing functionality parameters are introduced. Furthermore, a first possibility to take into account the dynamics of dependencies during an I/RA run (requirement R2.1) in terms of the time-dependencies for functionalities is introduced.

example

In the example of Sect. 2.3.1 restrictions on some functionalities as subjects of target or source of a degradation dependency were necessary in order to specify these dependency in detail, e.g., $f_{\mathrm{mail/use/send}}(\textit{authentication} = \mathrm{yes})$, $f_{\mathrm{mail/use/recv}}(\textit{receiver} \in \mathrm{LRZ})$, $f_{\mathrm{mail/use/recv}}(\textit{receiver} \in \mathrm{LMU} \cup \mathrm{stud})$.

Each functionality class of a service (on any level of the inheritance hierarchy) represents a subset of the service interactions (its instances) of this service with some common characteristics. In other words, each functionality class is

concerned with some common type of access (service interaction) to usage or management functionality of the service.

functionality parameters to distinguish specific service functionality instances of the same functionality class

Furthermore, the specific instances of a service functionality class, i.e., the possible service interactions represented by it, can be distinguished by different invocation parameter values, such as the specific customer or user accessing the functionality, the time or some sort of session id, requested QoS parameter ranges, or any other additional parameter. For example, for the web hosting example service's *accessWebPage* functionality ($f_{\mathrm{web/use/apage}}$) can have an additional parameter 'URL'. The necessary parameters, including their data type and allowed values, of a service interaction are depending on its functionality class, i.e., each functionality class determines the list of necessary *functionality parameters*. This is similar as for functions and methods in programming languages, e.g., as in C, C++, or Java. So, in this respect a service functionality class can be compared to a function, and one of its actual or possible service interactions can be compared to an actual or possible function call. Therefore, for each functionality class a *list of functionality parameters* has to be declared including a type specification for each parameter (*functionality parameter type*).

different generic kinds of functionality parameters

Each functionality class has its individual list of parameters, but this should in most cases probably include parameters like the ones described above, mainly the specific accessing role or at least the associated SLA, and the access time.

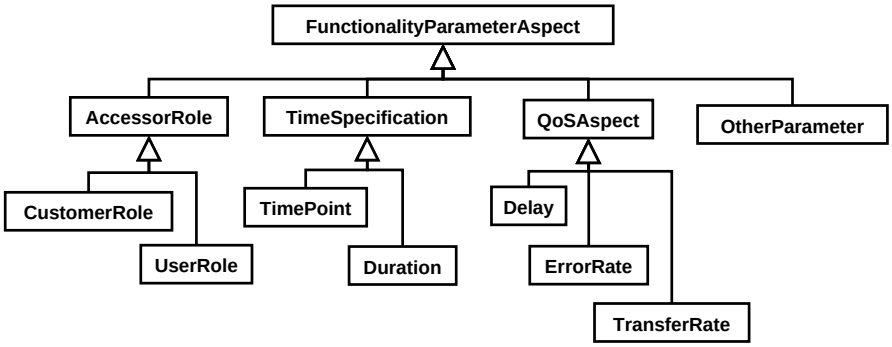

Figure 4.73: General kinds (aspects) of functionality parameters

To sum it up, the types of functionality parameters can be generally classified by different kinds (using the term *kind* in the meaning *type of type*) of functionality parameters (*functionality parameter kind*):

- SLA parameters/Accessing role parameters: making it possible to identify the SLA and contract, i.e., the specific customer side, but also the specific role of this customer side which accesses the functionality.
- time parameters: time point, time period, session id etc. , depending on the specific functionality
- requested/negotiated QoS parameters for the accessed functionality
- further functionality parameters: may be very specific to type of functionality

Fig. 4.73 presents an overview of the different kinds of functionality parameters introduced above.

Everything necessary to distinguish between functionality interactions regarding degradation subject specification, may be used as functionality class parameter. This may also include particular *specifications of access point references* (e.g., IP addresses for IP service in Sect. 2.3.3, sender address explicitly specified by the user in an e-mail to be sent) as far as necessary. In the case of management functionalities, it may include functionality classes themselves as parameter types, e.g., for change or order management purposes (e.g., if the order management for different usage functionalities is depending on different resources respectively basic management functionality).

Each parameter type may have its own *type hierarchy*. That is, a parameter type, e.g., *TimeSpecification*, might have different, more specific subtypes, e.g., *TimePeriodSpecification*, *TimeInstantSpecification*, *TimeOfDaySpecification*. Another example might be the parameter type *URL* for the functionality $f_{\text{web/use/apage}}$ of the web hosting service, which has e.g., as subtypes *WebPageURL*, *ImageURL*, *DownloadableObjectURL*. The subtypes give a first possibility for restricting the service interactions of a functionality class to a particular subset e.g., for specifying a functionality as dependent object in a dependency.

specification of a functionality parameter by name and type

Additionally to its type definition, a functionality parameter declared for a functionality class has a *parameter name* in order to uniquely identify it within the context of its functionality class.

UML classes to represent functionality classes

There is an obvious way of representing an functionality class in UML: A functionality class can be represented by an UML class with the UML class attributes denoting the functional parameters of the functionality class. At this, the UML attribute type corresponds to the functionality parameter type, and the attribute name accordingly corresponds to the name of the functionality parameter. In Fig. 4.74 two specific functionality classes taken from the example services in Sect. 2.3 are depicted as UML classes.

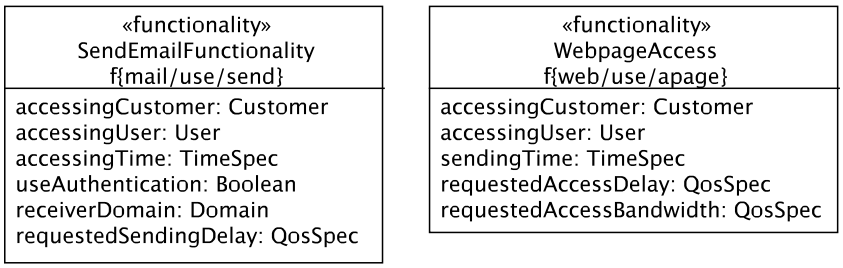

Figure 4.74: Representation of functionality class with their functionality parameters as UML classes

inheritance of functionality classes and inherited parameters

The relationship between inheritance of functionality classes and the parameter list are as follows: The list of parameters of a subclass is a specialization of the parameters of its upper class. Specialization of parameter list here means that the parameter list of the upper class is included in the parameter list of the

subclass, possibly with some of the parameter types restricted to subtypes of the original parameter type. Each subclass is free to add additional parameters to the list of parameters of its upper class. Fig. 4.75 shows an example, where a part of the functionality inheritance hierarchy of the example mail service of Sect. 2.3.1 is represented by an UML class hierarchy including inherited attributes representing inherited functionality parameters (compare with Fig. 4.74 especially concerning *SendEmailFunctionality*).

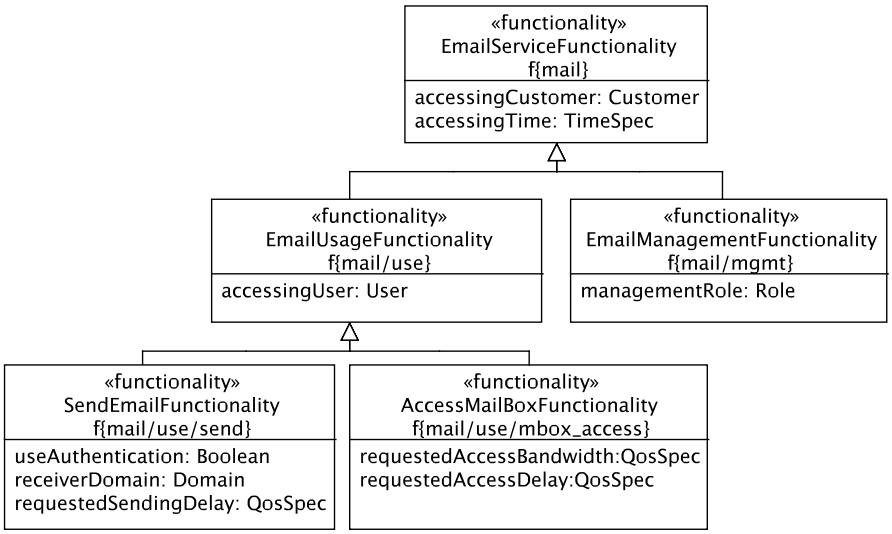

Figure 4.75: Functionality class hierarchy including functionality parameters as UML class hierarchy

declaration and definition of functionality classes

Finally, it can be said, that each functionality class has its declared functional parameter list, and a functionality class together with its parameter list will be called *functionality class declaration* in this thesis, similarly as in C or C++ function declarations or method declarations.

Instead, the term *functionality class definition* will be reserved for the actual specification and definition of a functionality class by means giving more details concerning the realization of the functionality. This, of course, includes all details about dependencies on resources and on other functionalities, but is not limited to. Furthermore, in general this comprises the complete service logic necessary to realize the functionality. But for the purpose of this thesis, this term will only be applied to aspects relevant for I/R analysis.

The distinction between declaration and definition of functionality classes relates to the two representation for functionalities proposed by the MNM service model. There, on the one hand, a functionality, also called a process, is represented as a use case in an UML use case diagram with the users or customers accessing it represented by actors. This representation is similar to the notion functionality class declaration. It allows inheritance between functionalities to be modeled as inheritance between use cases, and further gives a simple possibility to represent dependencies among functionalities by use case inclusion or use case extension. Otherwise, a functionality or process in the MNM service model can be further specified by UML activity dia-

grams and even more detailed by UML collaboration diagrams describing the full service logic for that functionality. This refined and detailed specification relates to the here used term functionality class definition.

instantiations and template instantiations as value restrictions for functionality classes

In the following a generic notion for specifying subsets of a functionality class by restricting the value ranges is introduced. As a functionality class represents all its corresponding service interaction as its instances, and a functionality class instance (*functionality instantiation*) with fixed values for all its functional parameters represents exactly one specific service interaction, the notion *functionality template instantiation* is introduced as an intermediate construct. A functionality template instantiation of a functionality class represents a subset of the service interactions of its functionality class by restricting the values of the functional parameter in some way: e.g., no restriction at all, restriction to a subtype, prescribing a certain condition over the parameter values, or even specifying a concrete, single value for it.

Examples of usage for the notion functionality template instantiation are:

- A service instance (service restricted to a specific customer): this can be seen as a special case of a functionality template instantiation of the services' top functionality class with only the accessing role parameters restricted to the specific customer.
- A functionality regarded in a certain time range can be described as functionality template instantiation with only the time parameters restricted in appropriate manner.
- A functionality with a certain parameter, e.g., as *sendingDomain* for *sendEmail* can be regarded as a functionality template instantiation with this specific parameter restricted to the specific value or subset.
- Or even, all of the described types of restrictions can be combined in a single functionality template instantiation.

subject instantiation specification

Template instantiations correspond to the degradation subject specifications of SIDepMod(Subj:Fcty/Inst) (see Fig. 4.72 on p. 253).

functionality class instantiations as UML objects

In a similar manner as functionality classes together with their parameters can be represented by UML classes and their attributes, functionality class instantiations can be represented by UML objects accordingly. Fig. 4.76 gives an example of such an illustration of functionality class instantiations pertaining to the functionality classes represented in Fig. 4.74.

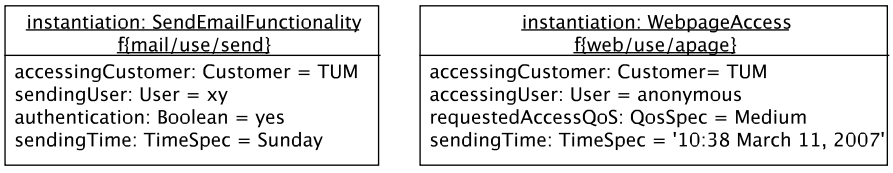

Figure 4.76: Functionality class instantiations as UML objects

Putting all concepts introduced so far together, for the purpose of service degradation subject specification, the functionality of a service is in abstract

manner declared by an inheritance hierarchy of functionality classes whereat the granularity of this hierarchy can be chosen depending on the given scenario. Each functionality class has its specifically declared functional parameter list which can be chosen freely as appropriate.

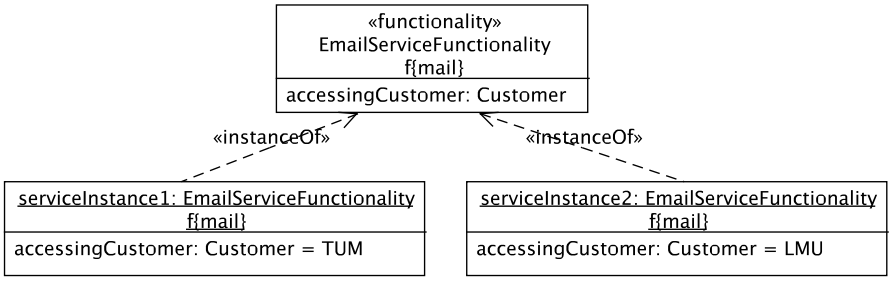

Figure 4.77: Service instances as special instantiations of the generic functionality class represented as UML objects

service instances as special functionality class instantiations

Especially, choosing appropriate parameters values for the SLA parameters (see above) allows to differentiate different service instances of a service: A service instance for a specific customer (with respective SLA) can be regarded as a functionality template instantiation of the service's main functionality class where the SLA functional parameters are instantiated to the specific customer, whereas the other possible parameters are left unspecific. Fig. 4.77 illustrates this for the example mail service in UML.

further use of functionality class instantiations

Concluding, the functional parameter concept allows to treat both service as well as its service instances for a specific SLA and to easily switch from one of these notions to the other. But this concept allows even more. By sub-classing of parameter types or by instantiating parameters other than SLA parameters one can further distinguish various interaction subtypes of a functionality. Examples are:

- restriction of a time parameter: access of a functionality during a specific time interval.
- restriction of a further parameter: e.g., for AFS filesystem access, the restriction to a specific directory.

This will prove extremely useful, when considering degradations of specific functionality template instantiations. For both introduced examples, considering degradation of them, this means:

- restriction of a time parameter: degradation taking place during a specific time interval, i.e., availability of the respective functionality is only restricted in this specific period (first approach to fulfill requirement R2.1, the dynamics of dependencies during an I/RA run).
- restriction of a further parameter: partial outage of the AFS filesystem, i.e., only a particular part of the AFS cells are affected by a degradation (compare degradation g_2 of example I/RA run in Sect. 2.3.4).

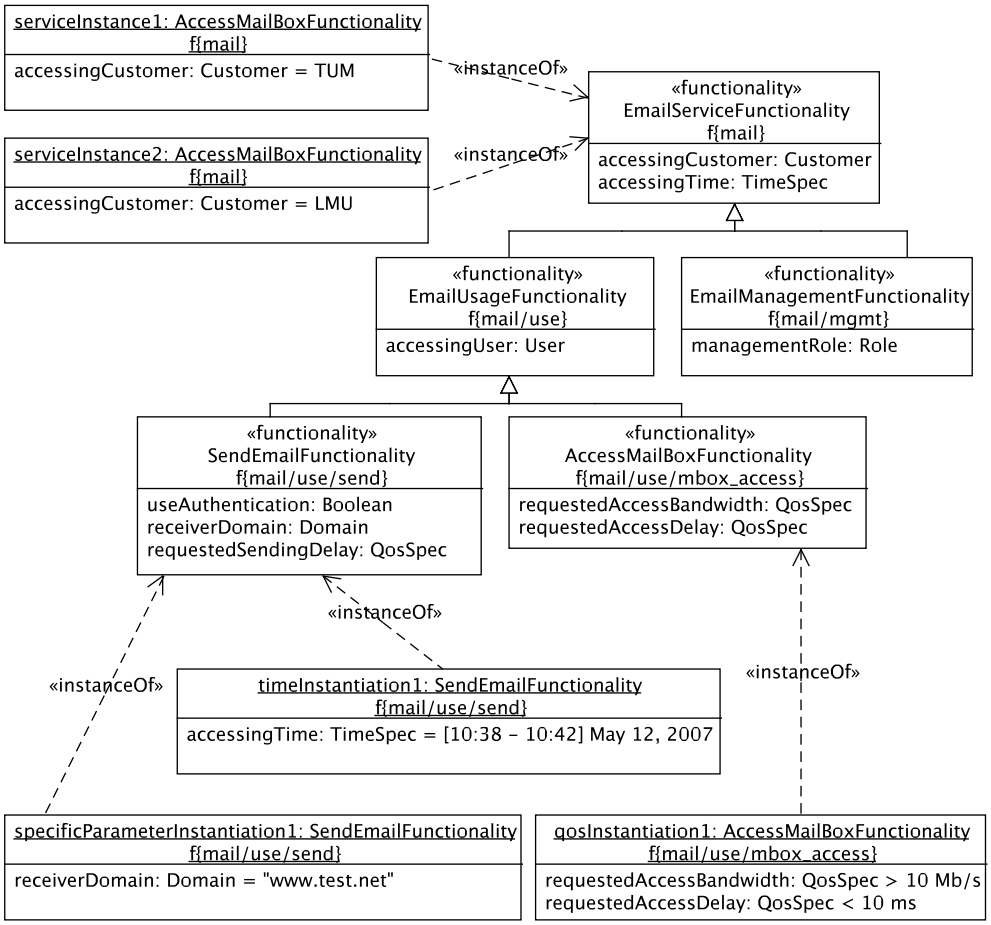

Figure 4.78: Examples of functionality class instantiations represented as UML objects

In Fig. 4.78 some further examples of instantiations of functionality classes of the example mail service are given in UML notation (compare Fig. 4.75). Moreover, in Table 4.15 some examples for functionality classes and their specific instantiations regarding the example scenario of Sect. 2.3 are presented in textual notation.

summary

Concluding, Fig. 4.79 illustrates the generic notions of functionality classes, functionality class instantiations, and functionality class template instantiations as a means to specify functionality of a given service scenario with appropriate granularity as needed for determination of service impact.

Examples of functionality class (template) instantiations:

- $f_{\text{mail/use/send/extra}}($
  $useAuthentication : Boolean =$ yes,
  $accessingCustomer : Customer =$ TUM,
  $accessingUser : User =$ xy,
  $accessingTime : TimeSpec =$ Sunday
)

- $f_{\text{web/use/apage}}($
  $accessingCustomer : Customer =$ LMU,
  $accessingUser : User =$ anonymous,
  $accessedURL : Url =$ /info/,
  $requestedSendingDelay : QosSpec <$ 10 min,
  $accessingTime : TimeSpec =$ "10:38 March 11, 2007"
)

with $f_{\text{mail/use/send/extra}}$ and $f_{\text{web/use/apage}}$ being specific functionality classes.

Both examples are in fact not simple instantiations, but template instantiation as both represent multiple functionality class instantiations, because the parameter "accessingTime" in the first case and the parameter "requestedSendingDelay" in the second case are only constrained to a value range instead of to a single value.

explanation for the set of parameters: necessary to distinguish between different customers/users if dependencies on resources/subservices are different, e.g., mail boxes on different parts of AFS filesystem.

Table 4.15: Examples of functionality classes instantiations and functionality class template instantiations

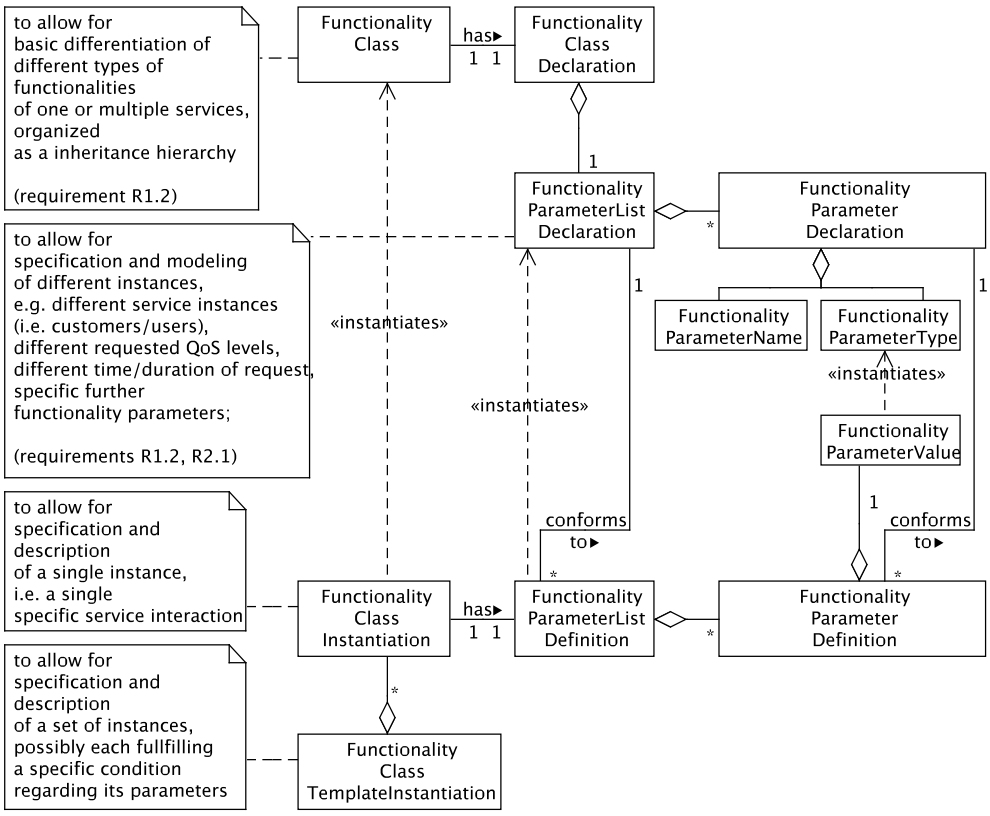

Figure 4.79: Notion of functionality classes, instantiations, template instantiations to specific functionality with appropriate granularity

Examples of resource (usage) class (template) instantiations:

- $r_{\mathrm{mailin}}($
  $useAuthentication : Boolean =$ yes,
  $accessingCustomer : Customer =$ TUM,
  $accessingUser : User =$ xy,
  $accessingTime : TimeSpec =$ Sunday
  $)$
- $r_{\mathrm{webs\,v(3)}}($
  $accessingCustomer : Customer =$ LMU,
  $accessingUser : User =$ anonymous,
  $accessedURL : Url =$ /info/,
  $requestedSendingDelay : QosSpec <$ 10 min,
  $accessingTime : TimeSpec =$ “10:38 March 11, 2007”
  $)$

Table 4.16: Examples of resource (usages) from the example services of Sect. 2.3

4.3.7.2 Resource usage instantiations and resource usage parameters

In Sect. 4.3.6 resources as degradation subject were refined into so-called resource usages. Here instantiations of resources (or specifically resource usages) are treated in detail introducing concepts as it was done for functionalities in the previous section.

Similarly, as functionalities are modeled by functionality classes which represent their set of instances, i.e., its service interactions, resource (usages) are modeled by so-called *resource usage classes* which represent also their instances, which are called *resource usage instances* in turn.

Moreover, similarly as for the instances of functionality classes, different resource usage instances of the same resource usage class are distinguished by parameters, namely the *resource access parameters*. Examples of such distinguishing parameters are (similarly as for resources):

- accessing role parameters, i.e., parameters identifying the user or customer or similar concept on behalf of that the resource access is performed: e.g., e-mail address.
- parameters for specifying the access time and duration.
- requested/negotiated QoR/QoD parameters which are derived from requested/negotiated QoS parameters on the functionality level
- further parameters as necessary, e.g., directory name of the file accessed from an AFS server.

Each resource usage class has a specific set parameters defined by names and corresponding types which determines its *resource usage class declaration*. A specific instance of a resource usage class has for each parameter assigned a unique value of corresponding type.

Moreover, the following notions are defined in a analogous manner as for functionalities (compare Sect. 4.3.6.1): *resource usage parameter type*, *resource usage parameter kind*, *resource usage class inheritance*, *resource usage declaration*, *resource usage definition resource usage instantiation*, *resource usage template instantiation*.

Furthermore, resource usage classes and their instantiations can similarly as functionality classes and instantiations represented in UML by UML classes and UML objects accordingly.

examples

In Table 4.16 some resource usage classes and their specific instantiations from the example scenario of Sect. 2.3 are presented in textual notation.

4.3.8 SIDepMod(QoX) and SIDepMod(QoXInst): SI dependency models with dependencies of QoX degradations

In this section SI dependency models for taking into account degradation specification aspects other than degradation subject, namely quality degradation aspects, are developed. They are actually based on the degradation subject specification of any one of the previously developed SI dependency models (Sect. 4.3.4-4.3.7).

The particular degradation specification aspects (compare Fig. 4.7 on p. 135) degradation manner (affected QoR/QoS parameter sets), and degradation value accuracy combined with degradation time are basically covered here. This is related to the requirements R2.2 and R2.3, which demand for multiple types of degradation types (affected QoR/QoS parameter sets) per subject, as well as multiple value (ranges) per degradation type (value accuracy), value accuracy potentially combined with degradation time, e.g., described as a function of time/duration of QoR/QoS values.

notion of QoX parameter

The general kinds of service degradation subjects are resources and service (functionalities). Quality of a resource is normally described by *QoR parameters* (also often called *QoD parameters*), quality of a service (functionality) is normally described by *QoS parameters* (agreed upon with the customer). To subsume both terms, QoR/QoD and QoS parameters, the general term *QoX parameter* is introduced to generally designate quality parameters for degradations and their subjects.

characteristics of QoX parameters

QoR/QoD and QoS parameters, i.e., both types of QoX parameters, share some common characteristics anyway: First, they relate to a specific *QoX subject* (degradation subject, for the purpose of I/R analysis and of SI dependency models specification). Second, they are based on a clear definition, including a *QoX measurement metric* as well as the actual *QoX measurement methodology* being appropriate for the metric (compare basic introduction of QoS parameters in Sect. 2.3.1 on p. 37). The difference between them is basically that QoS parameters are agreed upon with the customer, and relate to service (functionality) in a way that is understandable, comprehensible, and useful for the customer. QoR/QoD are focused on resources, i.e., on the actual realization of the service, and so normally only known and understandable provider-internally.

For the degradation specification and degradation dependency specification, these general characteristics of QoX parameters have the following consequences (compare Fig. 4.7 on p. 135): First, per degradation subject there might be different QoX parameter (sets) which affect it, i.e., have it as QoX subject. In terms of degradation specification, there might be various possibilities for the degradation manner in which a degradation subject can be affected. Different combinations of degradation subject and degradation manner (QoX parameter sets) might have different dependencies among each

other. That is, the sole specification of a degradation subject, as it is done by subject-induced SI dependency models SIDepMod(Subj:X) (Sect. 4.3.4-4.3.7) is often not enough for degradation dependency specification. An example for this from $ExSit1$ (see Table 4.3 on p. 136) is the high mail sending delay (g_{s1-1}) caused by the high link utilization (g_{r1}). In contrast, a complete outage of the link (total unavailability) would have caused also a total outage (total unavailability) of the mail sending functionality. Concluding, a generally applicable SI dependency model should have the possibility to take into account the degradation manner.

Second, each degradation manner, i.e., QoX parameter (set), has concrete values (normally per time/duration), which conform to the QoX metric and are measured actually by its QoX measurement methodology. For degradation specification, these aspects are covered by the terms degradation value accuracy and degradation time (compare Fig. 4.7 on p. 135). Different combinations of degradation subject, degradation manner, and degradation value accuracy/time often have different dependencies among each other. This is e.g., true for the above mentioned example of high link utilization and high mail sending delay (compare Table 4.3). Furthermore, in general, a source degradation with degradation values which are relatively small, may not even cause a target degradation: e.g., the relatively high web page access delay (g_{s1-2}, compare Table 4.3) whose degradation value is nevertheless too less in degree to entail a business degradation. So, a generally applicable SI dependency model has also to take into account degradation value accuracy (possibly combined with degradation time).

approach for SI dependency model

The development of a SI dependency model which actually covers all degradation specification aspects mentioned above, is actually done in two steps here:

First, the SI dependency model SIDepMod(QoX) is discussed, which allows to take into account degradation manner (QoX parameter sets) for degradation dependency specification. SIDepMod(QoX) is actually based on the degradation subject specification of one of the previously developed subject-induced SI dependency models SIDepMod(Subj:X) for some X (e.g., X=FctyInst). So, in fact there are multiple instantiations of SIDepMod(QoX) depending on the actually utilized subject degradation notion X. For this reason the full designation of this SI dependency model is SIDepMod(QoX/Subj:X) with specific X. The abstract form SIDepMod(QoX) is used as a template designation subsuming any one of these specific ones.

Second, based on SIDepMod(QoX/Subj:X), a refined SI dependency model, SIDepMod(QoXInst/Subj:X), is introduced which takes into account degradation value accuracy, potentially combined with degradation time, i.e., taking into account actual QoX value ranges/specifications for the respective degradation manner (QoX parameter set).

SIDepMod(QoX)

Fig. 4.80 illustrates the class structure used for the specification of degradations and degradation dependencies by SIDepMod(QoX/Subj:X) (compare Fig. 4.55). SIDepMod(QoX) actually reuses the subject degradation specifi-

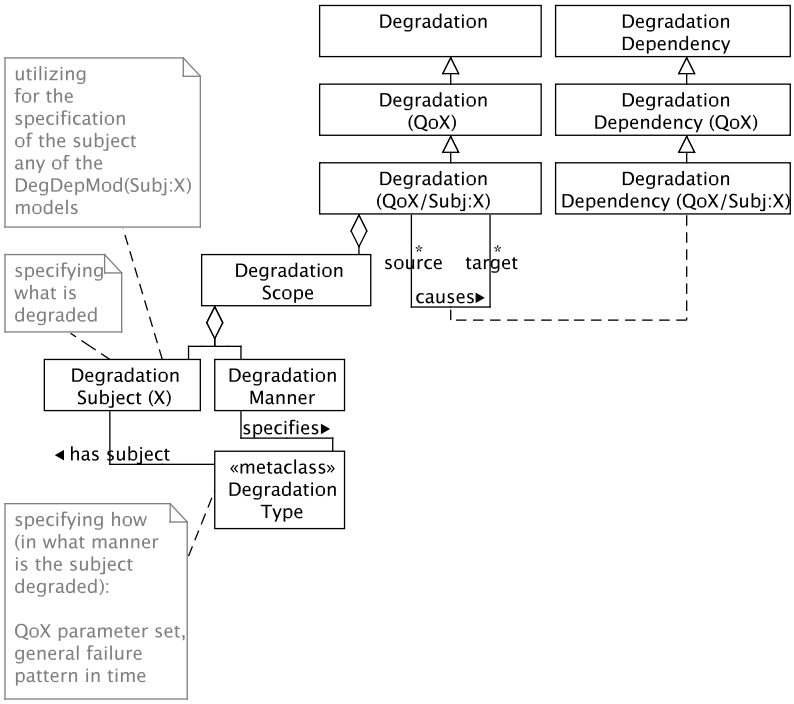

Figure 4.80: SIDepMod(QoX/Subj:X): considering QoX degradation types, refining Fig. 4.55, reusing SIDepMod(Subj:X) for some X

cation of some SIDepMod(Subj:X), X $\in$ { Sv, SvInst, Fcty, FctyInst }, for the specification of its degradation subject. The degradation manner is specified by a so-called *degradation type*, which actually determines the affected QoX parameter (set). The degradation type has to conform to the specified degradation subject, i.e., the determined QoX parameter (set) has to have the degradation subject as QoX subject. Together, the specification of degradation subject and degradation manner cover the complete specification of the degradation scope or degradation value granularity (compare Fig. 4.7 on p. 135).

Examples of degradation types (marked by abbreviation gt) in $ExSit1$ (Table 4.3)) are: $gt_{highlinkutilization}$ for $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$, and $gt_{unavailability}$ for $f_{\mathrm{mail/use/mbox\_access}}(\mathit{user} \in GrpMail)$. Further possible degradation types (which may be dependent on each other) for the respective subjects are: $gt_{highpacketloss}$, $gt_{lowreliability}$ for $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$, and $gt_{highmailboxaccessdelay}$, $gt_{lowmailboxaccessbandwidth}$ for $f_{\mathrm{mail/use/mbox\_access}}(\mathit{user} \in GrpMail)$. Moreover, an example for a degradation dependency among degradation types is $gt_{highlinkutilization} \rightarrow gt_{highmailsendingdelay}$, which has $gt_{highmailsendingdelay}$ for $f_{\mathrm{mail/use/send}}$ as target.

Table 4.17 gives a summary of the DegDep specification with SIDepMod(QoX/Subj:X) for a given subject specification notion X from dependency model SIDepMod(Subj:X). Additionally, Table 6.10 on p. 377 in Appendix A introduces a set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:QoX/Subj:X).

SIDepMod(QoX/Subj:X)	
reusing SIDepMod(Subj:X) for some subject specification type X	
types of dependent degradations:	degradations described mainly as QoX degradations, i.e., by degradation subjects (of type X), and by degradation manner (set of affected QoX parameters);
	secondly described by additional information (other than subject or manner, compare Table 4.7);
associations of dependent degradations:	degradation dependencies fully determined/derived from the dependencies of their respective degradation subject/manner;
	multiplicities left open;
additional dependency constraints:	none

Table 4.17: Overview of DegDep specification for SIDepMod(QoX/Subj:X) (compare Table 4.5 and Fig. 4.80)

notion of QoX degradation instantiations

For SIDepMod(QoX) the notion of *degradation type* (or *degradation class*) was introduced as a class for actually specifying degradation manner, describing the affected QoX parameter (set). But for many degradation dependencies the specification of degradation manner alone is not enough, rather the actual value accuracy potentially combined with degradation time (e.g., as function time) has to be included in the specifications of dependent degradations. These degradation specification aspects (value accuracy combined with degradation time) are typically specified by the QoX parameter values (per time) or at least some abstraction thereof, e.g., potential value ranges, value distributions. QoX parameter values (per time) or abstractions thereof can be regarded as specific instantiations of a degradation type, e.g., for the degradation type *mail sending delay* of subject $f_{\mathrm{mail/use/send}}$, the actual degradation type instantiation specification (QoX value accuracy): $> 50\ \mathrm{min}$ permanently. Therefore, specification of degradation value accuracy (potentially combined with degradation time), as e.g., actual QoX parameter values or abstractions thereof, are subsumed under the generic term *QoX degradation instantiation*, and regarded as particular instantiations (or instantiation subsets) of a degradation type. Similarly, the degradation aspects degradation value accuracy and degradation time are subsumed under the generic term *degradation degree* of a degradation. Using this new terminology, a particular QoX degradation instantiation (set) specifies degradation degree of a degradation. The QoX degradation instantiation (set) is actual an instantiation (set) of a particular degradation type, which in turn specifies degradation manner. So, on an abstract level, degradation degree can be regarded as instantiation of degradation manner. Fig. 4.81 illustrates the concept of QoX degradation instantiations, as well as the relationship to degradation degree, degradation type and degradation manner. Degradation type is regarded as a meta class, so that its instances (particular degradation types) have own instances (QoX degradation instantiations). For actually specifying QoX degradation instantiations

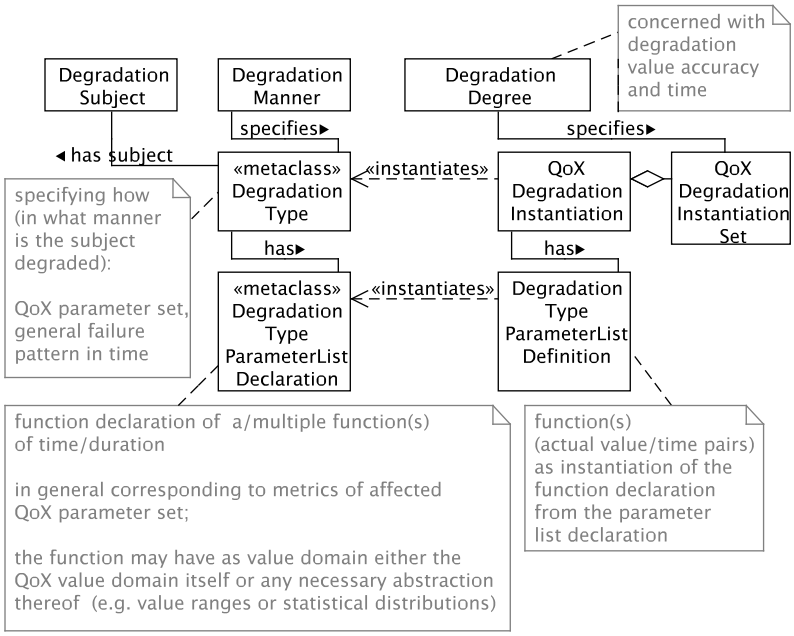

Figure 4.81: QoX degradation instantiations as instantiating degradation types

as instances of a particular degradation type, *degradation type parameter list declarations* are defined for each particular degradation type. These declarations are a list of named parameter type declarations. Actual QoX degradation instantiations have a corresponding *degradation type parameter list definition* consisting of parameter definitions (name/value pairs) conforming to the parameter type declarations of their degradation degree. This concept used for the instantiation of degradation types here is actually very similar to the instantiation of subject specification classes used in Sect. 4.3.7 for the refinement of SIDepMod(Subj:X) to SIDepMod(Subj:X/Inst). The parameter list declarations (and corresponding definitions) for degradation type usually comprise one or multiple parameters for describing the actual QoX values (per time) or any abstraction thereof (see discussion above) in order to specify the actual degradation degree.

These are possible alternative examples for degradation type parameter declarations, e.g., for the above mentioned degradation type high link utilization of $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$: the current metric value (range), current metric value and range of past metric values, current metric value with an estimation about how long it will remain in this state, a probability/possibility distribution describing how the metric will (in estimation) evolve in the near future, a combination of the former ones. Any such declaration is used to specify and differentiate QoX metric values directly or by an appropriate abstraction thereof, potentially per time/duration. In general, the choice for the parameter declaration has to be appropriate and accurate enough for specifying the quality relationships and temporal relationships, as far as necessary for the degradation dependency specification which will be described below. Correspondingly, QoX degradation instantiations are specified by concrete values

assigned to the declared parameters, e.g., value $> 60\%$ permanently in the near future for the high link utilization.

SIDepMod (QoXInst)

Having introduced the notion of QoX degradation instantiations instances of a degradation type, now the SIDepMod(QoX/Subj:X) introduced above can be refined into SIDepMod(QoXInst/Subj:X). Fig. 4.82 illustrates the classes used for the specification of degradations and degradation dependencies by SIDepMod(QoXInst/Subj:X) (compare Fig. 4.80). The actual refinement of SIDep-

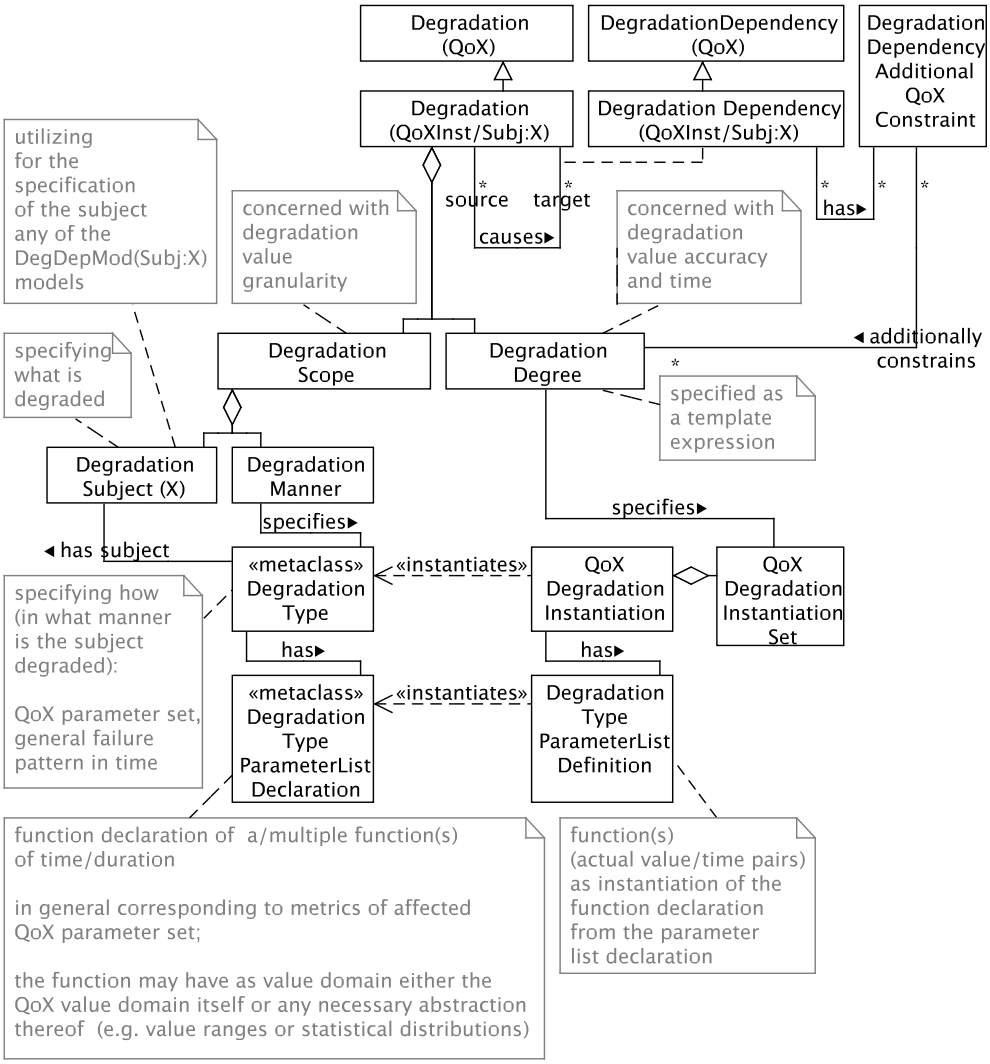

Figure 4.82: SIDepMod(QoXInst/Subj:X): considering QoX degradations instantiations, refining Fig. 4.80, reusing SIDepMod(Subj:X) for some X

Mod(QoX/Subj:X) to SIDepMod(QoXInst/Subj:X) is similar to the refinement of SIDepMod(Subj:X) to SIDepMod(Subj:X/Inst) in Sect. 4.3.7. Here not the subject, but instead the degradation type is instantiated in the manner as already described above (Fig. 4.81). The degradation subject specification is retained from SIDepMod(QoX/Subj:X), depending on the chosen X. Similar as in SIDepMod(QoX/Subj:X), the degradation type has to conform to this subject specification.

The degradation degree is specified in SIDepMod(QoXInst/Subj:X) as a set of QoX degradation instantiations described by a template expression. So, a degradation in SIDepMod(QoXInst/Subj:X) being source or target in a degradation dependency is described by a degradation subject specification (of type X), a conforming degradation type, and a degradation degree specified as template expression. These template expressions of dependent degradations can be additionally restricted by constraints over their common variables. So complex quality and temporal relationship in a degradation dependency can be expressed. This is again similar to the approach used for refining SIDepMod(Subj:X) to SIDepMod(Subj:X/Inst) (Sect. 4.3.7). Concerning additional dependency constraints compare especially Fig. 4.52 on p. 218.

examples

Two particular examples for degradation dependencies taking into account degradation type as well as the specific degradation degree are illustrated in the following. The first example continues the above mentioned dependency $gt_{highlinkutilization} \rightarrow gt_{highmailsendingdelay}$, the second one is concerned with the dependency between $gt_{highdnsdelay}$ and $gt_{highmailsendingdelay}$:

As already introduced above, the dependency $gt_{highlinkutilization} \rightarrow gt_{highmailsendingdelay}$ (refined specification of $g_{r1} \rightarrow g_{s1-1/1}$ in Fig. 4.6 on p. 131, with $g_{s1-1/1}$ representing the sole effect of g_{r1} without g_{r1b}) describes that a degradation of the IP link $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{r\_sw\_2}})$ (degradation subject), namely a high link utilization (degradation type $gt_{highlinkutilization}$), entails a degradation of the mail sending functionality (degradation subject $f_{\mathrm{mail/use/send}}$), namely a high mail sending delay (degradation type $gt_{highmailsendingdelay}$). This relationship of mail sending delay from link utilization is also further related to additional, intermediary context factors, e.g., the actual ratio of mail traffic to other IP traffic, and the actual amount of current mail sending requests. So, in order to take actually into account degradation degree for both dependent degradations, i.e., specific QoX degradation instantiations and their inter-relationship, these additional factors have to be considered, too: On the one hand specific, current value (ranges) (over time) have to be specified for the link utilization values as well as the mail sending delay itself, including the deviation from their normal levels. On the other hand, to actually describe their inter-relationship, information about the additional, intermediary context factors (intermediary QoX context, as part of the QoX degradation instantiation specification) is necessary: e.g., the current/-future estimated value specification (over time) of the ratio of mail sending traffic to other IP traffic and of the amount of mail sending requests. Using current values for these intermediary factors, an estimation for the QoX values of mail sending delay can be derived from the QoX values of the link utilization. Such an estimation may be supported by comparing historic measurements of all inter-related QoX values. To sum it up, a refined specification of $gt_{highlinkutilization} \rightarrow gt_{highmailsendingdelay}$ by taking into account QoX value instantiations can be done as shown in Table 4.18.

This example also shows how a QoX degradation instantiation is determined by multiple degradation type parameters (compare Fig. 4.81), e.g., in the case

$g_{r1} \rightarrow g_{s1-1/1}$, with
$g_{r1} \equiv g_{high\_iplink\_util} := degradation($
subject: $r_{\mathrm{iplink}(\mathrm{r_{rt\_lrz}},\mathrm{r_{sw_2}})}$,
manner: $gt_{highlinkutilization}$,
avg. link util: $> 60\%$,
intermediary context: avg. ratio mail traffic/other traffic: ca. 30%,
intermediary context: avg. mail requests/min: 200,
) and
$g_{s1-1/1} \equiv g_{high\_mail\_sending\_delay/1} := degradation($
subject: $f_{\mathrm{mail/use/send}}$,
manner: $gt_{highmailsendingdelay}$,
avg. mail sending delay rise: 2.5 min,
).

Table 4.18: Specfiction of a degradation dependency in SIDepMod(QoXInst)

of $g_{high\_iplink\_util}$ the parameters *avg. link util*, *avg. ratio mail traffic/other traffic*, and *avg. mail requests/min*. The specific relationships between the values/value ranges of all these degradation type parameters for both degradations are restricted by appropriate dependency constraints.

The second example is the degradation dependency $gt_{highdnsdelay} \rightarrow gt_{highmailsendingdelay}$ (refined specification of $g_{r1} \rightarrow g_{s1-1/2}$ in Fig. 4.6 on p. 131, with $g_{s1-1/2}$ representing the sole effect of g_{r1b} without g_{r1}) which specifies the relationship between DNS request delay (particular degradation type $gt_{highDNSdelay}$ for degradation subject $r_{\mathrm{dns\_sv1}}$) and further aggravation of the mail sending delay (particular degradation type $gt_{highmailsendingdelay}$ of degradation subject $f_{\mathrm{mail/use/send}}$). Similar to the first example for actually taking into account QoX value instantiations for both degradations and their inter-relationship, some additional, intermediary context information has to be considered, although it is of another type as in the first example: Actually for sending a single mail ($f_{\mathrm{mail/use/send}}$) multiple DNS requests are necessary in order to resolve all mail domain names involved with the mail to be sent, at least two, one for the sender domain, and one for a single receiver domain (nevertheless of course the sender domain normally is part of the local domain of the LRZ). That is why an aggravation of the DNS request delay entails a respectively multiplied (at least doubled) aggravation of the mail sending delay. This relationship may be represented by a respective weighting factor (multiplier) for the DNS delay. Consequently, the degradation dependency may be as a first step more precisely specified as $gt_{highdnsdelay} \rightarrow_{\times 2} gt_{highmailsendingdelay}$ Actually taking into account actual QoX value instantiations, it can be specified in as shown in Table 4.19.

Table 4.20 gives a summary of the DegDep specification with SIDepMod(QoXInst/Subj:X) for a given subject specification dependency model SIDepMod(Subj:X). Table 6.11 on p. 378 in Appendix A introduces a set-

$g_{r1b} \rightarrow g_{s1-1/2}$, with
$g_{r1b} \equiv g_{high\_dns\_delay} := degradation($
subject: $r_{\mathrm{dns\_sv1}}$,
manner: $gt_{highdnsdelay}$,
avg. request delay: 15 s,
additional context: min. number of dns request / mail sending: 2,
) and
$g_{s1-1/2} \equiv g_{high\_mail\_sending\_delay/2} := degradation($
subject: $f_{\mathrm{mail/use/send}}$,
manner: $gt_{highmailsendingdelay}$,
avg. mail sending delay rise: 30 s,
).

Table 4.19: Specfiction of a degradation dependency in SIDepMod(QoXInst) (second example)

theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Subj:QoXInst/Subj:X).

SIDepMod(QoXInst) is already very elaborated and in this SI dependency model most degradation dependencies can be expressed in appropriate granularity. However, dynamics of degradation dependencies, and related to this redundancy and load-balancing (cooperation patterns of degradation subjects) cannot explicitly be specified. These issues are tackled in the following two sections. limitations

4.3.9 SIDepMod(Coop): SI dependency model with dynamics at a time instance

Here dynamics of degradation dependencies are investigated and classified into two types of dynamics. For the first type, a refined SI dependency model, SIDepMod(Coop) is introduced, which allows to express any such dynamics as far as is it necessary for appropriate degradation dependency specification. The second type of dynamics is treated in the following section.

In the following, the general term dynamics of degradation dependencies is analyzed by looking at three different examples based on the example scenario of Sect. 2.3. The aspect of dynamics for degradation dependencies is related to the requirement R1.4 (interacting/cooperating subjects of multiple source degradations, very short-term) and R2.1 (dynamics in general, i.e., change over time, more long-term).

On the one hand, in general any degradation dependency, even if having only one source degradation and one target degradation may be subject to dynamics, i.e., some of its aspects such as the two dependent degradations may change in some way. On the other hand, there are especially degradation de-

SIDepMod(QoXInst/Subj:X)	
reusing SIDepMod(Subj:X) for some subject specification type X	
types of dependent degradations:	degradations described as QoX degradation instantiations, i.e., by degradation subjects (of type X), and by sets of QoX parameter metric values, or abstractions thereof (value ranges, distributions)
	$\Longrightarrow$ degradation value accuracy/degradation time explicitly expressible;
associations of dependent degradations:	dependencies between sets of QoX degradation instantiations:
	complex relationships concerning degradation value accuracy/degradation time of dependent degradations are expressible;
additional dependency constraints:	possibility to constrain the dependent sets of QoX degradation instantiations by the use of templates with variables;

Table 4.20: Overview of DegDep specification for SIDepMod(QoXInst/Subj:X) (compare Table 4.5 and Fig. 4.82)

pendencies with multiple source degradations, for which it has to be specified how these multiple sources interact to result in the target degradation(s).

Dynamics of degradation dependencies are concerned with the change of dependencies over a longer period of time, to the redundant replacement of degradation subjects in a relatively short time interval, as well as to the simultaneous interaction/collaboration of the subject of multiple source degradations, e.g., for performance (by load-balancing) and reliability (by redundancy) reasons, or even a combination of two or all of the three aspects. Specifically, the latter aspect is also concerned with degradation dependencies which are specifically related to each other, i.e., which should be combined into a single degradation dependency with multiple sources, so it can be processed together by impact analysis.

examples of dynamics of dependencies

For introduction and illustration three different examples concerning dynamics of dependencies over time are explained and resulting issues are discussed.

dynamics example 1

The first example are the 10 load-balanced web servers $r_{\text{websv(x)}}, x = 1, \ldots, 10$ of the web hosting service, e.g., appearing in the dependency $r_{\text{websv(x)}}(\textit{configuration} = \text{normal}) \rightarrow f_{\text{web/use/apage}}(\textit{customer} = \text{TUM})$. In fact, the load-balancing, is performed by a load-balancing switch r_{lbswitch} located in the IP service (described by $f_{\text{ip/use/load\_balance}} \rightarrow f_{\text{web/use/apage}}$). The load-balancing algorithm of r_{lbswitch} is using a least-resource-usage distribution method. Furthermore, if r_{lbswitch} detects that one of the web servers is not working correctly anymore, it avoids considering this one for dispatching WWW queries to it. This way, also reliability is ensured to some degree, far as r_{lbswitch} is able to identify incorrect operation of a web server correctly and

timely. So, this example includes load-balancing for performance reasons, as well as (to some degree) redundancy issues for reliability reasons.

dynamics example 2

The second example are two mail relay servers of the mail example service, which are load-balanced by DNS round-robin method. Hence the distribution-method here is round-robin, and also performed by the help of a subservice, namely DNS service (described by dependency $f_{\mathrm{dns/use}}, r_{\mathrm{mailrelay1}}, r_{\mathrm{mailrelay2}} \rightarrow f_{\mathrm{mail/use/recv}}$). Reliability is not ensured here, as in case of unavailability of one mail relay server, the DNS subservice (unaware of this failure) will further resolve respective DNS queries to the IP addresses of both servers in a round-robin manner.

dynamics example 3

Third, located in the IP service, there are 2 load-balanced NAT (network address translation) servers r_{natsv1} and r_{natsv2}, which filter and check IP traffic from mobile and VPN (Virtual Private Network) users for security risks (described by the dependency $r_{\mathrm{natsv1}}, r_{\mathrm{natsv2}} \rightarrow f_{\mathrm{ip/use/con/nat}}$). In normal operation, each of the two NAT servers are responsible for distinct IP address spaces of different users. That is, the distribution-method of their load-balancing is client-hash-based. But in case of a failure of one of them, the one left working dynamically takes over the responsibility for the IP address spaces of the failing one. Thus, in this example load-balancing (for performance reasons) and redundancy (for reliability reasons) are appearing together. The hash-based load-balancing, as well as the dynamic redundant take-over in case of failure are performed by both resources cooperatively without further intervention of another resource or subservice.

dynamics example 4

Another general example is the case of two load-balanced servers, which are both simultaneously active only in specific periods of the day, while for the rest of day, only one is active e.g., because of high operating costs. A specific example scenario for this case might be a telecommunication service, which has its peak-off hours around noon and after 18:00 o'clock a day (described by $r_{\mathrm{sv1}}, r_{\mathrm{sv2}} \rightarrow f_{\mathrm{tk\_sv/use}}(\textit{time}$ = 10:00-14:00 or 18:00-6:00$)$, $r_{\mathrm{sv1}} \rightarrow f_{\mathrm{tk\_sv/use}}(\textit{time}$ = 6:00-10:00 or 14:00-18:00$)$). So the dynamics involved here, i.e., the using of two servers in specific time periods, are concerned with performance reasons.

dynamics example 5

A further example of redundancy is a redundant resource which can automatically replace another normally working resource when this one is broken (hot-standby).

classification of dynamics by reason and method

Consequently, these different examples show mainly two reasons for dynamics of dependencies: On the one hand there is the reason of *performance* accomplished by load-balancing multiple components using various distribution-methods, and on the other hand there is the reason of *reliability* accomplished by providing redundancy using different switching methods in different time scale.

Concerning redundancy switching further details of its realization may be important for the specification of degradation dependencies: Redundant switching may be accomplished either by joint, direct cooperation of the redundant

realization details of redundant switching

components, or by coordination via a particular coordination component, or by human interaction by hand. Furthermore the time scale is of interest, i.e., the granularity of the time units used to specify times of different switching states, e.g., in the order of magnitude of seconds (fully automated), minutes (automated or human interaction), hours/days (human interaction).

Often all this information is useful and necessary for degradation dependency specification, as illustrated by the following examples of possible degradations:

The first example is the unavailability of one of the web servers, which even after the detection by the load-balancing-switch entails a possible performance degradation as only 9 servers are left working sharing the load.

In a second example, with the response time of one web server becoming very high (but left in allowed range for load-balance-detection), the response time of a part of the WWW queries is rising in an apparently random-manner (for all requests which are dispatched to the slow web server).

Third, the unavailability of one of the mail relay servers results in the mail sending functionality (apparently) randomly failing.

Moreover, one of the NAT servers becoming unavailable results on the one hand in a failure of all its active connections, and on the other hand in a (possible) reduced performance of the second NAT server, after this one has detected the outage of the first and thenceforward is responsible for serving all requests.

So these degradation examples related to dynamics of dependencies illustrate, that for a detailed, useful, appropriate degradation dependency specification it is often necessary to specify all this dynamics-related information about a dependency.

general properties of dynamics of dependencies

In order to allow for a generic classification and modeling of all necessary aspects related to dynamics of dependencies, from the examples above some general properties related to dependency dynamics suitable for the specification of degradation dependencies are identified. These properties are more general than the issues of dynamics of dependencies discussed above, as e.g., the purpose (reliability, performance) or the realization method (load-balancing by various specific distribution methods, static redundancy, dynamic redundancy). Fig. 4.83 gives an overview of these aspects of dynamics for degradation dependencies.

As a first aspect of dynamics, the dependency may be in use (or active) only in various instances of time or in various time intervals. Furthermore, the sources or targets may change over time (e.g., the example of the two servers of the telecommunication service, which are simultaneously active only in defined time intervals a day).

In case of multiple sources being involved further aspects of dynamics arise: For multiple sources, there is a specific *cooperation pattern*, which describes how the source degradations are combined or how their particular subject in-

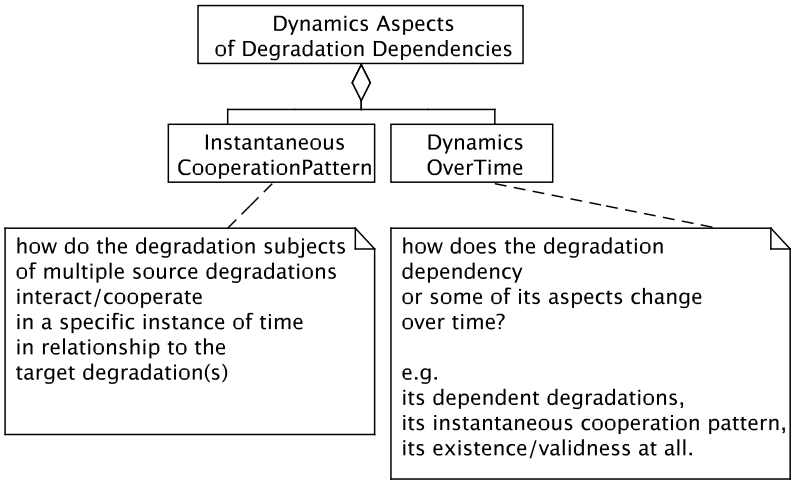

Figure 4.83: Overview of dynamics aspects of degradation dependencies

teract/cooperate in order to result in the target degradation(s). Concerning degradation dependency specification, this cooperation pattern can be further divided up in two different components: its *instantaneous cooperation pattern*, i.e., the interaction/cooperation of multiple sources at a specific instance of time, or its dynamics at a specific instance of time, as well as its *change of dynamics over time*:

In a specific instance of time, the instantaneous cooperation pattern specifies how the sources interact in the specific instance of time in order to result in the target, e.g., by load-balancing using a specific distribution-method. Whereas, the dynamics over time comprise changes of the state of the degradation dependency from one instance of time to another, i.e., a possible change of its sources, its targets, or even its instantaneous cooperation pattern, or the complete validness of the dependency.

The reason for the separation is that for degradation dependency specification it is necessary to know the state of a dependency at a specific time instance (sources, targets, instantaneous cooperation pattern, activeness) as well as how this state changes over time. For instance, in order to appropriately derive target degradations for one of the examples above concerned with some load-balancing mainly the aspect of instantaneous cooperation pattern is important. For the examples concerned with redundancy by switching the aspect of the change of state over time is more important. For some examples, both aspects are relevant.

Additionally, it has to be mentioned, that a time dimension has to be defined which is appropriate for the actual course of the impact analysis, which allows to distinguish between different time instances with a suitable granularity.

To sum it up, concerning the modeling of dynamics of degradation dependencies, three general aspects have to be taken into account: An appropriate time dimension has to be defined with a time granularity appropriate for performing the steps of impact analysis. In a specific instance of time (related to the

defined time dimension), the current state of the degradation dependency has to described. The current state comprises the question whether it is currently active or not, its current source and target degradation (subjects), as well as the specific instantaneous cooperation pattern between the sources. The state may be change over time, i.e., may be different for different time instances (according to the defined time dimension). Namely, in general, any aspect of the current state may be subject to change, i.e., the question whether currently active or not, the sources, the targets, or the instantaneous cooperation pattern. In the following, the change of the dependency over time is also described by the term *dynamics over time*, whereas the term instantaneous cooperation pattern describes the *dynamics at a specific time*.

abstract specification of dynamics of dependencies

After having identified these three general aspects of dynamics of dependencies, their abstract specification is discussed:

Starting with the time dimension, time instances can be generically specified, as a parameter t ranging over time values of time set $Time$. Based on this, the different components of the state of a dependency d at a specific instance t can be described by functions of this parameter t:

- whether the dependency is active or not at time t can be decided by function $isActive(d, t)$
- its related degradations (e.g., specified as classes or more refined their instantiations), can be determined by two functions $degr\_src(d, t)$ and $degr\_dst(d, t)$ respectively
- furthermore, the instantaneous cooperation pattern (of the related degradations or mainly their subjects) can be generically described by a function $inst\_coop\_pat(d, t)$.

These introduced functions allow to specify the general dynamics aspects a given dependency d at a specific time instance t. Furthermore, with the parameter t being varied over $Time$, they also specify the change over time of the dynamics.

Of course, the subject on the type of function $inst\_coop\_pat$, i.e., the explicit specification of the instantaneous cooperation pattern, is still to be investigated:

In general, information that a dependency is valid at some time describes that a degradation at that time of one or some of the sources may cause degradations of the targets at that time or in a time near to that time.

The using of information about the instantaneous cooperation pattern should allow for a more explicit and refined determination of this possibility of a degradation propagation and the explicit specification of the relationship of the time of the source degradations and the time of the target degradations.

A time specification for the validity of a dependency might describe a specific time instance t or a time period $[t_1, t_2]$ of the time dimension introduced

above. Subsuming both cases, such time specification will be written by the Greek letter τ in the following.

The time specification for a source degradation may not be exactly the same as the time specification of the entailed degradation: for instance, network subservice being unavailable for a minute, causing an application service based on it being unavailable for 1 hour, because of active connections between resources or to subservices being broken and not easily recovered automatically. But in any case, the time specification of target degradations, is related to the time specification of source degradations. For a time specification τ_{src} (representing t or $[t_1, t_2]$) of some source degradations the time specification τ_{dst} of target degradations may describe e.g., $[t, t + 1\ \mathrm{min}]$, or $[t, t + 60\ \mathrm{min}]$, $[t_1, t_2 + 1\ \mathrm{min}]$, $[t_1, t_2 + 60\ \mathrm{min}]$, i.e., a time instance or time range near to τ_{src}.

Consequently, the specification of the instantaneous cooperation pattern should allow for derivation of degradations from some sources at time specification τ_{src} to degradations of the targets of the dependency at some time specification τ_{dst} related to τ_{src}. If the explicit time specifications are not relevant, e.g., if they are the same, they may be left out of the specification. But it should be possible by using the instantaneous cooperation pattern to derive such more complex time relationships between source degradations and target degradations if it is necessary for degradation dependency specification.

Some possibilities for this specification might be either UML activity diagrams and UML collaboration diagrams, petri nets, or specifications by logical formulas, e.g., using temporal logics (p. 105), or even only propositional logical formulas with logical connectors like OR, AND, XOR, being potentially combined with the use of probability/possibility distributions.

But speaking generally, the instantaneous cooperation pattern is specifiable by some additional constraint expression on the degradation specifications of the dependent degradations of a degradation dependency, independent of the building blocks, functions, operator, or algorithms the this constraint expression is made or evaluated by (compare especially Fig. 4.52 on p. 218 concerning additional dependency constraints).

For example, concerning the availability of particular functionalities:

general examples for cooperation patterns specified by attachments to the dependency specification

- refine $f_{\mathrm{dns/use}}, r_{\mathrm{mailrelay1}}, r_{\mathrm{mailrelay2}} \rightarrow f_{\mathrm{mail/use/receive}}$ to:

  $f_{\mathrm{dns/use}} \rightarrow_{always} f_{\mathrm{mail/use/receive}}$,
  $r_{\mathrm{mailrelay1}} \rightarrow_{possibility=0.5} f_{\mathrm{mail/use/receive}}$,
  $r_{\mathrm{mailrelay2}} \rightarrow_{possibility=0.5} f_{\mathrm{mail/use/receive}}$,
  which have to be processed together.

- refine $f_{\mathrm{ip/use/load\_balance}} \rightarrow f_{\mathrm{web/use/apage}}$, and
  $r_{\mathrm{websv(x)}}(\mathit{conf} = \mathrm{normal}) \rightarrow f_{\mathrm{web/use/apage}}, (x = 1, \ldots, 10)$ to:

  $f_{\mathrm{ip/load\_balance}} \rightarrow_{always} f_{\mathrm{web/use/apage}}$
  $r_{\mathrm{websv(x)}}(\mathit{conf} = \mathrm{normal}) \rightarrow_{\substack{before\_detection\_or\_if\_already\_in\_progress, \\ possibility = 1/10}} f_{\mathrm{web/use/apage}}$,
  which have to be processed together.

- even for a dependency with only one source:
  refine $f_{\text{ip/use}} \rightarrow f_{\text{mail/use/receive}}$ to:
  $f_{\text{ip/use}} \rightarrow_{always} f_{\text{mail/use/receive}}$.

As these examples show, the information concerning the instantaneous cooperation pattern used for derivation of entailed degradations is specific to a certain type of degradation (degradation manner).

dependency specification with cooperation pattern

In the following the actual degradation dependency specification concerning the instantaneous cooperation pattern is discussed in more detail. As introduced above - the instantaneous cooperation pattern is concerned with dynamics at a specific instance of time, specifically the interaction of multiple source degradation (subjects), i.e., in the case of source multiplicity of the degradation dependency > 1 (so-called composite dependency). Speaking generally, the instantaneous cooperation pattern can be specified by additional constraints on the specifications of the source and target degradations of the dependency.

The SI dependency model SIDepMod(QoXInst) developed in the previous section has to some degree the power to specify (at least simple) instantaneous cooperation pattern, depending on the complexity needed, by using constraints over the degradation type degree template expressions of dependent degradations. If the number of interacting source degradations is fixed (and relatively small), constraints on the degradation type degree template expressions are enough to express the cooperation pattern as a temporal/quality relationship of the dependent degradations.

example using SIDep-Mod(QoXInst)

For example, in the case of the the 10 load-balanced web servers $r_{\text{websv(x)}}, x \in \{1 \ldots 10\}$, which are used for realizing $f_{\text{mail/use/apage/static}}$ (compare Sect. 2.3.2), a diminution of the overall web page throughput of $f_{\text{mail/use/apage/static}}$ (for all customers) is determined by the aggregation of the individual diminution of all particular web servers, as specified in Table 4.21 using SIDepMod(QoXInst).

$\{g_{r\_websv\_throughput(x)} | x \in \{1 \ldots 10\}\} \rightarrow g_{f\_webpage\_static\_throughput}$,
with
$g_{r\_websv\_throughput(x)} := degradation($
subject: $r_{\text{websv(x)}}$,
manner: $gt_{low\_throughput}$,
avg. web page throughput diminution: $T_{r(x)}$,
) for $x = 1 \ldots 10$, and
$g_{f\_webpage\_static\_throughput} := degradation($
subject: $f_{\text{mail/use/apage/static}}$,
manner: $gt_{low\_throughput}$,
avg. web page throughput diminution: T_s,
), with (QoX degradation) dependency constraint $T_s \equiv \sum_{x=1 \ldots 10} T_{r(x)}$

Table 4.21: Specification of the instantaneous cooperation pattern for a degradation dependency in SIDepMod(QoXInst)

The aggregation expression in the constraint may also be more complex than only summing up the individual diminution values. But the constraint is only concerned with the restriction of the values of degradation type parameters (*avg. web page throughput diminution*), i.e., with restriction of the QoX value instantiations, not interfering with restrictions on the degradation subjects.

But, if the number of the interacting source degradations is dynamic (and potentially high) joint constraints on the degradation degree template expressions and constraints on the subjects are necessary. For this reason, a new SI dependency model, SIDepMod(Coop), more elaborated than SIDepMod(QoXInst) is discussed now. Similar as SIDepMod(QoXInst), SIDepMod(Coop)

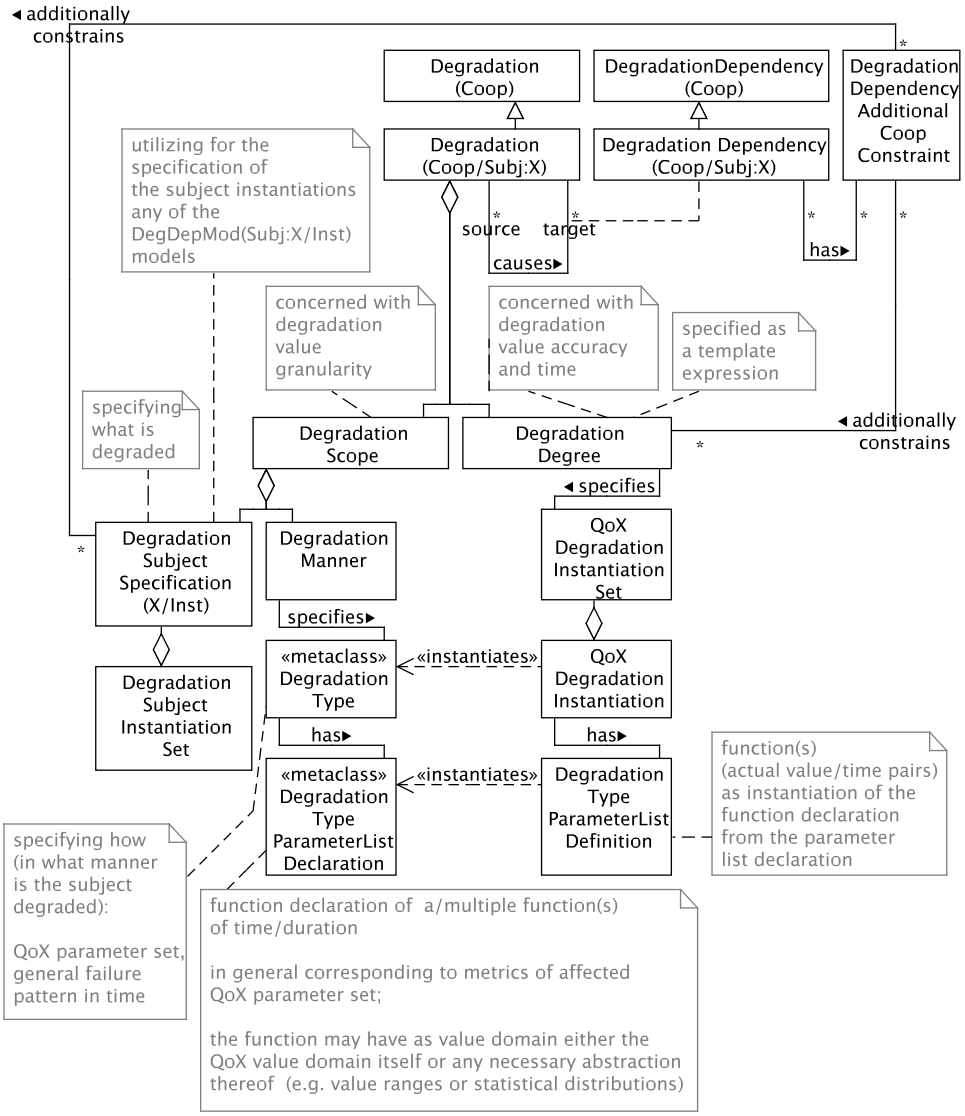

Figure 4.84: SIDepMod(Coop/Subj:X): considering cooperating QoX degradation instantiations , refining Fig. 4.55, reusing SIDepMod(QoX/Subj:X) for some X

the class structure used for the specification of degradations and degradation dependencies by SIDepMod(Coop/Subj:X) (compare Fig. 4.55). Actually not any subject specification X is allowed. As constraints on subjects (jointly on degradation degree) have to be possible, actually only a subject specification notion X is allowed which includes subject instantiation. I.e., actually only a refinement the abstract SIDepMod(Subj:X/Inst) (see Fig. 4.71 on p. 251) is allowed for reuse, e.g., SIDepMod(Subj:Fcty/Inst) (Fig. 4.72 on p. 253), as these models allow constraints over subject specifications. Other than this restriction, SIDepMod(Coop), or more specifically SIDepMod(Coop/Subj:X/Inst) depending on the used X/Inst, is similarly structured as SIDepMod(QoXInst) (compare Fig. 4.82 on p. 270). The sole further structural difference is that in SIDepMod(Coop), the additional dependency constraints restrict both, the subject instantiation template specification, as well as the degradation degree template specification, jointly by the use of common variables in both template specifications.

example using SIDepMod(Coop)

In the case of the the 10 load-balanced web servers $r_{\text{websv(x)}}, x \in \{1 \ldots 10\}$ for realizing $f_{\text{mail/use/apage/static}}$ (compare Sect. 2.3.2), a diminution of the overall web page throughput of $f_{\text{mail/use/apage/static}}$ for a particular customer C_1 is depending on a specific aggregation of the individual diminution of all web servers which are currently used for accessing the particular web pages of this customer (denoted by $WebPageList(C_1)$), as specified in Table 4.22 using the newly developed SIDepMod(Coop).

$\{g_{r\_websv\_throughput(x)}(webpage \in WebPageList(C_1))|x \in \{1 \ldots 10\}\} \rightarrow g_{f\_webpage\_static\_throughput}(customer = C_1)$,
with
$g_{r\_websv\_throughput(x)}(webpage \in WebPageList(C_1)) := degradation($
subject: $r_{\text{websv(x)}}$(*web page* $\in WebPageList(C_1)$),
manner: $gt_{low\_throughput}$,
avg. web page throughput diminution: $T_{r(x)}$,
) for $x = 1 \ldots 10$, and
$g_{f\_webpage\_static\_throughput}(customer = C_1) := degradation($
subject: $f_{\text{mail/use/apage/static}}$(*customer* = C_1),
manner: $gt_{low\_throughput}$,
avg. web page throughput diminution: T_s,
), with (Coop degradation) dependency constraints
$T_s \equiv \sum_{x \in IndexSet} T_{r(x)}$, $IndexSet \subset \{1, \ldots, 10\}$, and
$\forall_{x \in IndexSet} WebPageList(C_1) \cap WebPageRealizationList(r_{\text{websv(x)}}) \neq \emptyset$

Table 4.22: Specification of a more complex instantaneous cooperation pattern for a degradation dependency in SIDepMod(Coop)

The particular load-balancing method used here is hash-based, i.e., each web server currently serves a particular subset of all web pages, the current subset denoted by $WebPageRealizationList(r_{\text{websv(x)}})$. The aggregation expression in the constraint above may also be more complex than only summing up the individual diminution values of each relevant web server. But never-

theless, in the example above, the constraints above are concerned with both jointly, the restriction of the values of the degradation type parameters (*avg. web page throughput diminution*) as well as the restriction of the degradation subject: The variable terms C_1 as well as $r_{\text{websv(x)}}$, which are part of the respective degradation subject specifications, are used in the constraint expressions to express the particular, current inter-relationship with the QoX value instantiations (compare the simpler example above with constraints not interfering with the restriction of the degradation subjects).

SIDepMod(Coop/Subj:X)	
reusing SIDepMod(QoX/Subj:X) for some subject specification type X	
types of dependent degradations:	degradations described by templates specifying the degradation subject instantiations and their respective QoX degradation instantiation at once;
associations of dependent degradations:	dependencies between sets of interacting QoX degradation instantiations:
	complex relationships concerning degradation subject/manner, degradation value accuracy/degradation time of dependent degradations are expressible;
additional dependency constraints:	constraints over the dependent sets of interacting QoX degradation instantiations by the use of templates with variables;

Table 4.23: Overview of DegDep specification for SIDepMod(Coop/Subj:X) (compare Table 4.5 and Fig. 4.84)

summary

Table 4.23 gives a summary of the DegDep specification with SIDepMod(Coop/Subj:X) which is resulting SIDepMod(QoX/Subj:X) for a given subject specification dependency model SIDepMod(Subj:X). Table 6.12 on p. 379 in Appendix A introduces a set-theoretic, formal notation for degradations and degradation dependencies of SIDepMod(Coop/Subj:X).

In the next section the specification of dynamics over time as the second type of degradation dependency dynamics are covered, which are the last issue to cover.

4.3.10 SIDepMod(DynOT): SI dependency model for dynamics over time

In the previous section dynamics of degradation dependencies were analyzed, and two general cases of dynamics have been identified: dynamics at a time instance determining a specific cooperation pattern of multiple source degradations, as well as dynamics of degradation dependencies over a longer time period. Moreover, abstract specification notions for both types of dynamics were introduced. The degradation dependency specification of the for-

mer type was also discussed in the previous section. Here the degradation dependency specification of the latter one is treated based on the already introduced abstract specification notions. An abstract SI dependency model SIDepMod(DynOT) is introduced which basically discusses how to cover dynamics over time for degradation dependency specification.

Dynamics over a time is concerned with changes of the degradation dependency or some of its aspects over a longer time period. This issue is related to the requirement R2.1. As parts or aspects of the degradation dependency which can change over time, two basic cases were already identified in the previous section: First, the dependency as a whole may be valid or existing only for specific time intervals, i.e., its whole state of validness or existence can change over time. Second, any of its aspects, namely its dependent degradations or their instantaneous cooperation pattern (see previous section) may change.

But for the introduction of a generic SI dependency model, the explicit differentiation of particular dependency aspects is not necessary, as these actually depend on the used SI dependency model which is used before considering dynamics over time: The general idea for the design of a SI dependency model covering dynamics over time is to reuse any prior existing SI dependency model SIDepMod(Y) and extend it to a SI dependency model SIDepMod(DynOT/Y). The extended model has to allow to cover any of the both cases described above: change over time of the existence of a degradation dependency at all, as well as change over time of any of the information parts used to specify a degradation dependency in SIDepMod(Y). Change over time in general can be specified by a function of time/duration with appropriate time granularity, as discussed in the last section.

That is, for SIDepMod(DynOT/Y) the following extensions have to be made: For each degradation dependency, a function of time $is\_valid(t)$. For any existing information part $DegDepSpecPart$ used for degradation dependency specification in SIDepMod(Y) introduce a function $DegDepSpecPart(t)$. These functions of time used above may be given each explicitly, or by template expressions, which can jointly be constrained for each degradation dependency (compare list given on p. 278). For a more concrete example for applying SIDepMod(DynOT/Y), see example 4 on p. 275.

Table 4.24 summarizes the details of the abstract DegDep specification with SIDepMod(DynOT) which is reusing any prior existing SI dependency model SIDepMod(Y) and extending it for dynamics over time.

Concluding the complete design of SI dependency models (Sect. 4.3.2 to 4.3.10), it can be said the most elaborate and most powerful SI dependency model developed is the particular model SIDepMod(DynOT/Coop/Fcty/Inst)=SIDepMod(DynOT/Y) with Y=Coop/Subj:X/Inst with X=Fcty, i.e., the SIDepMod(Coop) instantiated for functionality instantiations (SIDepMod(Subj:Fcty/Inst)), and this extended to dynamics over time.

SIDepMod(DynOT/Y)	
reusing any SIDepMod(Y)	
types of dependent degradations:	reused from SIDepMod(Y), extended into a function of time potentially described by a template expression;
associations of dependent degradations:	reused from SIDepMod(Y), extended into a function of time potentially described by a template expression;
additional dependency constraints:	reused from SIDepMod(Y), extended into a function of time; potentially described by a template expression;
	additional function of time for expressing the validness/existence of the dependency at all potentially described by a template expression;
	general constraints over the temporal function templates expressions;

Table 4.24: Overview of DegDep specification for SIDepMod(DynOT/Y) (compare Table 4.5

4.3.11 Implementation of impact dependency models

After having iteratively designed impact dependency models from Sect. 4.3.2 to Sect. 4.3.10, guidelines for actual realization of impact dependency models, mainly with respect to the basic component architecture, BCArch, and its basic realized workflow, BRWf, (see Sect. 4.2.5), are presented in the following. Particular implementation techniques for the realization of the respective architecture components and the related, used data structures are introduced. First, the whole design of impact dependency models and related architecture components is summarized, and based on this guidelines for the realization by respective, concrete implementation techniques are given.

summary of impact dependency models and their usage

Impact dependency models (initially introduced in Sect. 4.2.2.2 on p. 140 and p. 144) are the key factor to impact analysis, as they specify all potential dependencies between the various types of degradations. The general information parts which are necessary to describe degradations and their dependencies have already been generically introduced in Sect. 4.2.2.1 on p. 134. The design of impact dependency models, performed from Sect. 4.3.2 to Sect. 4.3.10, in this sense, represents a high refinement of this early introduction on p. 134.

In the BRWf, impact analysis (IA), i.e., SIA and BIA, is actually realized by two particular *I/RA modules* (compare p. 199), namely the *service impact analyzer* and the *business impact analyzer*, which utilize the respective impact dependency models. These two I/RA modules are very similar concerning their operation, and therefore are both a refinement of a common, generic *impact analyzer* (compare p. 201). They differ only in their respective impact dependency models, i.e., the former is concerned with service impact

dependency models (Sect. 4.3.4-Sect. 4.3.10), while the latter is concerned with business impact dependency models (Sect. 4.3.3). The basic operation of these two I/RA modules is generically specified (generic impact analyzer) on p. 203 as part of the BRWf, namely the *basic realized IA subworkflows*, BRWf.1.1 and BRWf.1.2 (see especially Fig. 4.43 on p. 205):

degradations working memory and degradation derivation tree

In addition to its used impact dependency models, which generally specify all potential degradation dependencies, an impact analyzer comprises a so-called *degradations working memory* (compare p. 204), which actually remembers all degradations derived so far by the impact analyzer in the current run of I/RA, including their *degradation derivation tree* from source degradations to target degradations. That is, the contents of the degradations working memory represents the actual subset of degradations and their respective dependencies (potentially appropriately instantiated), which are related to the *actual impact situation* (compare Fig. 4.5 on p. 130 as well as Fig. 4.16 on p. 152) of the current I/RA run, out of the set of the all potential degradations and all their respective dependencies.

More specifically, the degradations working memory is used in the following way: It contains especially the (potentially instantiated to a degree as far as necessary) degradations initially given as input (initial resource degradations for SIA, and derived, top-level service degradations for BIA), as well as eventually the derived, top-level business degradations, which represent the essential output of IA. Moreover, it also contains and remembers any intermediately derived, instantiated degradations, as well as any instantiated degradation dependency which actually has been used to derive target degradations from source degradations. Thus, eventually it includes the complete (instantiated) derivation tree from source degradations to target degradations, for all instantiated degradations and their dependencies which are relevant to the actual impact situation. That is, it remembers any (instantiated) degradation dependency which was used during IA from its impact dependency model to derive particular (instantiated) target degradations from particular (instantiated) source degradations.

later reuse of degradation derivation tree during impact rating

The whole derivation tree of instantiated degradations and their dependencies is remembered in the degradations working memory, in order to ensure an easy later reuse of this particular (instantiated) dependency information, mainly for reuse during indirect impact rating of source degradations (compare p. 170): During impact rating, (instantiated) the rating of particular source degradations is derived from the rating of their particular target degradations. That is, the particular involved (instantiated) degradation dependencies are used in the reverse order (in comparison to IA), i.e., from target degradations to source degradations (compare also particular design of rating models, which will be discussed in Sect. 4.4.1). That is why it is useful to remember the used (instantiated) derivation tree from source degradations to target degradations, instead of having to recalculate all necessary information later-on during impact rating.

Fig. 4.85 presents an overview of impact dependency models and their rela-

tionship to the other related components of the basic component architecture, namely the impact analyzer, specifically the degradations working memory, as well as the impact rater as the I/RA module performing impact rating later-on.

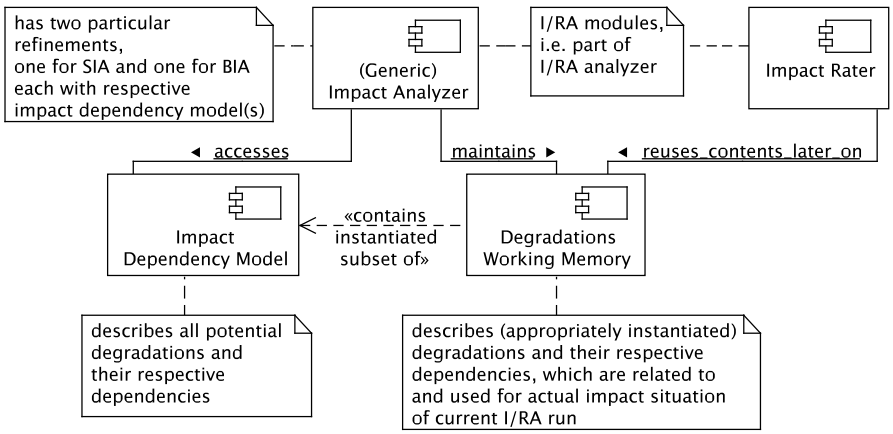

Figure 4.85: Overview of the impact dependency models and their relation to other components of the basic component architecture (compare Fig. 4.41 and Fig. 4.43)

towards a concrete implementation

Having summarized the design of impact dependency models and their related components of the basic component architecture, namely the impact analyzer as a particular I/RA module, and specifically its degradations working memory, in the following the actual realization guidelines by respective, concrete implementation techniques are discussed. Different types of implementations may be considered, each based on different existing methods or techniques, such as RBR (Sect. 3.6 on p. 103) or CBR (Sect. 3.6.3 on p. 108). The best suitable method or technique will be chosen, and necessary data conversions may be specified.

reasoning about degradation dependencies

For impact analysis, the main issue is the reasoning about degradation dependencies from source degradations to (entailed) target degradations. Consequently an implementation of impact models and of a corresponding *impact analyzer* (as an I/RA module, compare above) including a degradations working memory for remembering the currently derived degradation derivation tree, needs to support such a reasoning about dependencies appropriately. The degradation dependencies are to be specified beforehand to their actual usage, before actual I/RA runs, so that it can be assumed that they are in place in a complete, consistent manner. Of course, only proper, static model data has to be in place prior to actual I/RA runs, while for additional dynamic data only references have to be known priorly from this static data (compare p. 143). Consequently, for an implementation of impact analysis a reasoning about priorly, explicitly, and completely specified degradation dependencies has to be supported.

using RBR

In general, such a type of reasoning can be addressed with RBR (Sect. 3.6 on p. 103), i.e., the reasoning by logic rules. Basically, the degradation dependen-

cies can be encoded in logic rules, which can be followed by a logic reasoner to derive target degradations from source degradations. Nevertheless, RBR or logic reasoning has many varieties, e.g., depending on the specific purpose, the particular data paradigm and particular logical calculus actually utilized: In the case of impact analysis, from a potentially small set of given initial resource degradations a potentially large ramified tree of entailed degradations (degradation derivation tree) has to be derived. Stating it alternatively, impact analysis has to identify and derive all entailed degradations, i.e., to find all solutions for the problem of deriving target degradations from source degradations (impact derivation problem). But, most types of logical reasoning (prolog-like reasoning) are more designed to check only whether one solution exists at all (and present this one). That is why, these types often lack efficient ways to find all solutions of an impact derivation problem. Basically, their inefficiency is the result of their inability to remember past/partial solutions of parts of the search tree, so that these partial solutions often have to be recalculated. Nevertheless, among RBR in general, the deductive database (DDB) approach (see Sect. 3.6.1) efficiently addresses this issue, as it combines prolog-like reasoning with a database where past and partial solutions are remembered. This approach is specifically designed to calculate the whole set of valid facts which result from a set of specified base facts and specified rules. Basically, this approach of DDB corresponds to the impact derivation problem of IA in the following way: the initial resource degradations can be regarded as DDB base facts, all potential degradations dependencies of the impact dependency models can be regarded as the DDB rules (or are encoded by them), and all degradations (eventually) entailed from the initial resource degradations, i.e., all degradations in the degradation derivation tree, can be regarded as valid DDB facts.

using DDB approach

Consequently, the deductive database approach as a particular type of logical reasoning is chosen as the basis for the implementation of IAFw, i.e., of impact dependency models and of the impact analyzer.

But, among the DDB approach also varieties exist. Especially, concerning the data paradigm, relational and object-oriented can be distinguished: The latter one subsumes the former one, the more classical type of logical reasoning, in that it allows to combine the reasoning about logical relations or predicates in general with the reasoning about specific object-oriented related relationships of objects and classes. As the design of impact dependency models developed iteratively from Sect. 4.3.2 to Sect. 4.3.10 deals with many different objects and classes, e.g., for specification of degradations, an object-oriented approach is more promising. Concluding, the implementation of impact dependency models and the impact analyzer is based on the object-oriented deductive database approach.

object-oriented DDB approach

Consequently, all kinds and aspects of (class) dependencies between the information parts (classes of SIDepMod(.)) identified in the iterative design from Sect. 4.3.2 to Sect. 4.3.10 are to be encoded in deductive, logical rules. Conceptually, these rules can be used by a deductive database reasoner to de-

encoding of degradation dependencies by deductive rules

rive from given initial resource degradations as base facts all entailing target degradations as the set of all valid facts. Thus, the deductive database reasoner will represent the core of the implementation of the impact analyzer (as an I/RA module, p. 199).

Actually, only the proper, static data (compare p. 143) of the impact dependency model are to be encoded directly as rules. The access to additional dynamic data (compare p. 143), i.e., by m/d i/o modules controlled by m/d i/o data engine (see Sect. 4.2.5.3), has to be additionally integrated with the deductive database reasoner.

Conceptually, such an encoding of a degradation dependency may look like the following:

degradation dependency specification: $\{A_j(P_{j_k})\} \rightarrow B(P_{0k})$ with possible constraint condition $Cnd(P_{0k}, P_{j_k})$.

$A_j(P_{j_k})$ are parameterized source degradation specifications (parameters P_{j_k}) and $B(P_{0k})$ is the parameterized target degradation specification (parameters P_{0k}). Parameters are used for differentiating all information aspects of degradation specifications. $Cnd(P_{0k}, P_{j_k})$ is an optional condition for applying constraints on the actual matching source and target degradation specifications.

For the appropriate evaluation of the constraints of a degradation dependency the deductive database reasoner may have to be extended, at least for access to external components which allow to evaluate parts of the constraint expression which the deductive database reasoner is not familiar with.

use of Frame Logic

As one particular language for actually encoding the degradation dependencies *Frame Logic* (see Sect. 3.6.1) is chosen, as it is one of today's standard languages for use within object-oriented deductive databases. This language is specifically object-oriented, as discussed above, i.e., it supports explicitly inheritance class hierarchies, and attributes directly as a language feature.

example implementation

In Appendix B an example implementation for impact dependency models of the example scenario in Sect. 2.3 is given. This example implementation is using particularly such an approach with Frame Logic as described above, describing degradations and their dependencies by deductive facts and rules in an object-oriented way. Moreover, the deductive database system used as basis, Flora2 [YK00, YKZ03], uses an extension of Frame Logic so that it is not only a query language, but instead a full-featured programming language. This way, this example implementation provides a basis for the complete implementation of an impact analyzer module (compare above), at least as far as the use of static model data is concerned. For the integration of additional, dynamic data appropriate data fetch mechanisms have to be integrated, e.g., by using procedural methods in Flora2, or by adding corresponding prolog predicates with side effects (data exchange with appropriate m/d i/o modules) in the underlying prolog system XSB [xsb].

4.4 Recovery Analysis Framework

In the following the *recovery analysis framework* (*RAFw*) as the second extension framework of the *basic framework* (*BFw*) is treated. Fig. 4.86 as a reminder of Fig. 4.1 illustrates the overall structure of the I/RA framework specifically emphasizing the recovery analysis framework. The basic framework, introduced in Sect. 4.2, provides a generic basis for I/R analysis. It introduced basic terms, concepts, and workflow steps for I/R analysis in general. Based on this, the RAFw extends, refines, and details these basic notions, concepts, and workflow steps introduced particularly for recovery analysis (compare Fig. 4.1 on p. 122). More specifically, the RAFw is concerned with the design and realization of actual data structures, and their specific usage in the respective workflow steps.

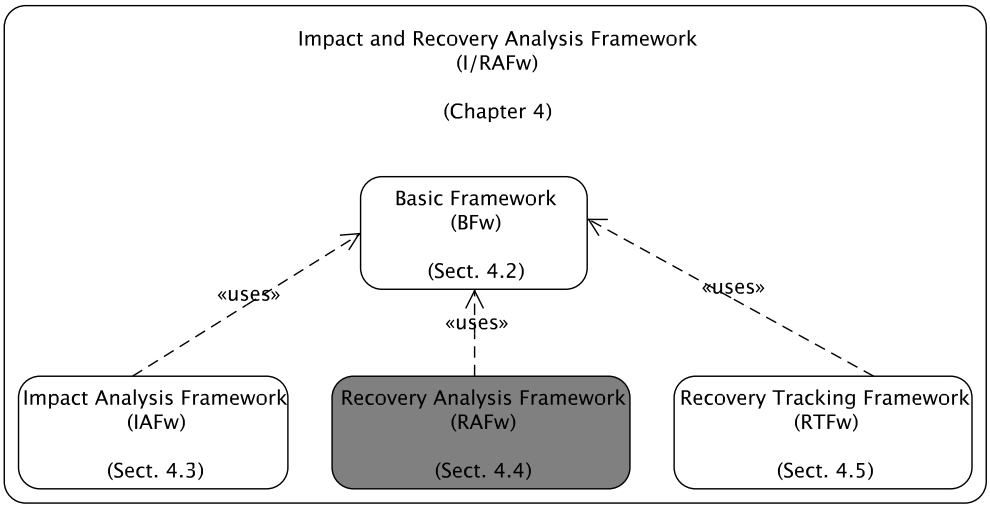

Figure 4.86: Recovery analysis framework (RAFw) as second extension framework of the basic framework (reminder of Fig. 4.1)

impact rating + recovery design
RA models

As having been discussed in Sect. 4.2.3.1 on p. 170, recovery analysis (RA) is subdivided into the two subsequently performed steps *impact rating* and *recovery (plan) design*. Moreover, in Sect. 4.2.3.2 the notions of the *impact rating model* for the purpose of impact rating, and the *recovery (action) dependency model* for the purpose of recovery design were introduced. Both are subsumed under the term *RA models* in the following.

basic RA subworkflows

The usage of both RA models roughly has been treated in Sect. 4.2.3.2 (basic refined abstract recovery analysis subworkflow, BRAWf.2) as well as in Sect. 4.2.5.6 on p. 206 (basic realized impact rating subworkflow, BRWf.2.1), and on p. 207 (basic realized recovery design subworkflow, BRWf.2.2).

impact rating model

The rating model allows the *direct rating of the business degradations* derived by IA. Based on this, the *indirect rating* of prior degradations which are entailing the business degradations is performed. Prior, entailing degradations here comprise prior service degradations, as well as prior resource degradations entailing the service degradations in turn. The indirect rating is actually

done by using the degradation dependencies of impact dependency models in the reverse direction (compared to IA). So, eventually also the originally given resource degradations, which are the essential input to the whole I/R analysis, are rated indirectly in this way. In Sect. 4.2.3.2 on p. 171 the rating model was introduced on an abstract level, without going into specific details concerning actual data structures or their usage.

recovery dependency model

The rating allows a basic prioritization or ordering by importance of the resource degradations, which are the targets of recovery plan alternatives to be designed by the following recovery design. The recovery (action) dependency model is used by the recovery design to construct one or multiple alternative recovery plans along with their estimated reduced impact. A recovery plan is comprised of one or multiple recovery actions, each of which is targeting one or multiple of the original resource degradations. For coordination of these recovery actions, appropriate scheduling information has also to be included in the recovery plan.

The general information parts which are necessary to describe alternative recovery plans, recovery actions, and their relationship to reduced impact were generically discussed in Sect. 4.2.3.1 on p. 164. Moreover, their general relationship to and usage for the specification of the recovery action dependency model was introduced in Sect. 4.2.3.2 on p. 173. These introductions stayed on a very rough and conceptual level, and did not include specific data structures for actual specifying particular degradations.

concretization of both RA models

Concluding, the RAFw is mainly concerned with the concretization - in terms of data structures and their actual usage - of both, the impact rating model as well as the recovery action dependency model. This concretization also comprises specification of actual data structures for recovery actions and their relationship to degradations, and recovery plans, consisting of recovery actions.

That is, the design and the subsequent implementation of both RA models, which are compatible with the design and implementation of impact dependency models of IAFw (Sect. 4.3), is the main subject of the RAFw.

only one possibility for recovery dependency models

However, this RAFw will concentrate on one particular possibility for design and implementation of recovery action dependency models. Anyway, this one will be compatible with the IA dependency model of IAFw, and will allow to perform an effective and appropriate RA. Further elaborations or alternatives of this possibility will only introduced shortly.

approach for RAFw development

Fig. 4.87 presents an overview of the approach for the development of the recovery analysis framework (RAFwAppr). First, the design of impact rating models is discussed in Sect. 4.4.1, second one particular possibility for a design of recovery action dependency models is discussed in Sect. 4.4.2. Based on these both design steps, representing only one particular possibility, guidelines for an implementation of both designed RA models are introduced in Sect. 4.4.3. Moreover, the guidelines treat also the actual usage of the implemented RA models as part of the basic component architecture introduced

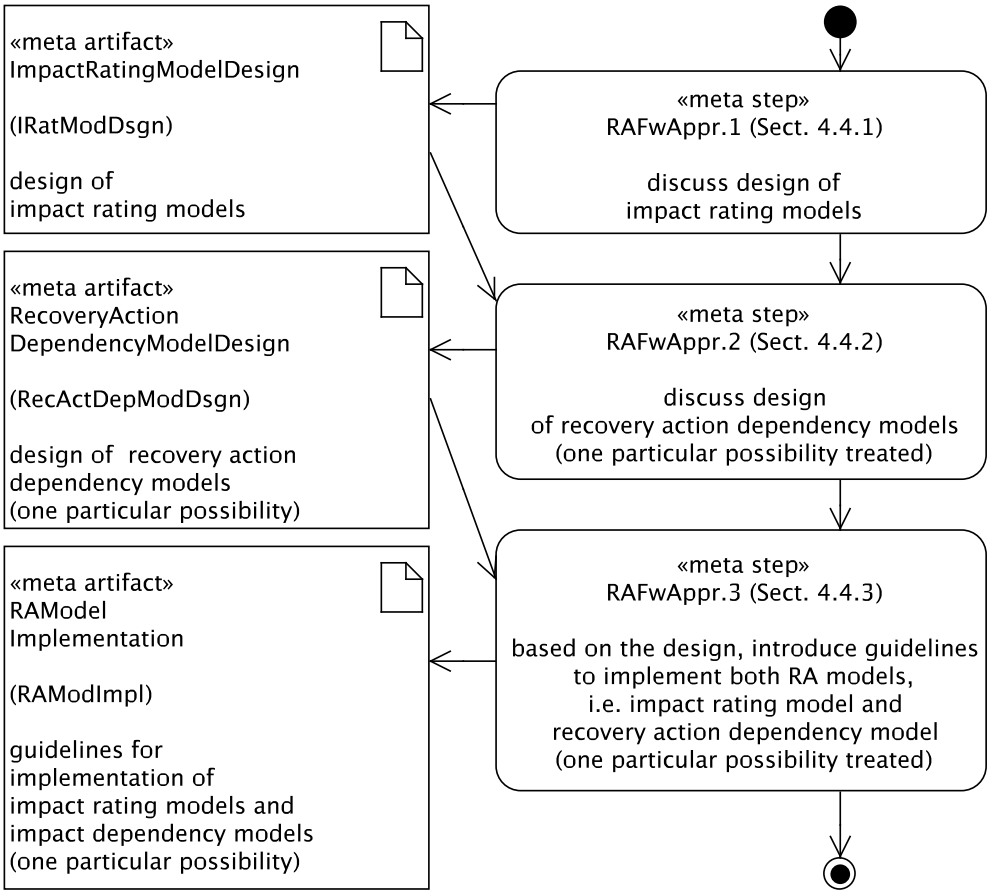

Figure 4.87: Approach (RAFwAppr) for the development of the recovery analysis framework (RAFw)

for the basic realized workflow BRWf (see p. 206 for BRWf.2.1, and p. 207 for BRWf.2.2).

4.4.1 Design of impact rating models

Here the design of business impact rating models is discussed. These models are used to rate the topmost business degradations as determined by impact analysis. The designation *topmost* for business degradations is used with respect to the derivation hierarchy of degradations from initial resource degradation and further entailed resource degradation, via service degradations, to basic business degradation, and eventual (topmost) aggregated business degradations (compare Fig. 4.11 on p. 145 and Fig. 4.13 on p. 148). In addition to the (direct) rating of topmost business degradations by the impact rating model, also the relationship to impact dependency models for the purpose of indirect rating of all degradations eventually entailing the topmost business degradations is discussed.

The rating of impact in general is performed to provide a basis for prioritization of originally given recovery degradations which eventually entail (via service degradations) the business degradations. These basic prioritization will

be input to recovery design, which further constructs complete, alternative recovery plan consisting of possibly multiple recovery actions, coordinated by an appropriate scheduling.

Concerning the requirements identified in Sect. 2.4, impact rating is especially related to and demanded by the requirement R4.1.2 (order and scheduling of recovery actions).

direct and indirect rating

As already mentioned above - impact rating is subdivided into two steps, direct rating of the topmost business degradations, and based on this indirect rating of all entailing degradations by utilizing the impact degradation dependencies in the reverse direction as used by IA (see p. 170). The former step is actually utilizing the (business) impact rating model whose design is treated explicitly in the following. Afterwards indirect rating by reverse-directed application of impact dependency models is discussed.

design of business rating model

Specifically, the design of (business) impact rating model is based on the specification of business degradations as introduced for the business impact dependency model BIDepMod(Char) (see Sect. 4.3.3), because this is actually the only BIDepMod capable of instantiation which has been treated in this I/RA framework. For BIDepMod(Char), all business degradations, i.e., also the topmost ones, are specified in detail by a *business degradation characteristic*, namely a function of time/duration specifically describing a concrete business degradation of a particular type (e.g., SLA penalty costs, revenue loss, customer satisfaction reduction, recovery costs).

rating of business degradation characteristics

With respect to BIDepMod(Char), the task of business degradation rating by the impact rating model is to unify these different types of business degradation characteristics, in terms of value unit, value granularity, and time granularity. This unification provides a consistent way to compare, rate, and so prioritize all business degradations even of different type. Moreover, in addition to unification of business degradation characteristics of different business degradation types, the impact rating model may be used to weight different business degradation of the same type, according to the actual needs of the service provider. E.g., SLA penalty costs of different customers may be weighted according to the importance the provider ascribes to its different customers: An old, long-term customer may be more important than a new one, not very acquainted yet, even if the SLA penalty costs of the latter one may be less in value in a concrete impact situation. The unification of different business degradation characteristic types in generally, but also especially the weighting of individual degradation characteristics of the same type, is highly depending on the policies of the service provider. The design of impact rating models has to allows a service provider to express such demands as required by his policies. However, this actual instantiation is left to the provider himself, who solely knows all its related business policies. Concluding, the (business) rating model specifies the unification and weighting of business degradation characteristics for all types of business degradations. Such unifications and weighting of the business degradation characteristics are specified

in the (business) degradation rating model as *business degradation characteristic to direct rating mappings*.

Fig. 4.88 gives an overview of the classes used to specify such mappings between business degradation characteristics types/instantiations and direct degradation ratings. Conceptually, each business degradation characteristic

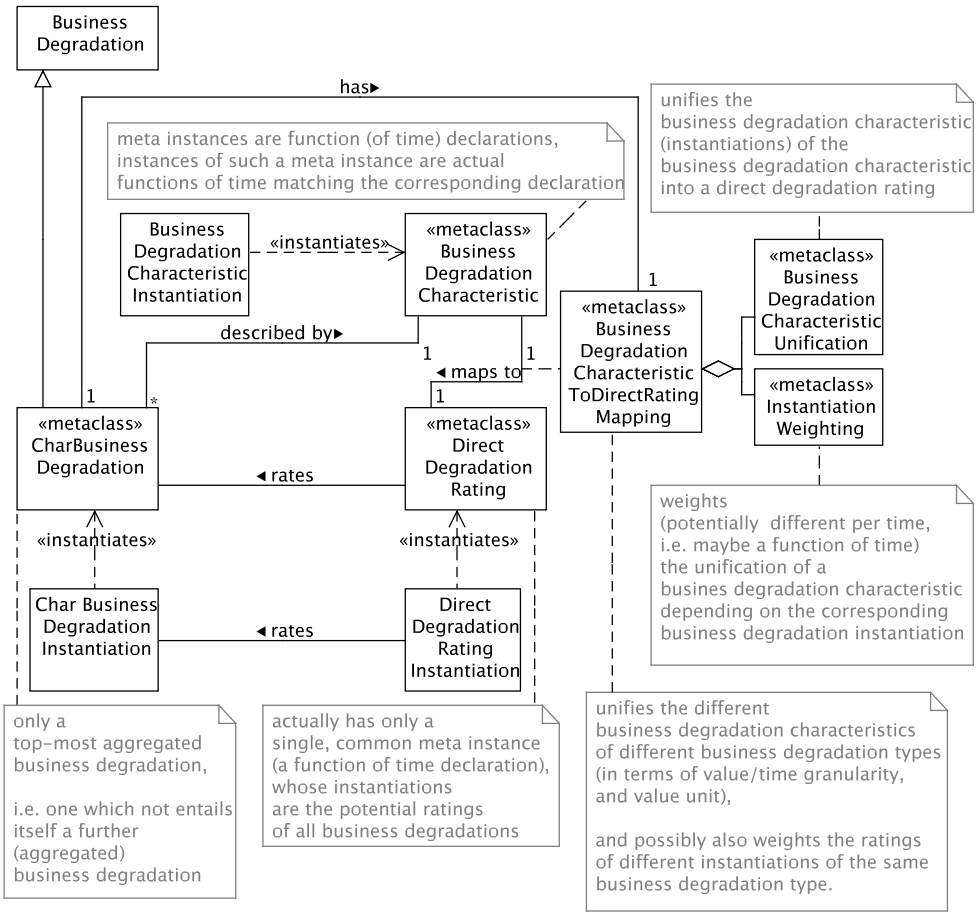

Figure 4.88: Design of the mappings between business degradation characteristic (types) and direct degradation ratings, contained in the rating model

type (a function of time declaration) is converted into a common, unique *business degradation rating declaration* (a unique function of time declaration). As *char business degradation* and its related classes defined for BIDepMod(Char) are actually meta classes (compare p. 228), this relationship can be reformulated correspondingly on the level of such business degradation instantiations: the business degradation characteristic instantiation of any business degradation instantiation, as a function of time and conforming to its business degradation characteristic type, is converted into a *business degradation rating definition*, again a function of time. All business degradation rating definitions of any business degradation type conform to the same, common function of time declaration, namely the common business degradation rating declaration. This way, all these rating definitions are directly com-

parable, and priorities (per elapsed time) can be uniquely and consistently assigned to all of them.

The *business degradation characteristic to direct rating mapping* for a business degradation type includes a *weighting function* for weighting individual instantiations (possibly also depending on elapsed time), and a *business degradation characteristic unification*, which is common to all instantiations of a particular business degradation type. The (business) rating model as a whole is consisting of all these mappings for any business degradation type.

indirect rating by using degradation dependencies reversely

Based on the directly rated, topmost business degradations, recursively all entailing degradations are indirectly rated, from the business degradations via service degradations to resource degradations, especially to the resource degradations originally given as input to I/R analysis. For this purpose the degradation dependencies of the impact dependency model, which were originally used to derive the business degradations from initial resource degradations, are used in the reverse direction.

Fig. 4.89 gives an overview of the combination of direct rating with the rating model mappings, explained above, and the recursively performed indirect rating of all entailing degradations with indirect rating mappings specified by the respective degradations dependencies.

The degradation dependencies, which are followed in reverse direction compared to IA, i.e., from their target degradations to their source degradations, are especially used to determine a (indirect) *rating weighting* for the source degradations. These rating weightings particularly specify how to distribute an already calculated degradation rating definition of a target degradation on multiple source degradations. That is why such rating weightings are only necessary in the case of degradation dependencies with multiple source degradations. For degradation dependencies with only a single source, the source can get assigned the same rating definition as the target.

The weighting ratings of multiple sources are calculated from expressions or by algorithms which are associated to the specific degradation dependency, depending on the particularly given source and target degradations. Later on in the actual realization of I/R analysis, these calculations of source weightings are ideally performed already during the impact analysis, i.e., when following the degradation dependencies from sources to targets, and are remembered (together with the corresponding source degradations) until the impact rating takes place.

additional degradation dependency constraints

The expressions/algorithms for weighting of source degradations can be actually specified by additional degradation dependency constraints (see Fig. 4.52 on p. 218). In the case of a complex service impact dependency model, such as e.g., SIDepMod(Subj:X/Inst) (see Fig. 4.72 on p. 253) or SIDepMod(QoXInst/Subj:X/Inst) (see Fig. 4.81 on p. 269), these constraint definitions can be part of the constraints which are already part of the complex dependency specification. Otherwise, such constraints can be added further to the dependency specifications for the sole purpose of source rating weighting.

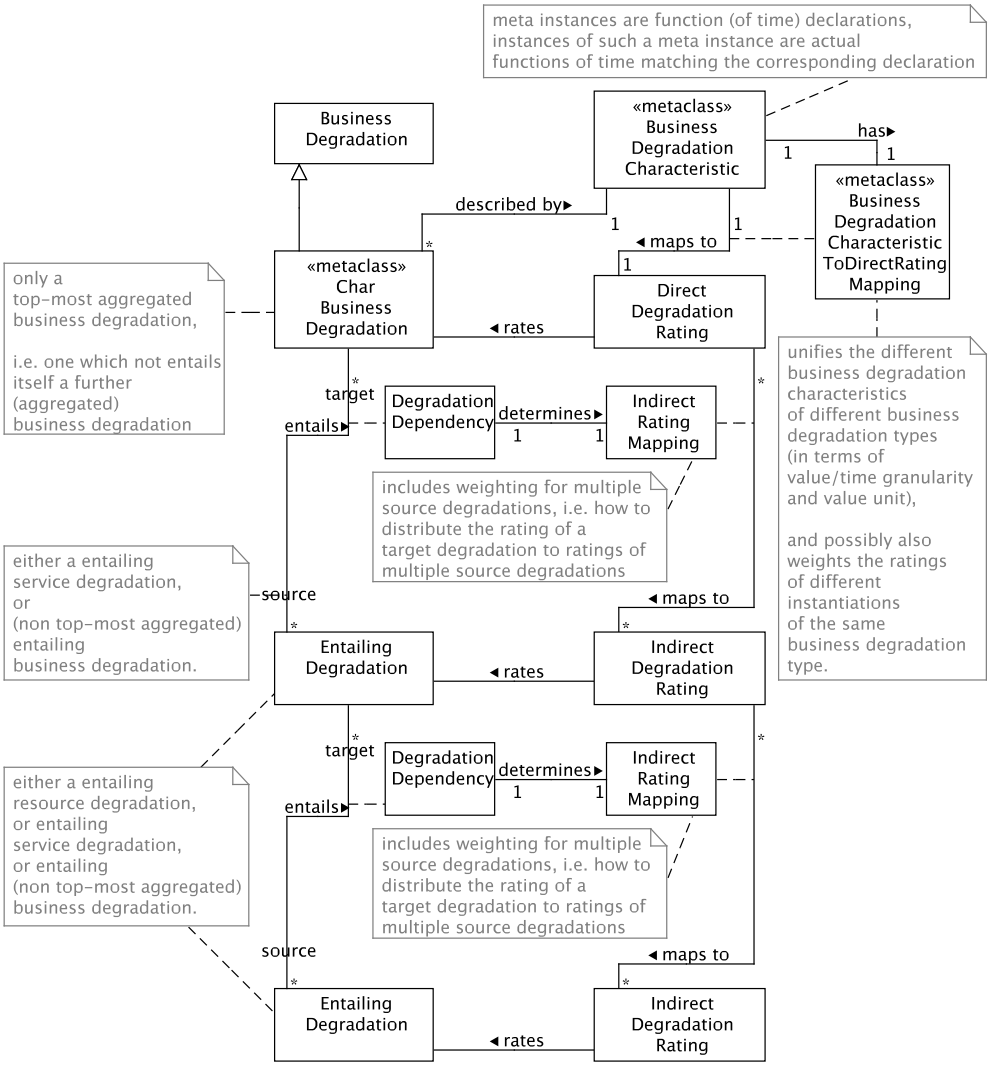

Figure 4.89: Overview of direct rating by use of business impact rating model and indirect rating of entailing degradation by use of weighting mappings determined from impact dependency models

Concluding, the degradation rating definitions of target degradations are distributed on their sources by weightings determined by the corresponding degradation dependency, whether the actual calculation is already done during impact analysis or deferred until their actual usage during impact rating. If a particular degradation takes place in multiple, different degradation dependencies as a source, the different weighted indirect rating definitions it gets assigned from its different targets are simply added up (as functions of time), as the different target degradations lastly represent independent business degradations.

Above, in the last case, i.e., of multiple, different degradation dependencies with common dependent degradations, for the specification of the (indirect) rating weighting specifications, it was assumed that a single source degradation has multiple dependencies to different target degradations, i.e., this situation represents a $1 : m$ relationship between a common source degradation

and multiple target degradations. In general, as an extension of this, a more elaborated notion of rating weighting specifications considering $m : n$ relationships with common dependent degradations can be developed. Thus, with respect to (indirect) rating weighting specifications of source degradations, also $m : n$ relationships between dependent source and target degradations can be taken into account, e.g., for the $m : n$ relationship of different QoX parameter values ($m : n$ quality degradation dependencies, on quality degradations in general compare Sect. 4.3.8): For example, the accumulated degradation of multiple different delay parameter values on the resource layer might entail jointly the degradation of multiple, different more high-level (level of functionality or service) delay parameter values. Consequently, in order to address an joint (indirect) weighting rating of such multiple, inter-related, common source degradations, a more complex specification about how to derive the (source) ratings as a combination of the ratings of their targets in a joint, consistent manner, instead of simply adding them up, is required. However, in this RA framework here, such complex $m : n$ rating weightings are not explicitly taken into account, and a corresponding, more elaborated notion is left as further work.

example

Following, the rating of the example recovery plan $ExRecPlan1$ of the example situation $ExSit1$ (compare p. 167) is discussed with respect to the above introduced design of impact rating models. For $ExSit1$ only SLA penalty costs were considered as business degradation types, so only SLA penalty functions (slps) are used as business degradation characteristic declarations (compare p. 227). That is why unification of different business degradation characteristic types is not necessary, i.e., the slp functions' declaration (euros per time) can be used as unified direct rating function declaration. Also, no particular weighting of different slps takes place.

Thus, in this example, the slp functions of each (top-level) business degradation can be used directly as the (direct) degradation rating definition, $dgrat(.)$ (compare Table 4.3 on p. 136):

- $dgrat(g_{b1-1}) := slp_{\text{mail3}}(t)$
- $dgrat(g_{b2-1}) := slp_{\text{mail1}}(t) + slp_{\text{mail2}}(t)$
- $dgrat(g_{b2-2-1}) := slp\_prem_{\text{web1}}(t) + slp\_prem_{\text{web2}}(t)$
- $dgrat(g_{b2-2-2}) := slp\_norm_{\text{web1}}(t) + slp\_norm_{\text{web2}}(t)$

In the general case these rating definitions would have the general form $dgrat(g) :=$
$weight\_function_g(unify\_function_g(business\_degradation\_char_g(t)))$.

Based on these direct rating definitions, indirect rating of entailing degradations is performed (compare Fig. 4.12 on p. 146, as well as p. 168): g_{b1-1} is entailed only by g_{s1-1} and so propagates its rating definition to g_{s1-1}, i.e., $dgrat(g_{s1-1}) = dgrat(g_{b1-1}) = slp_{\text{mail3}}(t)$. In turn, g_{s1-1}, i.e., high mail sending delay, is entailed jointly by g_{s1-0}, i.e., high IP path delay for finally sending the mail, and g_{1b-0}, i.e., high DNS response delay. The high IP path

delay increases the mail sending delay about $2.5\ \mathrm{min}$, whereas the DNS response delay increases it about $0.5\ \mathrm{min}$ (compare p. 168). The latter increase by the DNS response delay is about $4\ \mathrm{min}$, because the increase of the DNS delay itself is $0.5\ \mathrm{min}$, and a normal mail sent to only one receiver actually utilizes two DNS resolve requests, i.e., doubling the effect of large DNS response delay ($15\ \mathrm{s}$ instead of $< 1\ \mathrm{s}$) for mail sending. Concluding, the both source degradations, g_{s1-0} and g_{1b-0}, get assigned rating weightings of $5/6$ ($= 2.5/3$) and $1/6$ ($= 0.5/3$), that is $dgrat(g_{s1-0}) := 5/6 \cdot dgrat(g_{s1-1}) = 5/6 \cdot slp_{\mathrm{mail3}}(t)$ and $dgrat(g_{s1b-0}) := 1/6 \cdot dgrat(g_{s1-1}) = 1/6 \cdot slp_{\mathrm{mail3}}(t)$. In turn, g_{s1-0} and g_{s1b-0} propagate their rating definition to their entailing degradations, g_{r1} and g_{r1b} respectively, i.e., $dgrat(g_{r1}) = dgrat(g_{s1-0}) = 5/6 \cdot slp_{\mathrm{mail3}}(t)$ and $dgrat(g_{r1b}) = dgrat(g_{s1b-0}) = 1/6 \cdot slp_{\mathrm{mail3}}(t)$.

In the same manner, the indirect rating definitions for the entailing degradations of g_{b2-1}, g_{b2-2-1}, and g_{b2-2-2} are derived:
$dgrat(g_{s2-1}) := dgrat(g_{b2-1})$ (single propagation from target to source),
$dgrat(g_{s2-2}) := dgrat(g_{b2-2-1}) + dgrat(g_{b2-2-2})$ (two independent target degradations with same source),
$dgrat(g_{s2-0}) := dgrat(g_{s2-1}) + dgrat(g_{s2-2})$ (two independent target degradations with same source),
$dgrat(g_{r2}) := dgrat(g_{s2-0})$ (single propagation from target to source).
This actually yields
$dgrat(g_{r2}) = (slp_{\mathrm{mail1}}(t) + slp_{\mathrm{mail2}}(t)) + [(slp\_prem_{\mathrm{web1}}(t) + slp\_prem_{\mathrm{web2}}(t)) + (slp\_norm_{\mathrm{web1}}(t) + slp\_norm_{\mathrm{web2}}(t))]$.

Now, the degradation rating of the (initial) resource degradations g_1, g_{1b}, and g_2 can be uniquely compared (per elapsed time) to assign priorities (depending on elapsed time) (compare the evaluation on page p. 53).

4.4.2 Design of recovery action dependency models

In the following, a particular possibility for a design of recovery (action) dependency models is discussed. The design also includes one particular design of recovery actions, and recovery plan alternatives consisting of recovery actions, as well as the corresponding design of reduced impact of these recovery alternatives.

Related requirements (Sect. 2.4) for recovery dependency model are specifically R4.1 (aspects of recovery actions), as well as R4.2 (consideration of recovery costs).

As recovery action dependency models are concerned especially with recovery actions, a particular design and related modeling for them is treated first. Afterwards, the design of recovery action dependency model itself is discussed. Lastly, the design of reduced impact for a recovery plan is addressed.

The notion of recovery actions as part of a recovery plan alternative was introduced in Sect. 4.2.3.1. The relationships between recovery actions and degradations in general were also discussed in abstract manner (compare e.g.,

Fig. 4.21 on p. 165). Basically, a recovery action is concerned with the handling, i.e., (partial or full) reduction or elimination of some of the initial resource degradations. As consequence of trade-offs to be made they may aggravate or introduce other resource degradations. Moreover, usually they cause costs for their realization which is regarded here as a particular kind of business degradation (see p. 158).

scheduling and coordination

For any recovery action as part of a recovery alternative, it has further to be decided when (in relationship to the others) it should be executed, i.e., the coordination or scheduling of recovery actions has to be specified.

additional parameters

Moreover, a particular kind of recovery action, which is targeting at a particular resource degradation, may have some additional parameters additionally describing its actual realization, e.g., concerning the amount of effort used, or more specifically the manpower needed.

Concerning the example situation $ExSit1$ (see p. 164), additional parameters are e.g., for the recovery action $RecAct(g_{r2}, 1)$ (reboot of broken AFS device) the maximum number of tries to reboot, for the recovery action $RecAct(g_{r2}, 2)$ (replacing broken AFS device with backup device) which replacement alternative to actually use and how fast and with how much manpower it should be replaced, for recovery action $RecAct(g_{r2}, 5)$ (acquiring a new complete new AFS device) the maximal price for a new device and the duration for acquiring it.

Therefore, in contrast to coordination/scheduling issues, such additional parameters are depending on the particular kind or type of recovery action (as e.g., $RecAct(g_{r2}, 1)$, $RecAct(g_{r2}, 5)$ above). For a single kind of recovery action type, there may be multiple, even infinitive (continuous value spectrum) values for each of these additional parameters. In the following these additional parameters are designated as *recovery action (type) individual parameters* in order to distinguish them from scheduling aspects which can and will be also specified by parameters, so-called *recovery action coordination parameters* (or *recovery action scheduling parameters*). The recovery action coordination parameters will be in contrast to the recovery action individual parameters common to all recovery actions types.

The above discussion motivates the basic approach for the design of recovery actions: by recovery action types with parameter list declarations, and corresponding parameter action instantiations with conforming parameter list definitions. Fig. 4.90 shows the classes used for such a design and modeling of recovery actions. Furthermore, it also shows how recovery plans are specified, consisting of recovery actions. Recovery actions are basically distinguished by different recovery action types, as motivated above. Each recovery action can have different recovery action instantiations which are distinguished by specific recovery action parameters bindings: A recovery action type has a parameter list declaration, i.e., a list of named parameter type declarations. In turn, each recovery action instantiation of a particular recovery action type has a corresponding parameter list definition conforming to the parameter list declaration of its recovery action type. Actually, there are two kinds of recov-

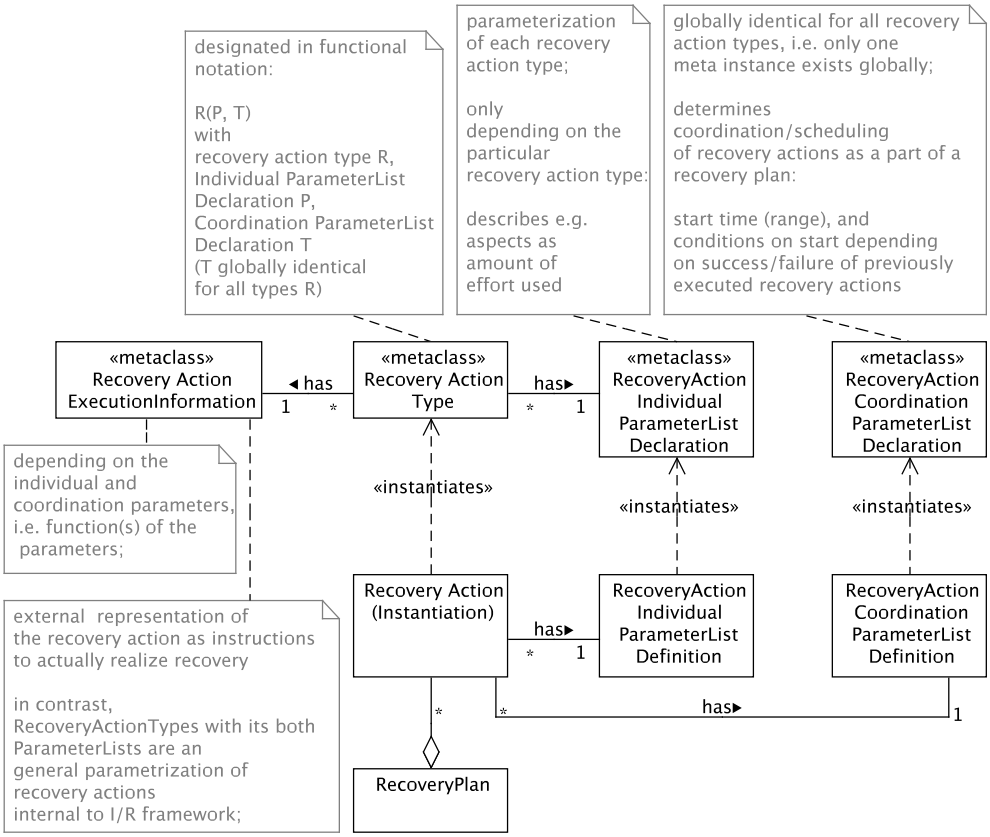

Figure 4.90: Design for recovery action (types and instantiations) and recovery plans consisting of them

ery action parameter list declarations: an individual recovery action parameter list per recovery action type, and a coordination recovery action parameter list common to all recovery action types. The former one covers the individual parameters (e.g., specific amount of effort), the latter one covers the common coordination and scheduling of recovery actions within a recovery plan.

Consequently, a recovery action instantiation is specified by its recovery action type, a corresponding individual recovery action parameter list definition, and a coordination recovery action parameter list definition. Based on this, recovery plan alternatives are modeled as a set of particlar recovery action instantiations.

This approach of modeling recovery actions by types with parameter list declarations, instantiations with parameter list definitions is similar to the approaches of e.g., modeling degradation subjects with SIDepMod(Subj:X/Inst) (see Fig. 4.72 on p. 253), or the modeling of QoX degradation instantiations with SIDepMod(QoXInst/Subj:X) (see Fig. 4.82).

internal and external parameterization

The specification by recovery action types and corresponding recovery action parameters is regarded as an abstract parameterization internal to the I/R analysis framework, which is suitable for applying search and optimization algorithms to it. In contrast to this internal parameterization there is the *recovery action execution information* (see p. 166) for a recovery action, which for the

design is regarded as an external representation of the recovery action, more specifically a set of instructions to operators for actual realization of the recovery action. Generally, for each recovery action instantiation, i.e., depending on its individual and coordination parameter values, there is a specific execution information, comprising an external description of the recovery action type, individual and coordination parameters for the purpose of operator instruction. This external description is not directly relevant for the search and optimization algorithms used to construct recovery alternatives. Only when the recovery plan alternatives are completely designed, and immediately before they are given as output of recovery analysis to the operators for further selection, the mapping to the external representation as execution information is performed. Moreover, the execution information of the recovery plan actually selected by the operators (selected recovery plan) is later on used as essential input for recovery tracking.

particular possibility for coordination parameter list declaration

The individual parameter list declaration is, as discussed above, depending on each particular recovery action type. But the coordination parameter list declaration is common for all recovery action types. That is why one particular possibility for such coordination parameters is presented here. This choice actually determines the power of expression for recovery plan alternatives which can be represented and designed (explicitly) by I/R analysis. The particular choice here will allow for programmatic concepts as sequential execution, conditional execution, and simple parallel synchronization of recovery action (instantiations). Loops (especially while-loops), sub-programm calls, or further programmatic concepts are not addressed here. For these an extended coordination parameter list declaration, or complete other approach for modeling of recovery actions and recovery plans would be necessary. The explicit choice of a recovery action coordination parameter list declaration proposed here is (denoted as proposal $CoordParamListDecl1$):

- recovery_action_number: integer or string (to allow to reference uniquely)
- start_time_range: time or time range
- start_depending_on: boolean formula (using *not*, *and*, *or* as operators) referencing precedingly executed recovery actions by their numbers.

The first parameter, *recovery_action_number*, is simply for identification and further reference of the respective recovery action. The second parameter, *start_time_range*, allows to specify sequential starting of recovery actions, as well as parallel starting of the recovery actions, depending on the actual time values used. Lastly, the third parameter, *start_depending_on*, allows to specify synchronization and conditional execution: First, it allows to defer the execution of a recovery action (instantiations) until one or multiple preceding ones are finished, i.e., it allows to synchronize the starting and ending of subsequent execution recovery actions. For each recovery action (instantiation) referenced in this third parameter value it is waited for the completion of the referenced recovery action (instantiation). Second, it makes it possible to base

the decision whether to actually execute the recovery action (instantiation) on the actual success of these precedingly executed recovery action (instantiations). That is, the execution of a (realized) recovery action instantiation is considered to have a particular success value, here a boolean value denoting actual success (value *true*) or failure (value *false*). For example, for the example recovery plan $ExRecPlan1$ (p. 164), the recovery action $RecAct(g_{r2}, 1)$ for handling g_{r2} by rebooting the broken AFS device may succeed or fail. This potential success or failure determines the execution of further recovery actions, such as $RecAct(g_{r2}, 2)$.

The respective recovery action (instantiation) is only actually executed if the boolean formula described by the third parameter value evaluates to true after substituting the reference numbers of preceding recovery action (instantiations) by their success values. A recovery action instantiation for which the decision to execute it conditionally has already been made, and has yielded the result of not executing it, is considered to have a positive success value (true) for the purpose of substituting it in the boolean formula (third parameter value) of a subsequent, conditionally executed recovery action instantiation.

In general, the structure of the (potential) success of a recovery action instantiation may be designed in a more complex manner, i.e., there may be multiple components of such a success value, or each of them may be represented by types more detailed than boolean, e.g., by appropriate random variables. That is, in the general case, the success of a recovery action instantiation is described by a *recovery action success parameter list declaration*. But for the particular proposal of the recovery action coordination parameter list declaration $CoordParamListDecl1$ presented above the simple form of a single boolean value, designated as $SuccParamListDecl1$, is chosen. A corresponding, explicit modeling of success parameter list declarations will be treated later on below.

example

For the recovery plan alternative $DgrHndl(g_{r2}, 1 + 2 + 3 + 4 + 5)$ of $ExRecPlan1$ (p. 164), this yields the following recovery action coordination parameter bindings (excerpt):

- $RecAct(g_{r2}, 1)$:
  - recovery_action_number = 1.1
  - start_time_range = 0 min
  - start_depending_on = -
- $RecAct(g_{r2}, 2)$:
  - recovery_action_number = 1.2
  - start_time_range = 5 min
  - start_depending_on = not(1)
- $RecAct(g_{r2}, 3)$:
  - recovery_action_number = 1.3

  - start_time_range = 3 hours
  - start_depending_on = not(1) and not(2)

- ...

After discussing the design and modeling of recovery action (and their execution information), the design for dependencies to influenced and resulting degradations as the main content of the recovery action dependency model is discussed. The notions of influenced and resulting degradations of a recovery action were introduced in Sect. 4.2.3.1 on p. 166. Based on this, the corresponding dependencies as part of the recovery actions dependency model were discussed in Sect. 4.2.3.2 (compare Fig. 4.24 on p. 172 as well as Fig. 4.25 on p. 174). These dependencies are specifically termed *recovery action effect dependencies* in the following, with respect to the notion *effect* being the subsumption of *influence* and *result* of a recovery action (see p. 166). Fig. 4.51 on p. 217 introduced a generic outline for the design and modeling of dependencies in general, which is also valid for recovery action effect dependencies. But these dependencies are more complex than degradation dependencies for which Fig. 4.51 was instantiated previously in Sect. 4.3. Degradation dependencies have specifically source degradations and target degradations as their kinds of dependent objects. But recovery action effect dependencies actually have three types of dependent objects: recovery actions, influenced degradations, and resulting degradations.

Fig. 4.91 shows the class structure used for modeling recovery effect dependencies between recovery actions and degradations. Recovery actions as dependent objects are specified by template expressions which determine sets of recovery action instantiations, i.e., instantiations of recovery action types with respective parameter values as discussed above. These template expressions can have free variables which can be further constrained to restrict the determined set of recovery action instantiations.

The specification of influenced or resulting degradations as dependent objects is reusing the specification of degradations used in impact dependency models treated in Sect. 4.3. That is, influenced and resulting degradations are specified by the means developed for the different types of impact dependency models, as e.g., BIDepMod(Char) for business degradations (Sect. 4.3.3), or e.g., SIDepMod(QoXInst/Subj:Fcty) (Sect. 4.3.8): Actually, these impact dependency models allow complex specifications of multiple degradation instantiations, and further restriction by constraints on free variables in the template expressions used for the specification.

So, free variables of recovery action specifications and degradation specifications can be constrained jointly allowing to express in a detailed manner all relationships among recovery actions, influenced and resulting degradations. This includes also relationship between influenced and resulting degradations, (e.g., in the case of reduced or aggravated degradations which have a version of it as influenced as well as resulting degradation). In general, there are

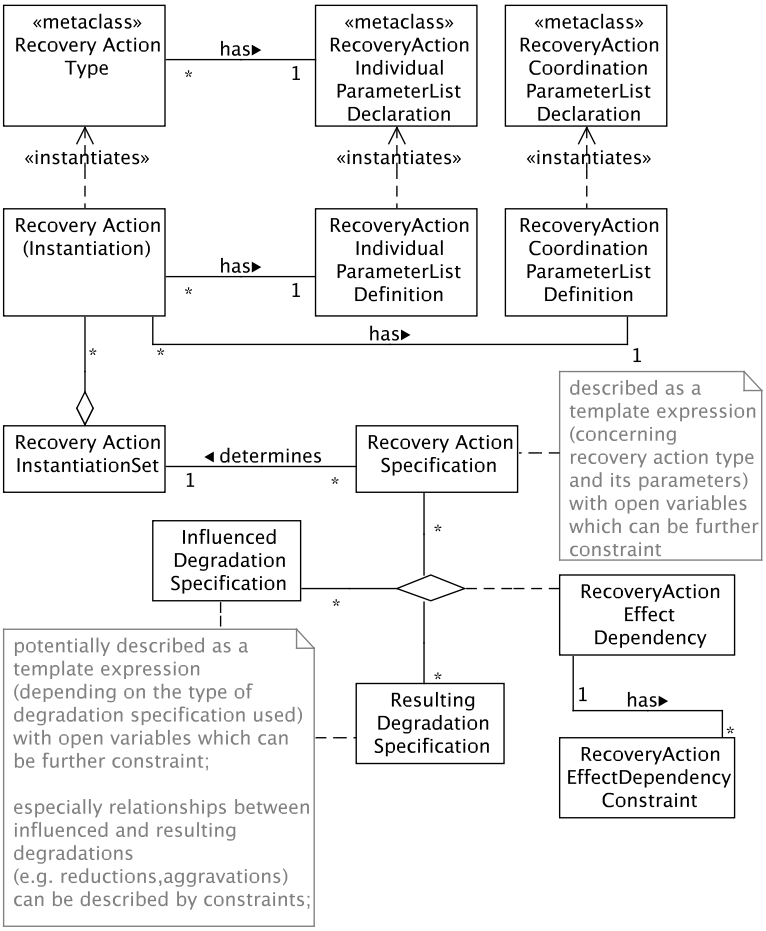

Figure 4.91: Design of recovery (action) effect dependencies between recovery action specifications and influenced/resulting degradation specifications (compare Fig. 4.90)

various types of influenced and resulting degradations identified already in Sect. 4.2.3 (see p. 157).

Some examples of dependencies for the example recovery plan $ExRecPlan1$ are illustrated in the following (compare p. 158 and p. 161):
reduction of g_{r2} by $RecAct(g_{r2}, 2)$:
$g_{r2}(duration = X) + RecAct(g_{r2}, 2) \rightarrow g_{r2\_post}(duration \leq 35 \text{ min})$,
aggravation of g_{s1-2} by $RecAct(g_{r2}, 2)$:
$g_{s1-2}(delay\_value = Y) + RecAct(g_{r2}, 2) \rightarrow g_{s1-2\_post}(delay\_value = Z)$,
with $Z > Y + 1$ s and (at least) $Z > 3$ s,
costs for $RecAct(g_{r2}, 5)$:
$RecAct(g_{r2}, 5)(price = P) \rightarrow g_{costs\_post}(value = P)$,
estimated benefit of $RecAct(g_{r2}, 5)$:
$RecAct(g_{r2}, 5)(price = P) \rightarrow g_{r2\_post}(duration < D)$,
with $P \in [1000 \ldots 10000]$€, and $D = f(P)$ for some time (estimation) function $f(.)$.
The argument sections of degradations above, e.g., $(duration : X)$ for g_{r2}, all denote QoX degradation type parameters for specifying the degradation

value accuracy and/or degradation time (compare Sect. 4.3.8). The argument section $(price : P)$ for $RecAct(g_{r2}, 5)$ illustrates the use of recovery action individual parameters.

The above introduced design for recovery action effect dependencies is still not taking into account an important aspect: The actual influence and result of a recovery action may depend on the actual success of the recovery action. In the case of the examples given above, e.g., the reduced degradation g_{r2_post} is only reached if the execution if $RecAct(g_{r2}, 2)$ succeeds. Consequently, a refined design of recovery action effect dependencies is developed in the following.

success parameter list declarations

The notion of *recovery action success parameter list declarations* has roughly already been introduced above on p. 301. For this notion an explicit design is now introduced, before actually targeting a refined design of effect dependencies. Fig. 4.92 presents a refinement of the class structure of Fig. 4.90

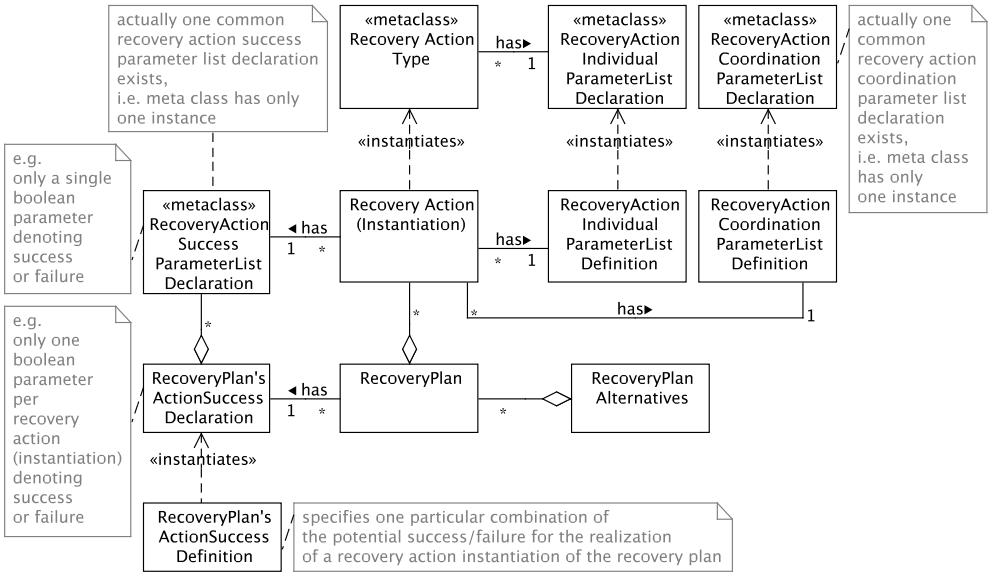

Figure 4.92: Design and modeling of the potential success/failure for the actual realization of recovery actions as part of a recovery plan alternative (refinement of Fig. 4.90)

which takes into account recovery action success parameter list declarations for describing the potential success/failure of the actual realization of recovery action instantiations, as part of a recovery plan alternative. The recovery action success parameter list declaration is common to all recovery action instantiations, similar as the recovery action coordination parameter list declaration is common to all recovery action types. But each actual realization of a recovery action instantiation has potentially multiple corresponding *recovery action success parameter list definitions* conforming to this declaration.

With respect to the particular proposal for a recovery action coordination parameter list declaration, $CoordParamListDecl1$, specifically a corresponding proposal for such a recovery action success parameter list declaration, $SuccParamListDecl1$ was introduced. $SuccParamListDecl1$ consists

only of one single boolean value denoting success or failure. But in general a recovery action success parameter list declaration may be more complex. In the case of the proposal, $SuccParamListDecl1$, there are only two possible recovery action success parameter list definitions, namely the two values *true* or *false* for the single boolean parameter declaration.

As motivated above, the various combinations of success or failure of each recovery action instantiation differentiate the possibilities of reduced impact of a single particular recovery plan alternative. As a subsumption for all combinations of the recovery action success parameter list declarations/definitions of all recovery action instantiations of recovery plan, the notions *recovery plan's action success parameter declaration* and correspondingly *recovery plan's action success parameter definition* for a whole recovery plan is introduced. The former one is common to all recovery plans, i.e., in the case of the proposal $SuccParamListDecl1$, simply one boolean success parameters per recovery action instantiation. The latter one is a particular combination of the potential value bindings of the former one, i.e., for $SuccParamListDecl1$ all true/false combinations for each recovery action (instantiation) of a recovery plan.

Based on the refined design of recovery action (instantiations) with success definitions, Fig. 4.93 shows the refined class structure for modeling recovery effect dependencies between recovery actions with success definitions and degradations. These refined type of effect dependencies have four kinds of dependent objects, i.e., additionally recovery action success parameter list definitions to take into account potential success/failure of a the specified recovery action. For the case of the proposed $SuccParamListDecl1$ this means, that actual success (value true) or failure (value false) for a recovery action is differentiated for determining the actual influenced/resulting degradations.

example

Now the above given examples of dependencies for the example recovery plan $ExRecPlan1$ can be specified in this refined manner, e.g., the (potential) reduction of g_{r2} by $RecAct(g_{r2}, 2)$:
$g_{r2}(duration = X) + RecAct(g_{r2}, 2)(success = true) \rightarrow$
$g_{r2\_post}(duration \leq 35min)$.

Whereas the afore-discussed type of effect dependencies (without considering success values) will be mainly used for constructing a recovery alternative, this refined type of effect dependencies (with taking into account success values) will be mainly used for the explicit determination of the resulting reduced impact of a recovery plan alternative as a kind of check and verification of the plan construction.

recovery action success and reduced impact

Having discussed the design of recovery action effect dependencies, i.e., dependencies from recovery actions (with success definitions) to influenced/resulting degradations, the design of reduced impact of a recovery plan alternative is treated. The output of recovery plan design is a list of potentially multiple recovery plan alternatives each with respective reduced (business) impact. But the actual reduced impact for such a single particular alternative may be different depending on the actual success of each of the ac-

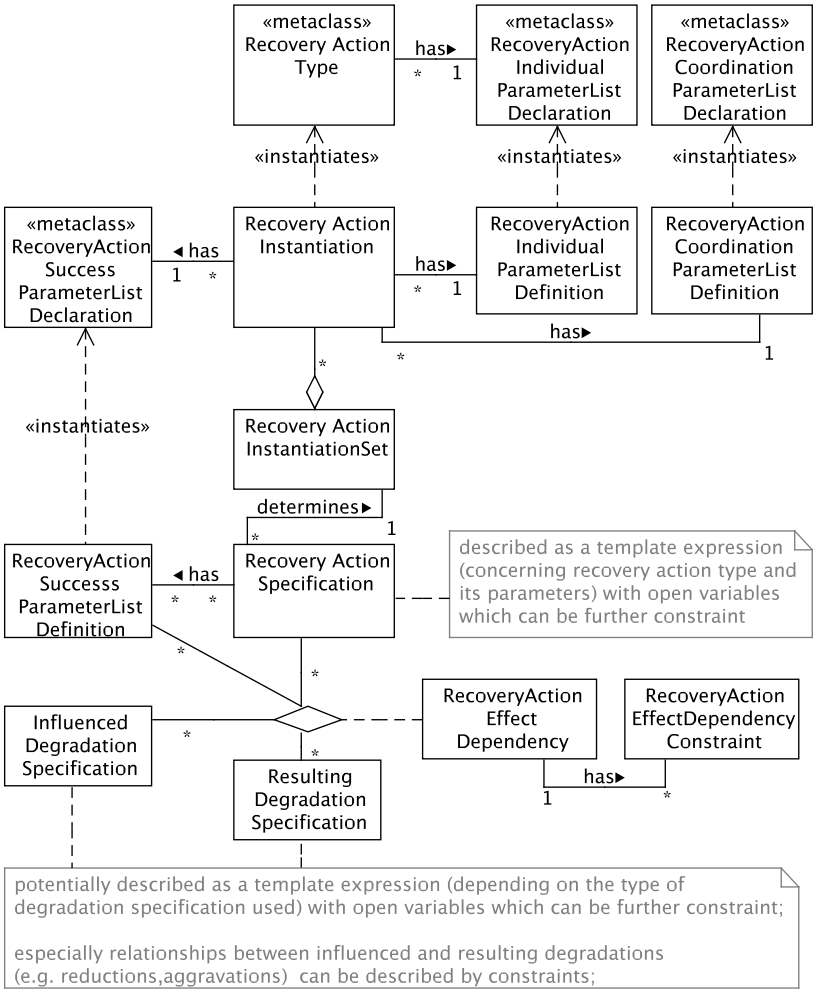

Figure 4.93: Design of recovery effect dependencies between recovery action specifications with success definitions, and influenced/resulting degradation specifications (refines Fig. 4.91)

tual realizations of its recovery actions. For instance, for the examples of $ExRecPlan1$ above, the success or failure of the subsequently tried recovery actions $RecAct(g_{r2}, 1)$, $RecAct(g_{r2}, 2)$, ..., $RecAct(g_{r2}, 5)$ determines at least the extent of reduction of g_{r2}, concerning duration and value. Furthermore, depending on the actual, conditional execution of recovery actions, e.g., $RecAct(g_{r2}, 2)$, further aggravations or additional costs may be introduced. That is why the reduced impact of a recovery plan alternative, similar as the recovery action effect dependencies it is actually derived from, is depending on the recovery action success parameter list declaration of each of its recovery action instantiations, i.e., the recovery plan's action success parameter declaration.

design of reduced impact

Conceptually, (pre-recovery) business impact (determined from IA) can be specified as a function of time or duration, whereas the reduced business impact of a particular recovery plan alternative has to be specified similarly as a function of time or duration, but one which is parameterized by the recovery plan's action success parameter declaration, as discussed above. The reduced

impact of a recovery plan may be different for each recovery plan's action success parameter definition for that recovery plan. Fig. 4.94 shows as an extension of the class structure of Fig. 4.92 the parameterization of reduced impact of a recovery plan by the recovery plan's action success parameter declaration.

Actually, reduced impact is specified as a rated business impact, which is derived from the recovery action instantiations of the recovery plan: first by following the recovery action effect dependencies (see above) to resulting degradations, second by following all degradation dependencies necessary to derive topmost business degradations from the resulting resource/service degradations, and finally by rating this determined impact in the same manner as the (pre-recovery) impact rating (Sect. 4.4.1). The rating is lastly done in order to allow a unification and comparison of the reduced impact of the different recovery plan alternatives.

Concluding, the output of recovery plan design is a list of potentially multiple recovery plan alternatives, each one consisting of one or multiple recovery action instantiations. The recovery actions instantiations have each a corresponding recovery action success parameter list declaration, which as whole, i.e., the recovery plan's action success declaration, parameterize the (rated) reduced business impact of the recovery plan, as a function of time/duration. This completes one particular possibility for the design of recovery action dependency models.

example

In the case of $ExRecPlan1$, specifically the handling alternative $DgrHndl(g_{r2}, 1+2+3+4+5)$ (p. 164), this results in the recovery plan's action success declaration $(boolean : success_1, \ldots, boolean : success_5)$. Consequently, the reduced impact for this alternative is function of time parameterized by that declaration.

refined design of recovery plan alternatives

But a closer look to Fig. 4.94 reveals a potential optimization for the representation of recovery plan alternatives. Often, different recovery alternatives are similar in their basic structure of recovery action instantiations, i.e., they have the same number and types of recovery actions, and they are only different concerning the actual individual/coordination parameter definitions for some/all recovery action instantiations. E.g., , for the handling alternative $DgrHndl(g_{r2}, 1+2+3+4+5)$ of $ExRecPlan1$ different actual individual parameters for recovery action $RecAct(g_{r2}, 1)$, i.e., number of tries to reboot and maximal time to try it, are possible. Moreover, there may be even unlimited (continuous parameter value spectrum) similar recovery plan alternatives. For such cases it is obviously difficult or impossible to actually enumerate explictly all such recovery plan alternatives explicitly (regarding the containment relationship between the classes *recovery alternatives* and *recovery plan* in Fig. 4.94. Such similar recovery plan alternatives may be better specified by template expressions for their recovery action instantiations, i.e., recovery action specifications as used already in Fig. 4.91, instead of particular, single recovery action instantiations. Fig. 4.95 presents correspondingly refined design for recovery plan alternatives described by *recovery plan specifications*.

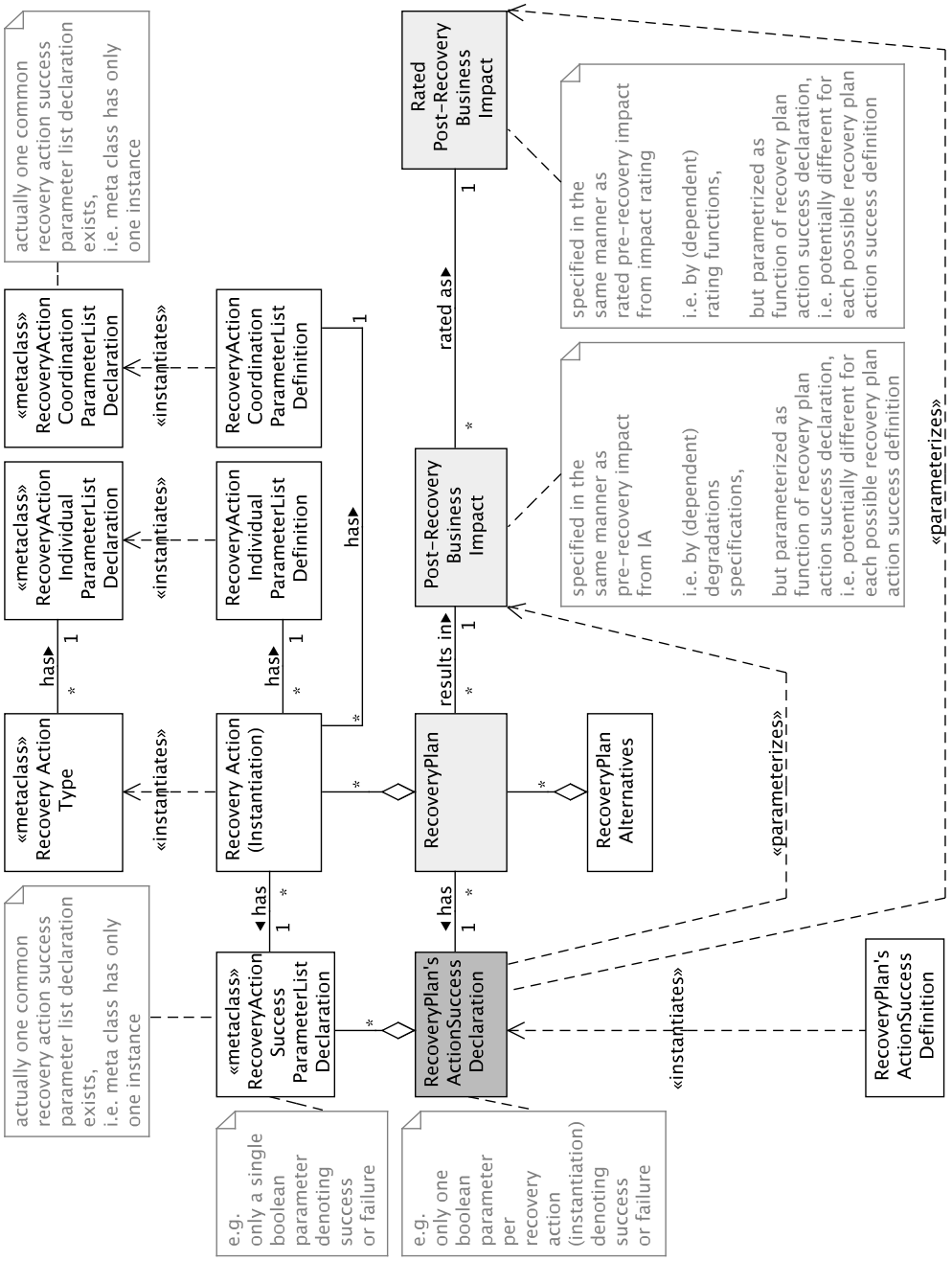

Figure 4.94: Design for reduced impact of recovery plan alternative, parameterized by recovery plan's action success the declaration

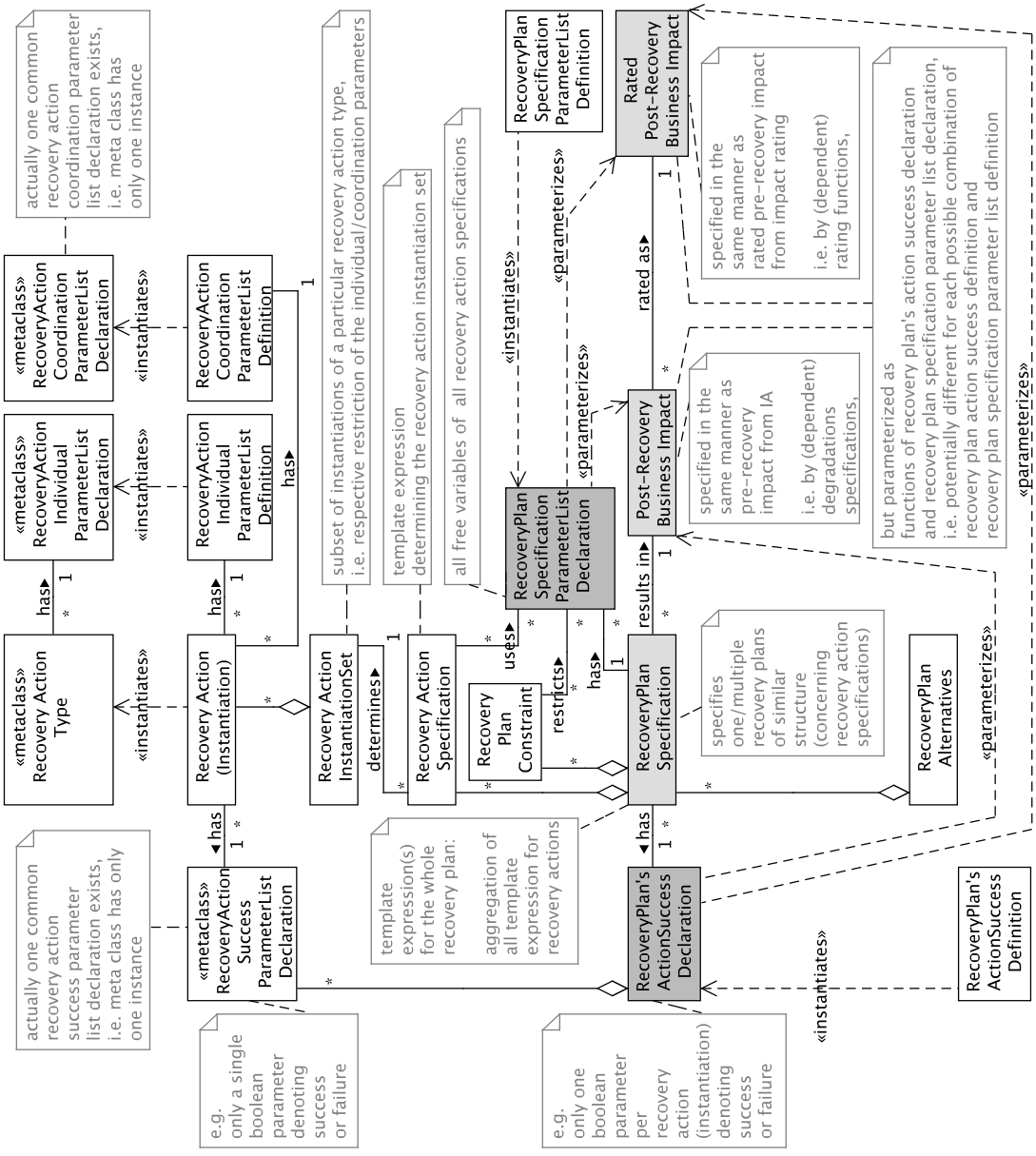

Figure 4.95: Design for recovery plan alternatives similar in structure with their reduced impact, parameterized additionally by recovery plan specification parameter list declaration (refinement of Fig. 4.94)

It thereby supersedes Fig. 4.90, Fig. 4.92, and Fig. 4.94 concerned with recovery plan design precedingly. A recovery plan specification is an aggregation of recovery action specifications, which in turn are template expressions determining a set of recovery action instantiations of a particular recovery action type. These template expression may have free variables, which aggregated as whole for a recovery plan, are subsumed under term *recovery plan specification parameter list declaration*. Such a declaration has *recovery plan specification parameter list definitions*, using actual parameter values, which conform to the declaration. Each recovery plan specification parameter list definition determines a recovery plan. The potential recovery plan specification parameter list definition can be further restricted by recovery plan (specification) constraints. As each such recovery plan subsumed under a recovery plan specification conforms to a similar structure concerning the recovery actions, they all share the same recovery plan's action success declaration.

Concluding, a recovery action specification as subsumption of recovery plans similar in recovery action structure, has a recovery plan specification parameter list declaration (differentiating its similar recovery plans) and a single recovery plan's action success declaration (differentiating the success/failure combinations for the realizations of all its recovery actions). Consequently, the (rated) reduced impact of a recovery plan specification as a whole, is parameterized not only by the recovery plan's action success declaration (as valid for the more simple design of recovery plans in Fig. 4.94), but is also parameterized by the recovery plan specification parameter list declaration. Nevertheless, the output of recovery plan design may comprise multiple recovery plan specifications, each with a different structure concerning their recovery action (instantiations).

For example, if the recovery alternative $DgrHndl(g_{r2}, 1+2+3+4+5)$ (p. 164) varies concerning the actual individual parameter values of recovery action $RecAct(g_{r2}, 1)$, the recovery plan specification itself is additionally parameterized by this varying parameters, e.g., $(integer : reboot\_tries, time : max\_total\_duration)$ with additional constraint $reboot\_tries \in \{3 \dots 10\}$, as its recovery plan specification parameter list declaration. Consequently, the reduced impact of the alternative as a function of time is parameterized by this declaration in addition to its recovery plan's action success declaration $(boolean : success_1, \dots, boolean : success_5)$ (compare above). example

4.4.3 Implementation of impact rating models and recovery action dependency models

Having designed impact rating models (Sect. 4.4.1) and recovery action dependency models (Sect. 4.4.2), guidelines for their actual realization, mainly with respect to the basic component architecture, BCArch, and its basic realized workflow, BRWf, (see Sect. 4.2.5), are discussed in the following. Similarly as done for IA in Sect. 4.3.11, particular implementation techniques for the realization of the respective architecture components and the related,

used data structures are introduced. First, the whole design of both types of RA models and related architecture components is summarized, and based on this guidelines for the realization by respective, concrete implementation techniques are given.

summary on recovery analysis models and their usage

Recovery analysis models, i.e., impact rating model as well as recovery action dependency model, being the key factors to an appropriate recovery analysis, have been initially introduced in Sect. 4.2.3.2 on p. 171 as part of the basic refined abstract workflow (BRAWf). In the basic realized workflow (BRWf, Sect. 4.2.5.6) the basic realization of recovery analysis, and the respective usage of recovery analysis models was discussed on p. 206 (BRWf.2.1) as well as on p. 207 (BRWf.2.2). There, the two steps of recovery analysis, i.e., impact rating (BRWf.2.1, basic realized impact rating subworkflow) and actual recovery plan design (BRWf.2.2, basic realized recovery design subworkflow) are performed by two particular *I/RA modules* (compare p. 199), namely the *impact rater* and the *recovery planner*. These particular parts of the I/R analyzer (Sect. 4.2.5.5) utilize the respective type of recovery analysis model for their tasks: The former component, the impact rater, uses the impact rating model for direct rating of top-level business degradations, in combination with the indirect rating of entailing degradations by re-use of the impact dependency models. Actually, for re-using the impact dependency models for this purpose, their degradation dependencies are used in the reversed direction, i.e., from target degradations to source degradations, or rather the information about particular current degradations and their dependencies (degradation derivation tree, compare p. 286) which has already been derived during IA from the impact dependency models and has been remembered until RA, is accessed and reused (in reversed direction) for indirect impact rating. The latter component, the recovery planner, uses the recovery action dependency model to construct the actual recovery plan alternatives along with their estimated, reduced business impact based on the (indirectly) rated, initial resource degradations.

In Sect. 4.4.1 and Sect. 4.4.2 one particular possibility for a design of both recovery analysis models including all related data structures, i.e., concerning the specification of rated impact, recovery actions, recovery plans, reduced impact of a recovery plan, was developed. This design of RA models represents a high refinement of the basic introduction given in Sect. 4.2.3 for RA models and related data structures (compare Fig. 4.21 on p. 165, Fig. 4.24 on p. 172, and Fig. 4.25 on p. 174). Moreover, it was taken care that the design of RA models is compatible with the design of impact dependency models in the impact analysis framework (Sect. 4.3), i.e., namely with the design of degradations and their dependencies.

towards a concrete implementation

Having summarized the design of recovery analysis models and their related components of the basic component architecture, namely the impact rater and the recovery planner as a particular I/RA modules, in the following the actual realization guidelines by respective, concrete implementation techniques are discussed. Especially in the case of recovery planning, different types of im-

plementations may be considered, each based on different existing methods or techniques, such as RBR (p. 103 in Sect. 3.6) or CBR (p. 108 in Sect. 3.6.3). First, the implementation of impact rating is discussed, second the implementation of recovery planning.

impact rating implementation

Impact rating is subdivided into direct rating of top-level business degradations, and the indirect rating of (underlying business degradations), service/resource degradations all of them eventually entailing the top-level business degradations. The former task is relatively easy, it mainly comprises the conversion of functions, namely the unification of business degradation characteristics to unified direct degradation rating functions. One alternative for actually performing such conversions may be RBR, i.e., logical reasoning. Anyway, similar function conversions may also have been contained in the impact dependency models (e.g., from the QoX metric of one type of degradation to another entailed one, described by appropriate dependency constraints) which are realized by an appropriate RBR approach (compare p. 287 in Sect. 4.3.11). So the implementation of direct impact rating may simply copy from this for the specific case of function conversion to unified direct degradation rating functions. As the latter task is re-using the remembered degradations derivation tree which is actually calculated by an deductive database engine (particular type of RBR) representing the core of the impact analyzer (compare p. 289 in Sect. 4.3.11), it is at least one possibility to also use RBR for this task. The RBR can then directly re-use the remembered degradation derivation tree (in the reverse direction as it has been constructed, from targets to sources).

The example implementation for impact dependency models in Appendix B, using logical deductive facts and rules in the Flora2 deductive database system, also provides some examples how direct and indirect rating of degradations can be implemented as generally described above.

recovery planning implementation

For recovery planning multiple implementation options are available. One option, which is similar to and therefore directly compatible with the option chosen for impact analysis, i.e., a deductive database approach as a logics/RBR approach, is also to use a logics/RBR approach for the recovery planning. One particular possibility for this is the use of Transaction Logic (see Sect. 3.6.2), which is also designed to generically address planning problems. The basic feature of Transaction Logic is the support of so-called backtrackable logical updates, i.e., the hypothetical updates (insertion and deletion) of new, potentially valid logical facts as part of the reasoning process. This way, the power of Transaction Logic goes beyond the basic derivation - in pure deductive manner - of already valid derivable facts (e.g., encoding entailed degradations) from given base facts (e.g., encoding the initially given resource degradations) and given logical rules (e.g., encoding degradation dependencies of the impact dependency models) During the reasoning by transaction logic new hypothetically valid facts (e.g., encoding actual recovery actions of a particular recovery plan alternative with particular recovery action type and parameters in order to determine aspects of them such as scheduling,

specific effort, duration) can be added (temporarily) as valid facts. In the remaining course of the reasoning their consistency with other existing facts and rules (e.g., encoding initially given resource degradations, all degradation dependencies as above, as well as all dependencies of the recovery action dependency model) can be checked, and potentially their original hypothetical insertion marking them as valid is canceled if inconsistency is determined. That is the logical search tree is consistently backtracked to the point where the logical insertion was made, including their removal from the set of valid facts.

As an alternative option, completely different methods (instead of RBR) may be chosen for the implementation of the recovery planning, e.g., the use of CBR (compare p. 108 in Sect. 3.6.3), a type of analog-like reasoning. Furthermore, mathematical optimization methods as used in [BS04, SB04], or the usage of simulation techniques might be adapted for recovery planning. But in order to actually use such other methods, such as CBR, mathematical optimization methods, or simulation techniques an appropriate adaption and conversion of the modeled data (of SI and BI impact dependency model, rating model, and of recovery action dependency model) into the particular domain of the technique used has to be performed. The detailed description of the use of Transaction Logic, as well as such further methods for recovery planning goes beyond the scope of this thesis.

4.5 Recovery Tracking Framework

This section treats the *recovery tracking framework* (*RTFw*) as the third and last extension framework of the *basic framework* (*BFw*). Fig. 4.96 as a reminder of Fig. 4.1 again shows the overview of the I/RA framework specifically marking the recovery tracking framework. The basic framework (see Sect. 4.2) provides a generic basis for I/R analysis by introducing basic terms, concepts, and workflow steps for I/R analysis in general. The RTFw extends, refines, and details these basic notions, concepts, and workflow steps introduced particularly for recovery tracking (compare Fig. 4.1 on p. 122). Therefore, the RTFw is concerned with the design and realization of actual data structures, and their specific usage in the respective workflow steps.

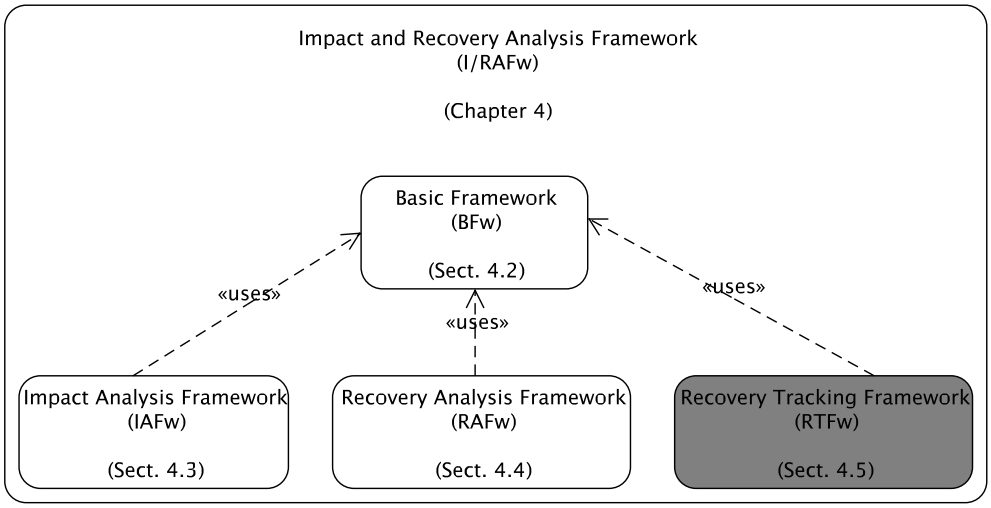

Figure 4.96: Recovery tracking framework (RTFw) as third extension framework of the basic framework (reminder of Fig. 4.1)

recovery changes tracking + model adaption

RT dependency models

As having been presented in Sect. 4.2.4.1 on p. 182, recovery tracking (RT) is basically subdivided into *recovery changes tracking* and subsequent *model adaption*. In Sect. 4.2.3.2 the *recovery changes tracking dependency model* for the purpose of recovery changes tracking, and the *model adaption dependency model* for the purpose of model adaption were introduced. Both models have been subsumed under the term *RT dependency models*.

basic RT subworkflows

The usage of both RT dependency models has been treated in abstract manner in Sect. 4.2.4.2 (basic refined abstract recovery tracking subworkflow, BRAWf.3) as well as in Sect. 4.2.5.6 on p. 209 (basic realized recovery changes tracking subworkflow, BRWf.3.1), and on p. 210 (basic realized model adaption subworkflow, BRWf.3.2).

recovery changes tracking dependency model

The recovery changes tracking is basically concerned with the monitoring and pre-aggregation of recovery changes resulting from the actual execution of the selected recovery plan (essential input of recovery tracking, compare Fig. 4.34). The notion of recovery changes has been treated in abstract manner in Sect. 4.2.4.1 on p. 180. The used *recovery changes tracking dependency*

model (*RecChgTrckDepMod*) serves two purposes during recovery changes tracking (compare p. 185 as well as p. 209): First, it allows to derive which types of recovery changes have to be monitored during or after the realization of the selected recovery plan. Second, it determines how these collected individual recovery changes are pre-aggregated, so they can be given as consistent input to model adaption.

model adaption dependency model

The model adaption dependency model is used during model adaption to derive from the aggregated recovery changes the *model updates* which are necessary to ensure complete and accurate synchronization of all models used by I/R analysis with the information about actually occurred recovery changes. Thus, as all models for I/R analysis are extracted from information sources in the existing SP/MI (service provisioning and management infrastructure), model adaption actually ensures that these information sources are really and completely up-to-date and synchronized with the information of occurred recovery changes. This includes check model actions, and appropriate actual model update actions in case of mismatch. Model adaption is more specifically performed by constructing and realizing a model adaption plan which comprises and coordinates all necessary model updates (compare p. 210).

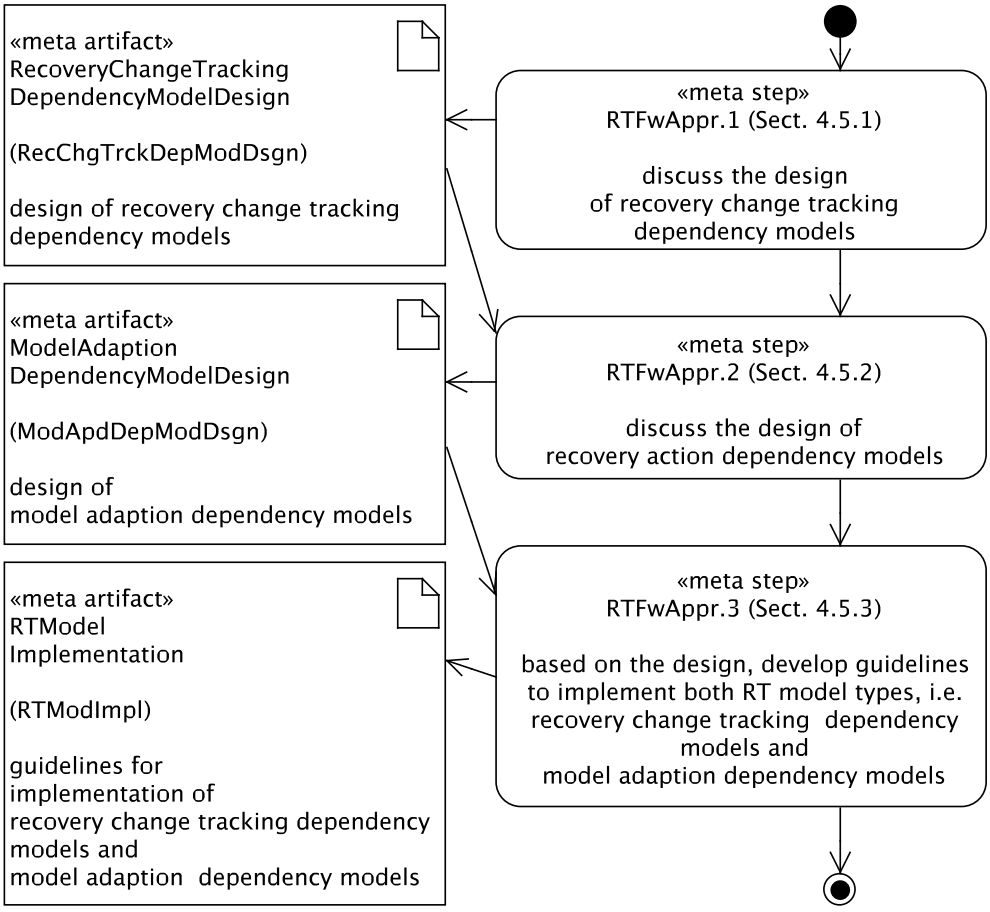

Figure 4.97: Approach (RTFwAppr) for the development of the recovery analysis framework (RTFw)

concretization of both RT models

Concluding, the above referenced basic introduction of both RT dependency models in the basic framework treated them only on a rough, abstract level. Also the used notions of recovery changes, as well model adaption plans, were only described on such an abstract level. Therefore, the RTFw is mainly concerned with the concretization, with respect to actual data structures and their actual usage, of both RT dependency models. This concretization further comprises the specification of actual data structures for recovery changes, model adaption plans, and their relationship to the selected recovery plan and its recovery actions. That is, the design and the subsequent implementation of both RT dependency models, which are compatible with the design and implementation of impact dependency models of IAFw (Sect. 4.3) and RA models of RAFw (Sect. 4.4), is main subject of the RTFw.

approach for RTFw development

Fig. 4.97 illustrates the approach for the development of the recovery tracking framework (RTFwAppr). First, the design of both types of RT models is discussed in Sect. 4.5.1 and Sect. 4.5.2. Based on this, guidelines for the implementation of RT dependency models are introduced in Sect. 4.5.3. These guidelines treat also the actual usage of the implemented RT dependency models as part of the basic component architecture introduced for the basic realized workflow BRWf (see p. 209 for BRWf.3.1, and p. 210 for BRWf.3.2).

4.5.1 Design of recovery change tracking dependency models

Following is a discussion on the design of recovery change tracking dependency models as used for recovery changes tracking. First a design and modeling of recovery changes is introduced, and based on this the proper design of the dependency models is treated.

recovery changes

Recovery changes have been discussed in abstract manner in Sect. 4.2.4.1 on p. 180. A recovery change is the conceptual description of a change concerning the resources or services, which is resulting from the actual realization of the selected recovery plan. Eventually such recovery changes have to be reflected in all models for I/R analysis in order to ensure a proper and accurate future I/R analysis.

A recovery change primarily manifests in updates of the respective data sources/sinks containing information concerning the changed resources in the SP/MI. But as these data sources/sinks in the SP/MI are the original base of all data contained in the respective models for I/R analysis, also all corresponding information parts in these models concerned with the changed resources/services have to be updated, too. Nevertheless, for I/R analysis it is necessary to ensure that this update of model information is happening in a timely, complete, and accurately synchronized manner.

Recovery tracking is subdivided into three steps, *individual recovery changes monitoring configuration determination*, *individual recovery changes monitoring*, and *recovery changes aggregation* (compare p. 185). In the first step,

the actual types of recovery changes to be monitored, i.e., the actual information sources to subscribe to as well as the proper subset of information to subscribe to, is derived from execution information of the selected recovery plan. For this purpose a first type of *recovery changes tracking dependencies*, namely the *execution information to recovery change type monitoring configuration mapping* is used. In the second step, during the actual realization of the recovery, the actual individual recovery changes are monitored by subscribing to respective information sources in the SP/MI. After the completion of the recovery, in the third step, all individual recovery changes are pre-aggregated (e.g., filtered) in order to provide a consistent input for model adaption, which actually will make sure that all I/R models are accurately synchronized. This aggregation is determined by a second type of dependencies, the *recovery change aggregation dependencies*. Both type of dependencies are part of recovery changes tracking dependency models.

design of recovery changes

Consequently, a design of recovery changes has to support a differentiation of multiple types of recovery changes, e.g., for the purpose of monitoring configuration of particular subsets of recovery changes. But furthermore, the design should support a refined notion to specify particular recovery changes of any recovery changes type, for specifying actual individual recovery changes monitored, or recovery changes aggregated from the former ones. That is why for the design of recovery changes, also an approach using types with parameter list declarations, and instantiations with conforming parameter list definitions is chosen. This is similar to the approaches for degradation subject instantiations and for QoX degradation instantiations in Sect. 4.3, or for recovery actions in Sect. 4.4. Fig. 4.98 presents the resulting class structure

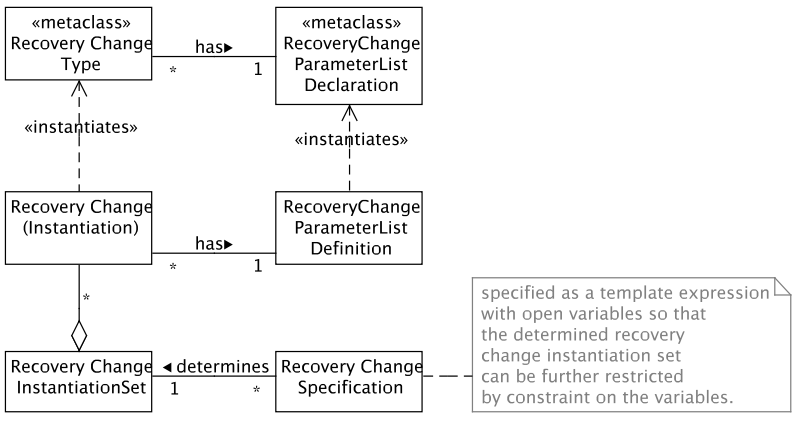

Figure 4.98: Design of recovery changes specified by types/instantiations with parameters

for the design and modeling of recovery changes. Recovery change types are determined by a corresponding parameter list declaration, i.e., a list of named parameter type declarations. Recovery change instantiations of a particular recovery action type have a corresponding parameter list definition conforming to the parameter list declaration of their type.

examples

Some examples concerning $ExSit1$ and its recovery plan $ExRecPlan1$ (see p. 180) are given in the following: basic recovery change types are *AFS device change* related to recovery action $RecAct(g_{r2}, 2)$, *IP link replacement* related to recovery action $RecAct(g_{r1}, 3)$, *IP routing change* related to recovery action $RecAct(g_{r1}, 2)$. Each of these ones has appropriate parameters, e.g., the device address and reachable configured performance for the first one, the changed path, related IP addresses, estimated available bandwidth for the second one, connected routers, link bandwidth, used technology for the third one. Moreover, each of these basic recovery change types has to be more differentiated into corresponding individual and aggregated recovery change types. For instance, the *AFS device change* has the associated individual recovery change types *afs device removed*, *new afs device*, *new IP device*, *new IP port* which are actually monitored during recovery e.g., from the AFS configuration, by SNMP traps, and by contacting the IP configuration database, and are afterwards aggregated to one particular *aggregated AFS device change* reflecting the complete replacement of the broken AFS device. This aggregation includes correlation of various address types, such as AFS device number, IP address, MAC address, connected port and router.

design of dependencies

Having introduced the design of recovery changes, the design for both types of dependencies of the recovery changes tracking dependency model is treated. Both types of dependencies were basically introduced in Sect. 4.2 (see p. 185 as well as p. 209), and have been shortly summarized already above. The following design of the dependencies is based on Fig. 4.51 on p. 217, which introduced a generic outline for the modeling of dependencies in general. Fig. 4.99 gives an overview of the classes for the design of both types of recovery changes tracking dependencies.

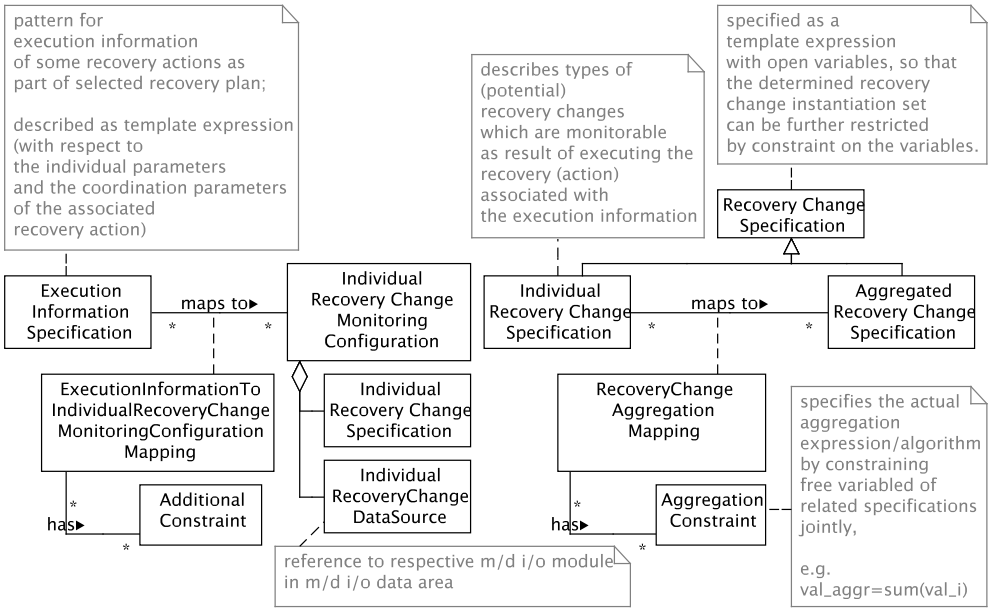

Figure 4.99: Design for recovery change (tracking) dependencies of both types

The *execution information to individual recovery change type monitoring configuration mapping* (*Type I recovery change dependency*) is a dependency between two kinds of dependent objects, namely the *recovery action execution information specification* and *individual recovery change monitoring configuration*. The former one is similar to the recovery action specifications for recovery action effect dependencies in Sect. 4.4.2 (compare Fig. 4.91 on p. 304), at least conceptually: Execution information of recovery actions is the external representation (instructions for operators to realize the recovery) of an recovery action, whereas the recovery action specification for recovery action effect dependencies in Fig. 4.91 used the representation internal to I/R analysis. But both representations are parameterized by the same type of parameters (recovery action individual ones, and ones for coordination), and the basic relationship between them is actually covered by the (left-hand side) *has* relationship in Fig. 4.90 on p. 300, which introduced the design of recovery actions. The latter one of the dependent object kinds, the individual recovery change monitoring configuration, comprises a (individual) recovery change specification (as introduced in Fig. 4.98) and a specification of the monitoring data source in the SP/MI. The monitoring data source is actually specified as a reference to the respective model/dynamic i/o data module which is used for the data exchange between the data source in the SP/MI and the I/R component architecture (see Sect. 4.2.5.4). The dependency additionally has constraints which constrain jointly the free variables in the template specifications of both, the recovery action execution information specification, as well as the (individual) recovery change specification, restricting their parameter values further.

The *recovery change aggregation mapping* (*Type II recovery change dependency*) is also a dependency between two kinds of dependent objects, namely its *individual recovery change specification* and its *aggregated recovery change specification*, both are template expressions as introduced in Fig. 4.98. For actually specifying the aggregation expression/algorithm, the dependency includes aggregation constrains for restricting jointly the free variables in the template expressions of these related recovery change specifications.

example

For illustration the examples from above are continued: First, the execution information for recovery action $RecAct(g_{r2}, 2)$ is mapped by particular Type I recovery change dependencies to corresponding m/d i/o modules for configuring and performing the monitoring of the respective individual recovery types, i.e., *AFS device removed*, *new AFS device*, *new IP device*, *new IP port* (see above). A restriction to particular parameter value ranges for these respective individual recovery types is included in the respective specifications, e.g., for *new IP device* a restriction to new IP devices within the subnet used by AFS devices. Second, particular Type II recovery change dependencies specify how to aggregate and consolidate these collected individual recovery changes.

4.5.2 Design of model adaption dependency models

Here the design of model adaption dependency models is discussed. First the design of model adaptions is treated, second the design of model adaption dependencies itself follows.

purpose of model adaption

The basic purpose of model adaption is to derive from the aggregated recovery changes, which are given as input by recovery changes tracking, model updates or model synchronizations to be performed.

This actually comprises two different subtasks: On the one hand, to ensure properly and timely updated data source in the SP/MI, on the other hand, based on this, the coordinated update of currently loaded model/dynamic data in I/R components (specifically the I/RA model/dynamic database, Sect. 4.2.5.2 on p. 196). The first task should ideally be not necessary, if the existing management processes of the service provider in the SP/MI are perfectly working and synchronized with the I/R analysis architecture. But for cases where the existing management processes of the SP/MI are not accurate enough, at least in terms of timeliness and fast synchronization, for the I/R analysis, it is nevertheless included in this discussion here.

subdivision of model adaption

The performing of model adaption as workflow is generally subdivided into two steps, model adaption planning and model adaption execution (compare Sect. 4.2.5.6 on p. 210). The former step determines a plan for the model adaption to be actually done with appropriate coordination, the latter step is concerned with the actual realization of this plan. Consequently, in Sect. 4.2.5.6 the general notion of particular *model adaptions* (or *model updates*) to be performed has been introduced. Examples of such particular model adaptions related to the example recovery plan $ExRecPlan1$, more specifically to the aggregated recovery change *AFS device change* (see p. 320), are the updating of the respective service degradation dependencies affected by the AFS device replacement, as well as updating of the recovery action dependency model as subset of potential backup devices now has been shrunk.

example

The notion of model adaption is now elaborated in greater detail, and afterwards an appropriate design is introduced. The result of model adaption planning is explicitly designated as a *model adaption plan*, which is comprising one or multiple *model adaption actions*. This way, model adaption plans can be specified and modeled in a similar manner as recovery plans: i.e., model adaption plans correspond to recovery plans, and model adaption actions correspond to recovery actions. That is why similar data structures can be used for the specification of both. The basic difference is that recovery plans are designed in potentially multiple alternatives for external realization by human operators, whereas for model adaption plans not multiple alternatives are considered, and that model adaption plans are designed ideally for automatic execution by the I/R analysis architecture. Moreover, no complex trade-offs have to be considered for the planning of model adaption plans. This planning only has to ensure timely synchronized and accurate model adaptions.

Nevertheless, model adaption actions as part of model adaption plans can be described, similarly as recovery actions, by model adaption types with individual and common coordination parameters. Fig. 4.100 shows the class structure for the design and modeling of model adaptions by model adaption plans consisting of model adaption actions (compare Fig. 4.90 on p. 300 for recovery actions and plans). Here, model adaption actions are modeled

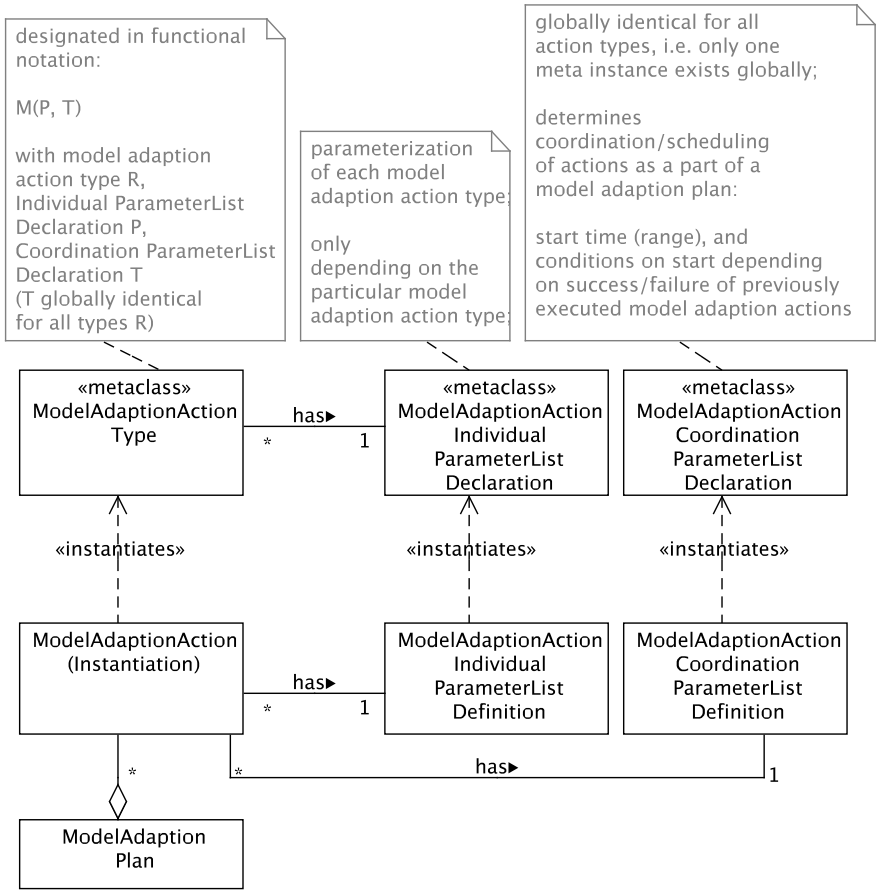

Figure 4.100: Design of model adaption plans consisting of model adaption actions of different types (compare design of recovery plans/actions in Fig. 4.90)

by *model adaption action types* with parameter list declarations, i.e., lists of named parameter type declarations. Each model adaption action type can have multiple instantiations. Each *model adaption action instantiation* of a model adaption action type has corresponding parameter list definitions, conforming to the parameter list declaration of its type. Lastly, a model adaption plan is consisting of one or multiple model adaption action instantiations.

example

The model adaption plan for example recovery plan $ExRecPlan1$ may be designated as $ExModAdpPlan1$. As example, the part of $ExModAdpPlan1$ concerned with the respective model updating for the aggregated recovery change *AFS device change* is illustrated here. As introduced already above, the consolidating of I/R models with *AFS device change* basically comprises two tasks: updating and reloading of respective service

degradation dependencies, as well as updating and reloading of respective recovery action degradation dependencies. The latter task comprises only checking and ensuring the update of the inventory database in order to reflect the shrunken subset of future, backup devices (task $ModAdp_{\text{invent\_db}}$). The former task is subdivided into subtasks:

- task $ModAdp_{\text{afs\_ip\_conf}}$: Checking and ensuring the update of all relevant information in the provider's IP configuration database (concerning the new AFS device) as this is used in this example as source for resource/service degradation dependencies of the IP service (subservice of AFS service).
- task $ModAdp_{\text{afs\_path\_conf}}$: Determining the AFS paths and directories stored (or to be stored) on the new device as this information is used for determining (refined) subject degradation dependencies for the AFS service, e.g., the mail folders of which users are stored on this device, which web pages are stored there.
- task $ModAdp_{\text{afs\_throughput}}$: Checking or initiating immediate as well as regular throughput tests (active or passive) because the measurement results are used for deriving QoX degradation types dependencies for the AFS service.

$ExModAdpPlan1$ comprises an appropriate subset of coordinated model adaption action instantiations, $ModAdpActSet1(task)$, with corresponding coordination/individual parameters to cover each specific tasks. $ModAdpActSet1(task)$, task $\in \{ModAdp_{\text{invent\_db}}, ModAdp_{\text{afs\_ip\_conf}}, ModAdp_{\text{afs\_path\_conf}}, ModAdp_{\text{afs\_throughput}}\}$ will be discussed below in more detail.

Having discussed the design of model adaptions, the design of *model adaption planning dependencies*, as being the essential content of model adaption dependency models, are now treated. These dependencies are involved only in the planning of the model adaption plan. Later on, for the actual realization of the model adaption the constructed plan is used directly.

In Fig. 4.51 on p. 217 a generic outline for the design and modeling of dependencies in general was introduced, which is also used for model adaption dependencies. Fig. 4.101 shows the class structure for the design of model adaption planning dependencies. These dependencies have two kinds of related objects, (aggregated) recovery change specifications (compare Fig. 4.98), and a model adaption specification. The former one is a template expression determining the set of (aggregated) recovery change instantiations, and can be further restricted concerning its free variables (referring recovery change parameters) by additional constraints. The later one is also a template expression determining the related set of model adaption instantiations (as introduced above, i.e., the constituents of a model adaption plan to design). The later one also can be further restricted concerning its free variables (referring to model adaption action parameters) by additional constraints. Actually, the free

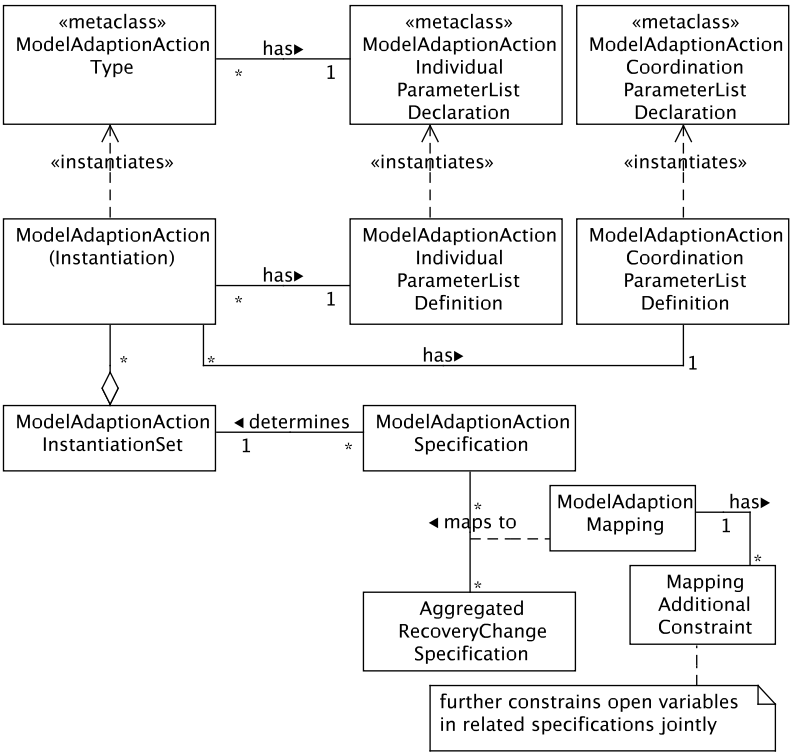

Figure 4.101: Design of model adaption (planning) dependencies between aggregated recovery action specifications and model adaption action specifications

variables of both kinds of dependent objects can be jointly restricted by such additional constraints of the dependency.

example

Examples of model adaption planning dependencies are the dependencies from aggregated recovery change *AFS device change* to the appropriate subset of coordinated model adaption action instantiations, $ModAdpActSet1(task)$ (see above). They will be elaborated in more detail below, after a more deep classification of model adaption action types has been discussed.

planning and execution of model adaptions

Model adaption actions are modeled similar as recovery actions. For recovery actions and their parameterization, basically two different types were distinguished: internal parameterizations of recovery actions suitable for search and optimization in order to construct recovery action plans, and external parameterizations or so-called recovery action execution information (see p. 300). Consequently, a similar differentiation can be made for model adaption actions: planning parameterization and execution parameterization. The *planning parameterization of model adaption actions*, corresponding to the internal parameterization of recovery actions, is the parameterization and modeling as introduced above. It is suitable for search and optimization algorithms in order to construct the model adaption plan. In contrast, the *execution parameterization of model adaption actions* or *model adaption action execution information*, is suitable for the (automated) execution of the model adaption plan by the m/d i/o data access area (see Sect. 4.2.5.4). Both representation are parameterized by similar coordination and individual action parameters, but

model adaption execution information is focused on the particular execution by m/d i/o data access area, i.e., by appropriate m/d i/o modules with appropriate data exchange parameters for these modules. Both representations are in this respect internal to I/R architecture. That is why the designation attributes *planning* and *execution* were chosen instead of *internal* and *external* as for recovery actions, whose actual execution is out of scope of the I/R architecture.

model adaption execution dependencies

Moreover, as already basically mentioned in Sect. 4.2.5.6 on p. 211, the actual execution of a particular model adaption action by an appropriate m/d i/o module with corresponding execution parameters has (potentially highly dynamic) execution dependencies with respect to its result in terms of actual specific I/R model data parts which are changed and updated corresponding: That is, the updates of the dependencies contained in the various model used for I/R analysis are in a sense (dynamically) dependent on the (actual realized) execution of model adaption actions. These dynamic execution dependencies are called *model adaption execution dependencies*, in contrast to the model adaption planning dependencies described above. Although conceptually they can be regarded as part of the model adaption dependency model, they are not specified explicitly, but only encoded into the respective m/d i/o modules and their actual execution. A concrete design of particular m/d i/o modules with respect to their usage for model adaption has to make sure that these execution dependencies are correctly encoded in these m/d i/o modules. Thus, the specification and respective up-to-dateness of these execution dependencies represent an open issue which mostly goes beyond the possibility of automated synchronization by the I/R analysis architecture: If changes on the respective m/d i/o modules are necessary to encode execution dependencies which have changed after an actual recovery, these changes necessary can mainly only be performed by human operators. Nevertheless, the respective m/d i/o modules may be designed in such a way that they at least detect that such changes on them itself are necessary, and maybe additionally propose possible solutions, so that at least support human operators concerning their update. An investigation regarding these issues, i.e., the specification/encoding, out-of-dateness detection, and the proposal of changes on m/d i/o modules with respect to (changed) model adaption execution dependencies, is left open as a further research issue.

classification of model adaption action types

Following is a classification of model adaption action types (compare introduction for BRWf in Sect. 4.2.5.6 on p. 210). Fig. 4.102 shows an overview of this classification. As introductory already mentioned above model adaption involves two tasks: first, the ensuring of properly and timely updated data sources in the SP/MI, and second, the proper coordinated update of currently loaded model/dynamic data in I/R architecture components from these data sources/sinks in the SP/MI. Ideally, the first task should be not required, and should already be covered by the existing management processes in the SP/MI. Thus, it is only meant as an additional support for the service provider to really ensure an overall, timely accurate synchronization in the SP/MI with the occurred recovery changes. The second task specifically targets at the

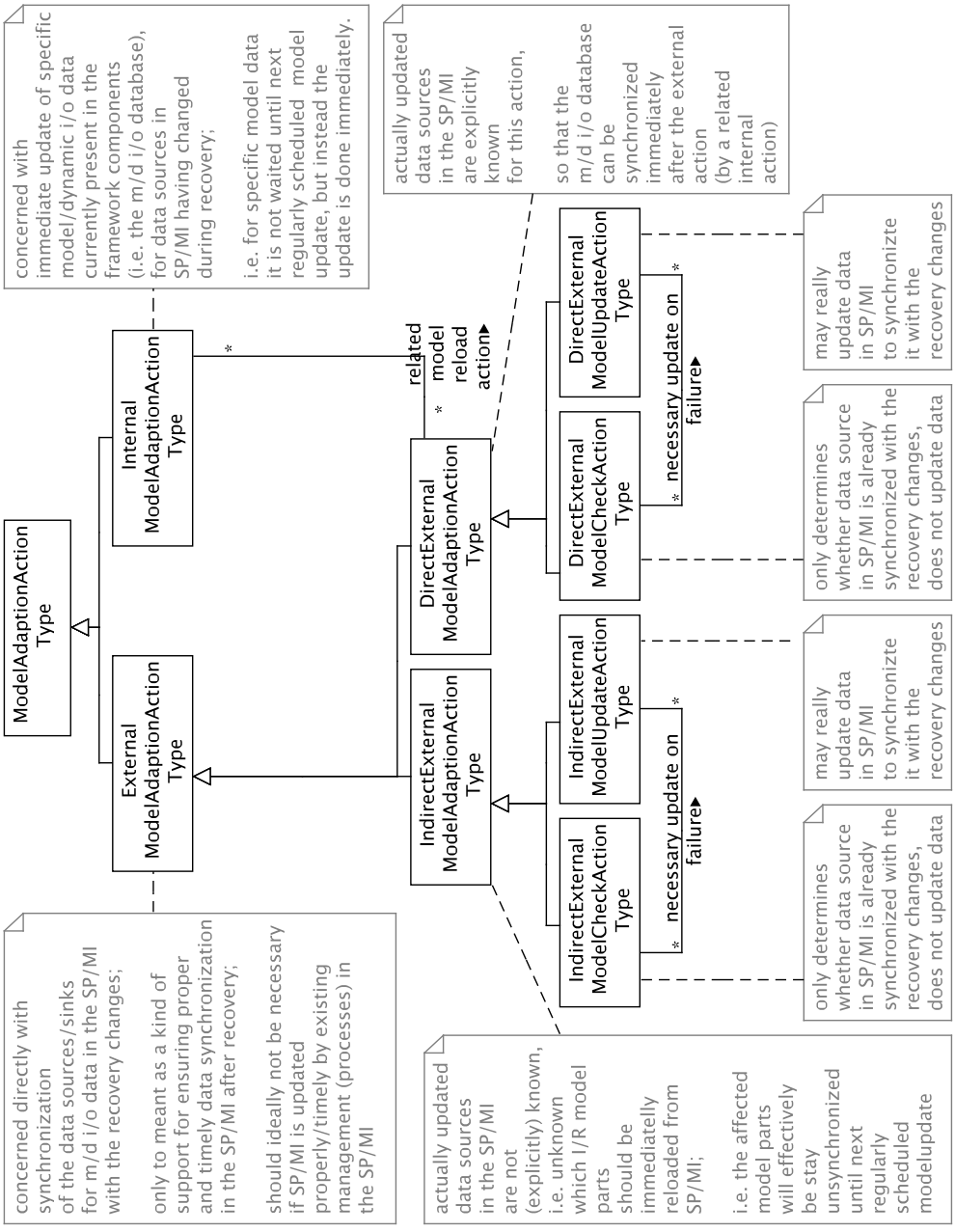

Figure 4.102: Classification of model adaption types

contents of the I/RA model/dynamic database (see Sect. 4.2.5.2 on p. 196), which as part of the I/R analyzer actually contains all currently loaded model or dynamic data from SP/MI.

external and internal model adaption actions

Consequently, this yields a basic classification of model adaption action (types): *external model adaption actions* concerned with the first task, and *internal model adaption actions* concerned with the second task. The designations *external* and *internal* are chosen with respect to the point-of-view of the I/R framework in contrast to the SP/MI. Concluding, external model adaption actions target directly at the consistency and synchronization of data sources/sinks in the SP/MI with occurred recovery changes, whereas inter-

nal model adaption actions are involved in the immediate reloading of corresponding model/dynamic data currently existing in the I/RA model/dynamic database to synchronize it with these updated SP/MI data sources/sinks.

check-only and true-update

External model adaption actions can be further sub-classified according to two different aspects: On the one hand, they can be classified as *check-only model adaption actions* and *update model adaption actions*. On the other hand, they can be classified as *explicit model adaption actions* and *implicit model adaption actions*. The former classification basically distinguishes whether an external model adaption action in any case only checks the consistency of SP/MI data sources/sinks with the occurred recovery changes (check-only action), or whether it also potentially performs updates if necessary (update action). A check-only model adaption action can reference a corresponding update model adaption action, which can be used if the check-only action detects inconsistency. The latter classification is differentiating the direct, explicit and the indirect, implicit modeling of the I/R model data parts affected by the external model adaption action: whether it is explicitly modeled which respective model/dynamic data parts in the I/RA model/dynamic database are actually involved by the updated and synchronized data sources/sinks in the SP/MI. I.e., for so-called *direct external model adaption actions* it is explicitly modeled which respective model/dynamic data parts can/should be reloaded immediately by a corresponding internal model adaption action. In contrast, for *indirect external model adaption actions* the affected I/R model data parts are not explictly modeled and, the actual synchronization with the I/RA model/dynamic database is not performed until the next regularly scheduled model update takes place by model/dynamic i/o scheduler (see Sect. 4.2.5.3). Consequently, direct external model adaption actions have references to corresponding internal model adaption actions, indirect external model adaption actions have not.

direct and indirect modeling of affected I/R model parts

Lastly, internal model adaption actions can also be further classified, depending on the kind of data they reload in the I/RA model/dynamic database: (static) model data, or additional dynamic data. This has already been introduced in Sect. 4.2.5.6 on p. 210.

To sum it up, the classification of model adaption action (types) and the relationships between the various kinds of model adaption action (types) determine some kind of direct dependencies among model adaption actions. This way, they can be regarded as a further kind of planning dependencies in the model adaption dependency model, in addition to the ones described previously, i.e., the ones relating (aggregated) recovery changes and model adaption actions. Nevertheless, such direct dependencies among model adaption actions can be also encoded by the first type of model adaption planning dependencies, by combining multiple model adaption action specifications (multiplicity > 1) with appropriate, joint dependency constraints on these model adaption action specifications, as a kind of partial plan/program pattern.

example

For example, for each of the tasks $ModAdp_{\text{invent\_db}}$, $ModAdp_{\text{afs\_ip\_conf}}$, $ModAdp_{\text{afs\_path\_conf}}$, $ModAdp_{\text{afs\_throughput}}$, for the synchronization with the aggregated recovery change *AFS device change*, the set of corresponding model adaption action instantiations per task, $ModAdpActSet1(task)$, can now be specified: For instance, $ModAdpActSet1(ModAdp_{invent\_db}) := \{ModAdpAct_{\text{invent\_db\_check}}, ModAdpAct_{\text{invent\_db\_update}}, ModAdpAct_{\text{invent\_db\_recovery\_deps\_reload}}\}$ with the following model adaption action (coordination) parameters (using the coordination parameter list declaration $CoordParamListDecl1$ proposed on p. 301):

- $ModAdpAct_{\text{invent\_db\_check}}$:
  - model_adaption_action_number: n_1
  - start_time_range: not constrained
  - start_depending_on: not constrained
- $ModAdpAct_{\text{invent\_db\_update}}$:
  - model_adaption_action_number: n_2
  - start_time_range: not constrained
  - start_depending_on: not(n_1)
- $ModAdpAct_{\text{inventdb\_recovery\_deps\_reload}}$:
  - model_adaption_action_number: n_3
  - start_time_range: not constrained
  - start_depending_on: n_2
- ...

The particular parameter bindings of *start_depending_on* ensure that action $ModAdpAct_{\text{invent\_db\_update}}$ is actually executed only if the execution of action $ModAdpAct_{\text{invent\_db\_check}}$ fails, and further that action $ModAdpAct_{\text{invent\_db\_recovery\_deps\_reload}}$ is executed not until the actions $ModAdpAct_{\text{invent\_db\_check}}$ and/or $ModAdpAct_{\text{invent\_db\_update}}$ are completed. Based on this, the corresponding model adaption dependency from *AFS device change* to $ModAdpActSet1(ModAdp_{invent\_db})$ is now defined in detail, and so also demonstrates how model adaption planning dependencies encode relationships between model adaption actions (as partial program patterns). Similarly, $ModAdpActSet1(task)$ and the corresponding model adaption dependency from recovery change *AFS device change* can be specified for the other tasks.

4.5.3 Implementation of recovery tracking dependency models

Having designed both types of RT dependency models in Sect. 4.5.1 and Sect. 4.5.2 guidelines for their actual realization, mainly with respect to

the basic component architecture, BCArch, and its basic realized workflow, BRWf, (see Sect. 4.2.5), are discussed in the following. Similarly as done for IA and RA previously in Sect. 4.3.11 and Sect. 4.4.3 respectively, particular implementation techniques for the realization of the respective architecture components and the related, used data structures are introduced. First, the whole design of both types of RT dependency models and related architecture components is summarized, and based on this guidelines for the realization by respective, concrete implementation techniques are given.

summary on recovery tracking dependency models and their usage

Both types of recovery tracking dependency models (RT dependency models) are vital to perform an appropriate recovery tracking at the end of each I/RA run, and therefore are vital for the usefulness of the whole I/RA framework. These models have been initially introduced in Sect. 4.2.4.2 on p. 186 as part of the basic refined abstract workflow (BRAWf). In the basic realized workflow (BRWf, Sect. 4.2.5.6) the basic realization of recovery tracking, and the respective usage of RT dependency models was discussed on p. 209 (BRWf.3.1) as well as on p. 210 (BRWf.3.2). There, the two steps of recovery tracking, i.e., recovery changes tracking (BRWf.3.1, recovery changes tracking subworkflow) and model adaption (BRWf.3.2, model adaption subworkflow) are performed by two particular *I/RA modules* (compare p. 199), namely the *recovery changes tracker* and the *model adaption component*. These two parts of the I/R analyzer (Sect. 4.2.5.5) utilize the respective type of RT dependency model for their tasks: The former component uses the recovery changes tracking dependency model, in combination with particular m/d i/o modules (see Sect. 4.2.5.2 on p. 194 in general) which are referenced in the dependency model, and which are used for the actual monitoring of individual recovery changes. The latter component uses the model adaption dependency model to construct a model adaption plan which can be executed in an (mostly) automated way by using appropriate m/d i/o modules for synchronization (model adaption) of the SP/MI and all depending I/R model with the occurred recovery changes.

In Sect. 4.5.1 and Sect. 4.5.2 a design of both RT dependency models including all related data structures, i.e., concerning the specification of recovery changes, model adaptions, model adaption plans, has been developed. This design of RT dependency models represents a high refinement of the basic introduction given in Sect. 4.2.4 for these models and their related data structures (compare Fig. 4.33 on p. 185). Moreover, it was taken care, that the design of RT dependency models is compatible with the design of the preceding IAFw (Sect. 4.3) and RAFw (Sect. 4.4), i.e., mainly with the design of recovery plans and their recovery actions.

towards a concrete implementation

Having summarized the design of recovery tracking dependency models and their related components of the basic component architecture, namely the recovery changes tracker and the model adaption component as particular I/RA modules, in the following the actual realization guidelines by respective, concrete implementation techniques are discussed. Here, different types of implementations may be considered, each based on different existing methods or

techniques, such as RBR (Sect. 3.6 on p. 103) or CBR (Sect. 3.6.3 on p. 108). First, the implementation of recovery changes is discussed, second the implementation of model adaption. Only one specific type of implementation, being directly compatible with the implementation of IAFw and RAFw, is introduced for each of them.

implementation of recovery changes tracking

Recovery changes tracking has first to determine the types of individual recovery changes to be monitored from the selected recovery plan's execution information by using the respective type of recovery change dependencies. This is a task which can be, at least as one option, be implemented in a similar way as impact analysis (compare p. 289 in Sect. 4.3.11), i.e., by RBR using an deductive logic (p. 105 in Sect. 3.6). The second task of recovery changes tracking is to actually monitor the determined types of recovery changes by respective m/d i/o modules (p. 194) during the actual recovery. So, this has to be mainly done by these modules, which eventually report back all monitored recovery changes. The third task of recovery changes tracking is to pre-aggregate the monitored individual recovery changes to provide an consistent input for model adaption. This may, as one particular option, also be implemented by an RBR approach, similarly as the first task of impact rating, i.e., direct business impact rating, was implemented in Sect. 4.4.3 on p. 313, as this task is mainly concerned with function conversion. Concluding, the implementation of the recovery changes tracking dependency model and the related recovery changes tracker can be done using RBR. Nevertheless, in addition to the implementation of recovery changes tracking dependency model and recovery changes tracker, the whole task of recovery changes tracking is also largely depending on the respective m/d i/o modules used for collecting the individual recovery changes from the SP/MI.

implementation of model adaption

Model adaption has two parts, the construction of the model adaption plan, and the execution of the plan by using respective m/d i/o modules. As the design of the model adaption dependency model and related data structures was largely adopted from the respective notions used for recovery planning (e.g., model adaption actions/plans correspond to recovery actions/plans) the implementation for model adaption planning can also be basically adopted from recovery planning. That is, similar techniques and methods can be used as are used for realizing recovery planning (see p. 313 in Sect. 4.4.3): RBR, or more specifically reasoning by Transaction Logic (Sect. 3.6.2), or alternatively CBR (Sect. 3.6.3)). For the actual automated execution of the model adaption plan a (simple) workflow engine is necessary which coordinates the actual model adaptions performed by the respective m/d i/o modules. Also, similar as recovery changes tracking, in addition to the implementation of model adaption dependency model and model adaption component, model adaption is largely depending on these respective m/d i/o modules.

4.6 Summary

In this chapter, I/RAFw, a generic, comprehensive framework for impact and recovery analysis (I/R analysis) was developed which fits to all requirements identified in Sect. 2. Fig. 4.103 as a reminder of Fig. 4.1 gives an overview of this developed framework. First, as a general basis, the basic framework (BFw) was introduced in Sect. 4.2, comprising a highly refined basic workflow for the whole I/R analysis, the basic realized workflow (BRWf), and a corresponding basic component architecture (BCArch) for actually executing this workflow. Based on the basic framework, consecutively three extension frameworks, namely the impact analysis framework (IAFw), the recovery analysis framework (RAFw), and the recovery tracking framework (RTFw), were treated. Each of them covers a detailed design and corresponding implementation guidelines for all necessary data models and further, related BRWf workflow artifacts, concerning one of the three particular tasks of I/R analysis, i.e., impact analysis, recovery analysis, and recovery tracking, identified in Sect. 2.

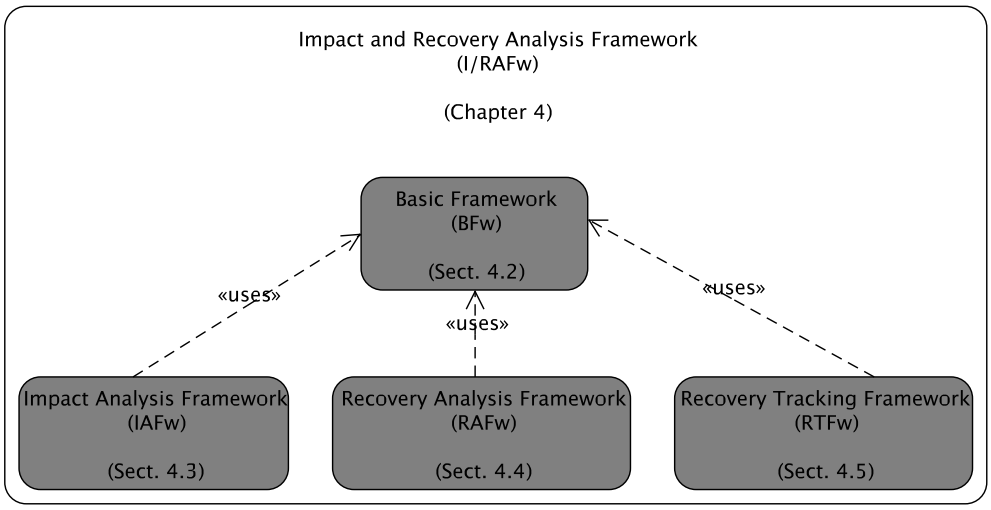

Figure 4.103: Overview of the developed I/RA framework (I/RAFw) consisting of basic framework and its three extension frameworks (compare Fig. 4.1)

In the following chapter, an instantiation methodology is discussed which enables a service provider to instantiate the generic framework developed in this chapter to his concrete real-world service scenario.

Chapter 5

Instantiation Methodology for Concrete Scenarios

Contents

After the development of the generic framework for impact analysis and recovery (*I/RAFw*) in Chapter 4, *I/RAFw InstMeth*, an instantiation methodology for concrete real-world service scenarios is presented in this chapter. This instantiation methodology enables a service provider to actually apply the generic framework to the concrete real-world services provided by him.

In Sect. 5.1, first, parts or tasks of instantiation of I/RAFw in general are presented, second, a particular top-down oriented instantiation methodology realizing these tasks is introduced. Based on this, Sect. 5.2, and especially Sect. 5.3 discuss the details of this top-down oriented instantiation methodology. Also included are example applications for the example services of Sect. 2.3.

5.1 Instantiation of the Impact and Recovery Analysis Framework in General

The instantiation of the generic impact and recovery analysis framework (*I/RAFw*, Sect. 4) to a given real-world service scenario, tersely called *I/RAFw instantiation (I/RAFw Inst)* in the following, in general requires that all particular parts of the framework are appropriately instantiated and adapted to the given service scenario: This particularly comprises the instantiation and adaption of all introduced workflows steps including their input/output artifacts, especially all particular data models, as well as the basic component architecture. Fig. 5.1 generally illustrates the subdivision of the I/RAFw instantiation into general tasks discussed in the following. Basically I/RAFw instantiation is subdivided into two main tasks, *I/RA component architecture instantiation* and *I/RA model instantiation*.

tasks of I/RAFw instantiation

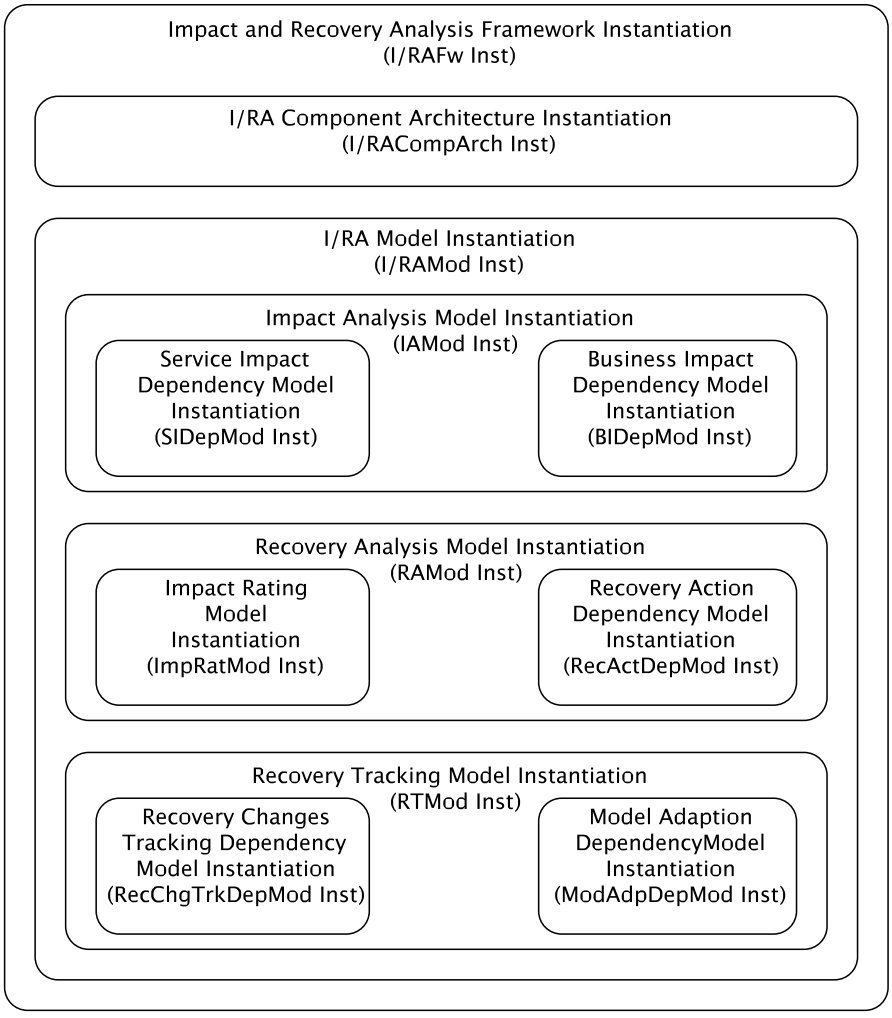

Figure 5.1: General tasks of instantiation for I/RA framework (I/RAFw Inst)

architecture instantiation

The former one, the architecture instantiation, is relatively generic in nature, but serves as basis for the latter one, the following model instantiation. Architecture instantiation comprises the instantiation of the whole basic component architecture (see Sect. 4.2.5.2), i.e., the components of m/d i/o data area and the proper I/RA area (compare Fig. 4.38 on p. 195). So, this includes generic support for all necessary m/d i/o data modules (compare Sect. 4.2.5.4), as well as generic support for m/d data to be used by the particular I/RA workflow modules (compare Sect. 4.2.5.5 on p. 199).

model instantiation

The latter one, the model instantiation, is further subdivided according to the different particular types of models used for I/R analysis, which correspond to steps of the basic refined abstract workflow, BRAWf (compare Fig. 4.35 on p. 189), as well as to the subworkflows of the basic realized reasoning workflow, BRWf.reason (see Fig. 4.42 on p. 202). For each type of I/R model, i.e., impact dependency model, impact rating model, recovery action dependency model, recovery changes tracking dependency model, and model adaption dependency model, the particular contents of each model are identified and specified according to the concrete service scenario given. E.g., in the case of the service/business impact dependency model, degradation dependencies for the concrete services of the given scenario are identified and specified. This includes, for each type of I/R model, an appropriate further subdivision into static model data and additional dynamic data which is only referenced by the static model data (compare p. 143 in Sect. 4.2.2.2, p. 175 in Sect. 4.2.3.2, and p. 186 in Sect. 4.2.4.2). Last, corresponding m/d i/o modules necessary for the exchange of this m/d i/o data with the SP/MI are designed and specified.

top-down instantiation methodology

Based on the subdivision of the framework instantiation into the separate tasks in general, now a particular, top-down-oriented methodology realizing these tasks is introduced. Fig. 5.2 illustrates this workflow of top-down I/RA framework instantiation methodology (top-down I/RAFw InstMeth).

orientation according to policies defining and prioritizing business degradations

Basically, *top-down orientation* of I/RA framework instantiation means the *orientation according to the relevant types of business degradations*, and more general the *orientation according to the business policies of the service provider* which define and prioritize these business degradations. As business degradations cover especially the canonical example of SLA violation costs (compare p. 224 in Sect. 4.3.3), i.e., I/R analysis being also concerned with the avoidance of SLA violations, the *top-down orientation* of I/RA framework instantiation can also be regarded as *customer-orientation*, depending on the particular business policies of the provider concerning the definition of business degradations.

identification and specification of degradation (dependencies) based on business policies

More specifically, top-down orientation means to identify and specify degradations and their dependencies in a top-down manner (from business degradations via service degradations to resource degradations), based on business policies defining/prioritizing the relevant business degradations: That is, the top-down instantiation methodology for a particularly given service scenario starts from high-level business policies concerning definition and prioritizing of business degradations (e.g., including customer-orientation, i.e., avoid-

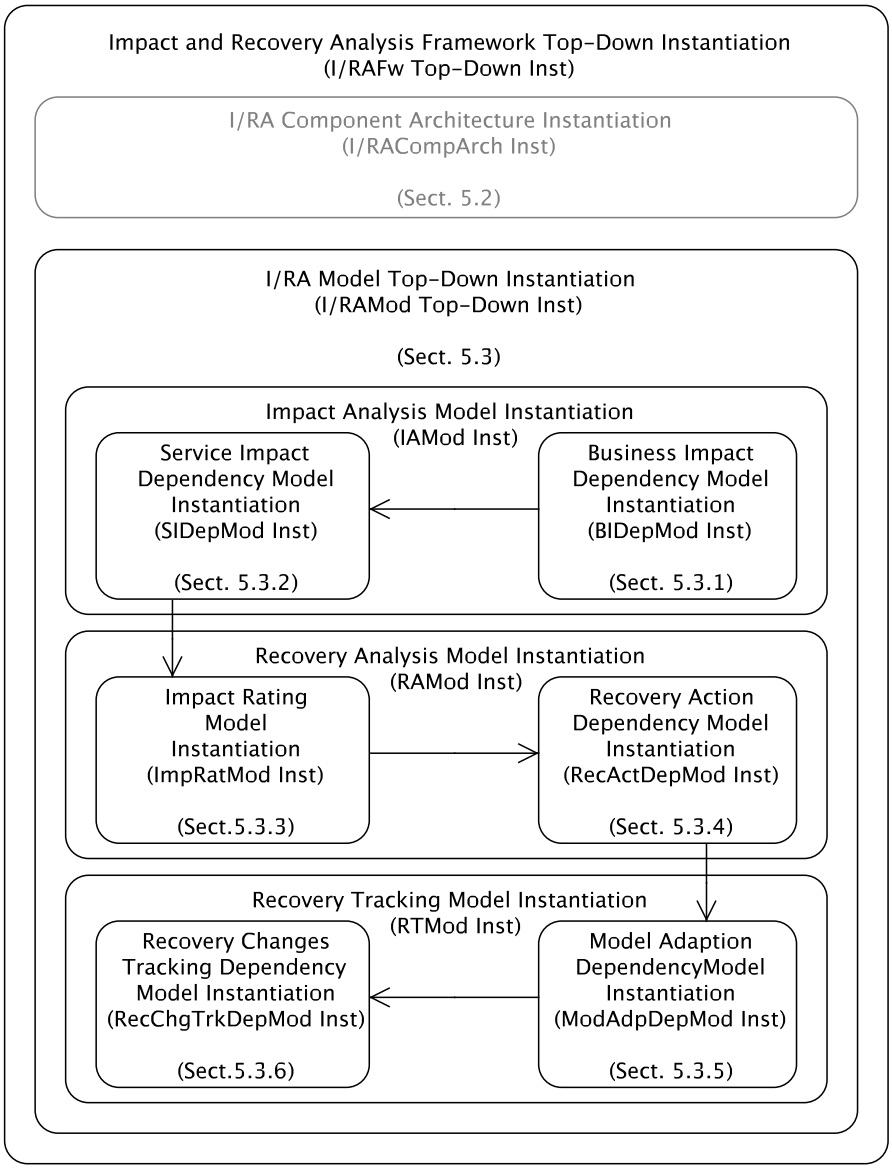

Figure 5.2: Overview of the top-down instantiation methodology for I/R analysis framework (compare Fig. 5.1)

ance of particular SLA violation costs). These policies are used to identify and specify the relevant types of business degradations and their dependencies on top-level service degradations within a business impact dependency model with proper granularity and accuracy. Based on this, relevant degradations of services and resources, as well as their degradation dependencies are identified and specified within a service impact dependency model with proper granularity and accuracy. That is, the identification and specification of degradation and their dependencies during framework instantiation follows the reverse direction as they will be used later-on during impact analysis, i.e., from business degradations to resource degradations instead of (initial) resource degradations to (entailed) business degradations.

direct business impact rating based on corresponding business policies

Following, the business impact rating model is defined according to the initially given business policies for prioritizing all relevant types of business degradations. Based on this, appropriately derived, indirect rating weighting specifications (see p. 295 in Sect. 4.4.1) are added to the business and service impact dependency model. Lastly, the recovery action dependency model, and both recovery tracking dependency models are defined.

parts and steps of top-down instantiation methodology

The particular top-down instantiation methodology (Fig. 5.2) is similarly subdivided like the framework instantiation in general (Fig. 5.1), namely in architecture instantiation and model instantiation, the latter one being subdivided according to the various types of I/R models: The top-down instantiation methodology is based on the relatively generic architecture instantiation, which is generically discussed in greater detail in Sect. 5.2. Thus, the proper description of the top-down instantiation methodology is mainly concerned with its particular model instantiation, whose steps, corresponding to the different types of I/R models, are organized as a methodology workflow with subsequent steps. These steps are outlined in the following, and are treated subsequently in detail in Sect. 5.3:

In correspondence with the introduction of top-down instantiation methodology above, these model instantiation workflow steps - in the order of their execution within the workflow - are: business impact dependency model instantiation, service impact dependency model instantiation, impact rating model instantiation, recovery action dependency model instantiation, model adaption dependency model instantiation, recovery changes tracking dependency model instantiation. Especially the instantiation of business impact dependency model precedes the one of the service impact dependency model, as motivated above. Moreover, also because of top-down orientation, model adaption dependency model instantiation precedes recovery changes tracking dependency model instantiation.

top-down BI dependency model instantiation

First, during business impact dependency model instantiation, from business policies concerning the definition/prioritization of business degradations (including avoidance of SLA violations) the relevant types of business degradations as well as their dependency on their entailing top-level service degradations are identified and specified in appropriate granularity and accuracy (compare above). The business policies used as starting point may include, as the canonical example, definitions of SLA constraints and related SLA penalty costs for any relevant service.

top-down SI dependency model instantiation

Second, during service impact dependency model instantiation, by using a given modeling of the services (e.g., MNM service model applied with UML diagrams for functionality specification as described on p. 87 in Sect. 3.3.1) whose top-level service degradations have been identified and specified before, any relevant dependencies on subservices and resources are recursively identified and specified. The used modeling of the service serves mainly for the identification of dependencies, whereas for their detailed, particular specification of these dependencies in appropriate granularity/accuracy existing detailed best practices, and expert knowledge combined with particu-

lar approaches for determining and specification of dependencies for particular technologies and fields involved are re-utilized. Additionally to the specification of the entailing degradations and their dependencies, also requirements to be posed on the actual technical realization for gathering/measuring all necessary static and dynamic service/resource dependency data from the SP/MI by m/d i/o data modules are determined.

top-down impact rating model instantiation

Third, as already introduced above, derived from the business policies concerning prioritization of business degradations, appropriate definitions for business impact rating model are performed, and based on this recursively indirect degradation rating weighting definitions are added to the impact dependency models (compare p. 295). The former of these actions could also directly follow the top-down SI dependency model instantiation, but latter has to be done after or at least in parallel to the top-down SI dependency model instantiation.

top-down recovery action dependency model instantiation

Afterwards, any potential recovery handling, correspondingly particular recovery actions, and their particular influence/effect/execution information (compare p. 173 in Sect. 4.2.3.2) are identified and specified. This is done by using the given modeling of the relevant services, in combination with existing best practices, and expert knowledge for all the above identified, relevant potential resource degradations, in appropriate granularity and accuracy, with respect to their particular resource technologies involved.

top-down model recovery tracking dependency model instantiation

Next, from the execution information of all potential recovery actions identified before, all potentially necessary model adaption actions, and their dependencies, together with the necessary m/d i/o data modules to perform them are identified, specified.

Lastly, for all potential model adaption actions identified before, appropriate recovery changes to allow to recognize which model adaption actions have to be performed during a concrete I/RA run later-on, and their dependencies are identified/specified together with a specification of all necessary m/d i/o modules to monitor such recovery changes.

The detailed treatment of each step in Sect. 5.3 will include references to particular related work, as well as example application of the respective model instantiation methodology step for the example services of Sect. 2.3.

other instantiation methodologies

It is to mention that the top-down I/RA framework instantiation methodology is only one particular possibility for such an instantiation methodology. Other methodologies can be considered also realizing the general tasks of Fig. 5.1. Such other methodologies may have especially another focus or orientation instead of being top-down oriented.

Two possibilities for a bottom-up oriented I/RA framework instantiation methodology outlined as examples in the following: the first one based on given, known types of resource degradations (resource degradation oriented, i.e., oriented according to resource degradations especially for their use for impact analysis), and the second one even further based on given types of resource degradations and their given types of recovery actions (resource re-

covery oriented, i.e., oriented according to resources and their potential recovery).

bottom-up instantiation methodology starting from known types of resource degradations

The first one is starting from given resource degradation types, including the granularity/accuracy of their specification, to derive all potential types of entailed business degradations which can be derived from the given resource degradation types. This methodology may be either used to check whether impact analysis for an existing set of business degradation policies (or related SLA constraints/penalty definitions) can be supported with appropriate granularity/accuracy by use of the resource degradations given, or more general to actually identify and propose a set of business policies for business degradations (or related SLA constraint/penalty definitions) for which impact analysis can be supported by use of the resource degradations given. That is, this bottom-up instantiation methodology starts from given resource degradations and check existing or identifies/proposes new business policies concerning business degradations (or related SLA definitions) suitable for impact analysis.

bottom-up instantiation methodologies starting from known types of (resource) recovery actions

The second bottom-up instantiation methodology is going a step further, it is starting from given resource degradations, and their potential (resource) recovery actions to derive the potential types of business degradations, or related business policies defining them (or related SLA definitions), for which a (business) recovery can be realized appropriately with the given set of potential (resource) recovery actions. Similarly, as the first bottom-up instantiation methodology, this one may be actually used either to check whether a set of existing business policies regarding business degradation definition (or related SLA definitions) can be appropriately supported concerning recovery, or in general identify and propose new appropriate business policies (or SLA definitions) which can be supported concerning recovery.

Any of the possible instantiation methodologies, including the top-down oriented one, are mainly concerned with the model instantiation, while the component architecture instantiation is more generically common to all of them. In the following, Sect. 5.2 discusses this component architecture instantiation in general, while Sect. 5.3 discusses in detail the different, subsequent steps of the top-down model instantiation.

5.2 Component Architecture Instantiation

Here, the component architecture instantiation, introduced above, is outlined in more detail. The component architecture instantiation is relatively generic in nature, but it serves as a basis for the particular model instantiation of the respective instantiation methodology, e.g., the top-down model instantiation methodology discussed in detail in Sect. 5.3. It comprises the instantiation of the various architecture components of the basic component architecture, BCArch (compare Fig. 4.37 on p. 194 and Fig. 4.38 on p. 195).

On the one hand, there is the instantiation of the m/d i/o data engine (see Sect. 4.2.5.3) for controlling the m/d i/o data access area (Sect. 4.2.5.4), as well as the instantiation of the latter. The instantiation of the m/d i/o data access area comprises the instantiation of any m/d i/o modules, identified in detail in the model instantiation.

On the other hand, there is the instantiation of the I/R analyzer (Sect. 4.2.5.5) comprising an I/RA model/dynamic database (see p. 196 in Sect. 4.2.5.2), and the I/R analyzer reasoning engine (p. 196 in Sect. 4.2.5.2) accessing all I/R model data via the I/RA model/dynamic database. In more detail, the I/R analyzer reasoning engine is comprising a generic I/R analyzer workflow engine to coordinate its various specific I/R analyzer workflow modules (tersely I/RA modules; compare p. 199 in Sect. 4.2.5.5, especially Fig. 4.40 on p. 200, and Fig. 4.41 on p. 201).

That is, component architecture instantiation can be subdivided into two tasks: First, the relatively generic instantiation of the m/d i/o data engine and the I/RA reasoning engine, and second, the generic support for all types of I/R model as part of the I/RA model/dynamic database, and the corresponding generic support of the respective m/d i/o data modules as part of the m/d i/o data access area.

The former task includes the instantiation of the particular I/RA (workflow) modules as part of the I/RA reasoning engine. Basically, the generic instantiation of these I/RA workflow modules has been already approached subsequently in Sect. 4.3.11, Sect. 4.4.3, and Sect. 4.5.3, where implementation guidelines have been introduced for each of the respective I/RA workflow modules concerned with impact analysis, recovery analysis, and recovery tracking respectively.

The latter task is specifically concerned with providing a common basis for the various steps of the particular model instantiation: In general, each model instantiation step is instantiating one of the I/R models. In more detail, each model instantiation step is comprising the specification of the contents of the respective I/R model according to the given service scenario, as well as the specification of all m/d i/o data modules necessary for exchange of the model/dynamic data with the SP/MI. As a further refinement of these tasks, any model instantiation step can be generically subdivided into the three following subtasks (being visualized in Fig. 5.3): identification/specification of model

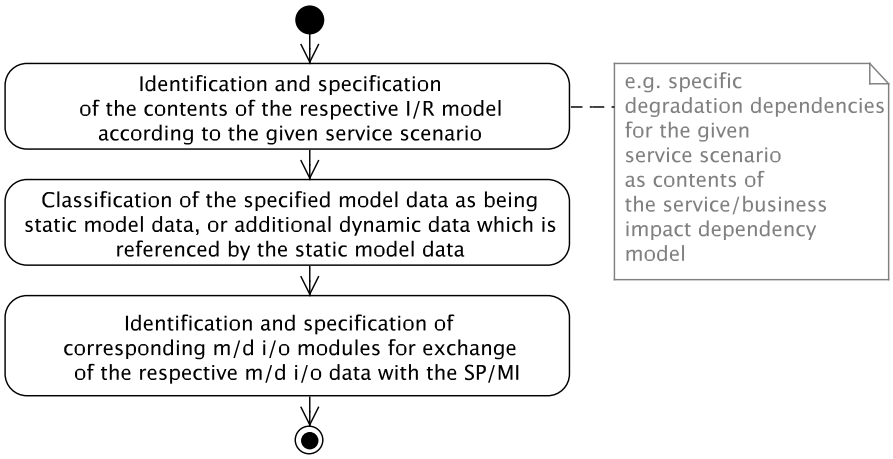

Figure 5.3: Generic substeps or subtasks of model instantiation for a particular I/R model (for one of the model instantiation parts/steps in Fig. 5.1/Fig. 5.2 respectively)

data, classification of this data as static or dynamic, identification/specification of respective m/d i/o data modules.

First, the identification and specification of model data for the particular I/R model (as contained in I/RA m/d database during actual I/RA runs later-on) takes place. Second, the specified model data is classified as being proper, static data or additional, dynamic data (only data source referenced in static data) of the respective I/R model (compare p. 143 in Sect. 4.2.2.2, p. 175 in Sect. 4.2.3.2, and p. 186 in Sect. 4.2.4.2). Third, the identification and specification of particular m/d i/o modules for realizing the exchange of the specified model/dynamic i/o data with the SP/MI is performed. In the following section the particular steps of top-down model instantiation methodology are treated subsequently in detail, also regarding these three generic subtasks.

5.3 Top-Down Model Instantiation

Following, the different steps of top-down I/RA model instantiation, having been introduced in Sect. 5.1 (compare p. 335 and Fig. 5.2), are subsequently discussed in detail.

5.3.1 Business Impact Dependency Model Instantiation

For business impact dependency model instantiation in top-down manner, from given business policies (or related SLA definitions), which determine and prioritize types of (top-level) business degradations (including accuracy/-granularity), the relevant top-level business degradations, and their dependencies on entailing (top-level) service degradations are derived, potentially via inter-mediate entailing business degradations (compare Fig. 4.13 on p. 148).

example instantiation

In the case of the example scenario of Sect. 2.3, e.g., it is started with the general business policy that given SLAs have to be meet, especially the related SLA constraint definitions and SLA penalty definitions of the example mail service (Table 2.6 on p. 40). Based on these related SLA definitions, business degradations and dependencies on top-level service degradations are derived. For example, from SLA constraint sla_cnstr_{mail3} and the corresponding SLA penalty definition sla_pnlty_{mail3}, the business degradation $g^{\star}_{b1-1}$ can be derived: $g^{\star}_{b1-1}$ as a general business degradation represents any SLA penalty costs arising from the violation of this specific SLA constraint sla_cnstr_{mail3}, i.e., the average mail sending delay rising above the limit of $5\ \text{min}$ (compare Table 2.6). Therefore, $g^{\star}_{b1-1}$ actually is a generalization of the particular degradation g_{b1-1} of example situation $ExSit1$ (compare Fig. 4.6 on p. 131 as well as Fig. 4.12 on p. 146).

When utilizing the business impact dependency model BIDepMod(Char) of Sect. 4.3.3 for the specification of $g^{\star}_{b1-1}$, the degradation details are basically determined by a business degradation characteristic function, or more specifically in this case by a service level penalty function $slp^{\star}_{\text{mail3}}(t)$, which is a generalization of the service level penalty function of the particular business degradation g_{b1-1}. In this specific case here, this function of time $slp^{\star}_{\text{mail3}}(t)$ may have the same definition as $slp_{\text{mail3}}(t)$ (compare p. 51 in Sect. 2.3.4), as the related SLA constraint sla_cnstr_{mail3} is not taking into account the actual exceed above the limit of $5\ \text{min}$, but in general it may be a function parameterized by the actual exceed in value above the limit.

Based on the identified potential top-level business degradation $g^{\star}_{b1-1}$, its entailing service degradation(s) can be identified and specified, also from the given business policies in general, or the related SLA definitions: the general service degradation $g^{\star}_{s1-1}$, actually representing a generalization of the particular service degradation g_{s1-1} of example situation $ExSit1$. $g^{\star}_{s1-1}$ as a general

service degradation represents any particular service degradation which will result in a violation of SLA constraint sla_cnstr_{mail3}, i.e., any degradation of the mail sending delay rising somehow (and for some duration or even permanently) above the limit of $5\ \text{min}$ as agreed in the SLA. Depending on the needs of the service provider, i.e., based on the given business policies, particular adapted (soft) limits (with some threshold) may be used instead of the hard limits as agreed in the SLA in order to also recognize and handle proactively future impending degradations. That is, for $g^{\star}_{s1-1}$ a more lower soft limit of $3 + \delta\ \text{min}$ may be used instead with δ (e.g., $= 0.5\ \text{min}$) representing a threshold, as $3\ \text{min}$ is the normal value of mail average sending delay in this scenario. Alternatively, this proactive consideration and handling of impending degradations in the near future may be covered by a additional type of business degradation further to SLA violation costs.

Concluding, for the business degradation types derived from SLA definitions (related to the given business policies), it can be seen, that the business degradations can be derived from the SLA penalty definitions (e.g., $g^{\star}_{b1-1}$ from sla_pnlty_{mail3}), and the corresponding, entailing (top-level) service degradations can be derived from the respective SLA constraint definition(s), namely as any violation thereof (e.g., $g^{\star}_{s1-1}$ from sla_cnstr_{mail3}). For other types of business degradations (compare p. 224 in Sect. 4.3.3), the particular mapping from the given business policies which determine and prioritize these business degradations, is depending on the particular type of business policy as defined by the service provider. That is, no concrete hints can be made for this general case, and the service provider has to provide some kind of such mapping along with the policies themselves, according to his needs.

steps of business impact dependency model instantiation

After having given a more detailed example how the identification and specification of degradations and their dependencies for a business impact dependency model in top-down manner can be done, the business impact dependency model instantiation as a whole is discussed more generically, especially concerning the three generic subtasks of model instantiation introduced in Sect. 5.2:

First, as introduced above, the relevant types of top-level business degradations are identified and specified according to given business policies, in appropriate granularity/accuracy, e.g., general potential business degradation $g^{\star}_{b1-1}$. This includes the specification of the corresponding business degradation characteristic (see p. 225), e.g., the business degradation characteristic $slp^{\star}_{mail3}(t)$ of $g^{\star}_{b1-1}$.

Second, any potential inter-mediate business degradation entailing the top-level ones and the involved degradation dependencies are identified and specified along with the specification of the corresponding business degradation characteristics. This step provides a way to subdivide the complete set of cases covered by a top-level business degradation into different subcases. In the case of SLA violation costs (e.g., $g^{\star}_{b1-1}$) this can e.g., be used to allow a differentiation of SLA violation costs for particular customers. Lastly, this potential subdivision, i.e., whether to actually make use of it, and to what de-

gree, is determined by the given needs of the service provider as defined by the given business policies. The identification and specification of inter-mediate business degradation and corresponding degradation dependencies (business-to-business degradation dependencies) actually may impose requirements on the later implementation, concerning any necessary data source in the SP/MI used for determining concrete degradation dependencies during I/RA runs later on. For example, the access to particular appropriate SLA databases defining such business-to-business degradation dependencies (e.g., in terms of SLA violations of a particular customer) may be necessary to allow to determine any concrete instantiation of these degradation dependencies later on.

Third, also as introduced above, the identification and specification of top-level service degradation entailing the types of business degradation determined before (top-level or intermediate ones) and corresponding degradation dependencies (service-to-business degradation dependencies) in appropriate granularity/accuracy is performed. For example, the service degradation $g^{\star}_{s1-1}$ entailing the business degradation $g^{\star}_{b1-1}$ is determined along with all degradation specification details (compare Fig. 4.7 on p. 135), e.g., the degradation value specification $> 5\ \mathrm{min}$. Here, similarly requirements are imposed on the implementation concerning data sources in SP/MI for collecting all necessary information about concrete instantiations of these general degradation dependencies during I/RA runs later on. This may include again access to necessary SLA databases, containing the current version of the particular SLAs with any potentially affected customer. Moreover, for business degradation types other than SLA violation costs, e.g., when the entailed business degradation is concerned with current/future service usage (e.g., to determine/estimate current/future revenue loss entailed from potential service degradations) access to measurement of current service usage and estimation of future service usage from historic/current measurements may be necessary (e.g., with extrapolation techniques having been treated in Sect. 3.5.2).

Fourth, based on the identification and specification of all business and top-level service degradations and their degradation dependencies, the details of these specifications are classified as either static model data or additional, dynamic data (compare p. 143 in Sect. 4.2.2.2), which again may impose further requirements for the actual later implementation.

Last, appropriate m/d i/o data modules for exchanging all the specified model or dynamic data to derive concrete degradation dependencies in later I/RA runs are identified and specified. This is specifically related to the above mentioned requirements having been imposed by the specification of general degradations and their dependencies. That is, the actual implementation of these m/d i/o modules has to fulfill all these requirements.

5.3.2 Service Impact Dependency Model Instantiation

For service impact dependency model instantiation in top-down manner, from top-level service degradations entailing resource degradations are identified and specified, potentially via inter-mediate entailing service degradations. (compare Fig. 4.13 on p. 148). For the identification a modeling of the involved services is used (e.g., by the MNM service model and its extension using UML diagrams for functionality specification, compare p. 87 in Sect. 3.3.1). For the following specification existing best practices, expert knowledge, combined with existing approaches for measuring/specifying all aspects of degradations and their dependencies with respect to the concrete technologies involved, are utilized.

example instantiation

In the case of the example scenario of Sect. 2.3, e.g., from the top-level service degradation $g^{\star}_{s1-1}$, identified in the previous section, respective entailing resource degradations $g^{\star}_{r1}$ and $g^{\star}_{r1b}$ are derived: The latter two resource degradations are generalizations of the two particular resource degradations g_{r1} and g_{r1b} from the example situation $ExSit1$ (Fig. 4.6 on p. 131 as well as Fig. 4.12 on p. 146) respectively

generalized degradation details

The generalized degradation details (Fig. 4.7 on p. 135) of $g^{\star}_{s1-1}$ (comprising the ones of g_{s1-1}) are as follows: The degradation subject is the same in this case, $f_{\text{mail/use/send}}$, also the degradation manner, *high mail sending delay*. Degradation time/duration is not further particularly constrained. The degradation value accuracy of $g^{\star}_{s1-1}$ is generally specified as *mail sending delay* $> 5\ \text{min}$ (comprising also the particular case of *avg. mail sending delay* $= 10\ \text{min}$ for g_{s1-1}), as $5\ \text{min}$ is the value defined as limit in the related SLA constraint sla_cnstr_{mail3} for the entailed general business degradation $g^{\star}_{b1-1}$. Alternatively, a more restricted value range, e.g., $> 4\ \text{min}$, may be used in order to take into account an additional threshold value, e.g., of $1\ \text{min}$. Similarly degradation details of the generalized entailing resource degradations $g^{\star}_{r1}$ and $g^{\star}_{r1b}$ are specified: e.g., the value accuracy of $g^{\star}_{r1}$, *link utilization* $> 2\%$, as 15% represents the normal, average value (in 5-minute average).

Based on these generalized degradation details, a detailed generalized degradation dependency using SIDepMod(QoXInst) of Sect. 4.3.8 is specified (compare particular example on p. 271 in Sect. 4.3.8) is given in Table 5.1.

During the derivation from $g^{\star}_{s1-1}$ to $g^{\star}_{1}$ and $g^{\star}_{1b}$, inter-mediate, general degradations are involved, namely the service degradation $g^{\star}_{s1-0}$ and $g^{\star}_{s1b-0}$, being generalization of g_{s1-0} and g_{s1b-0} respectively (compare refinement in Fig. 4.12 on p. 146). Similarly as for the generalized resource degradations above the degradation details of $g^{\star}_{s1-0}$ and $g^{\star}_{s1b-0}$ and their generalized detailed degradation dependencies on $g^{\star}_{r1}$ and $g^{\star}_{r1b}$ respectively, and to $g^{\star}_{s1-1}$ are specified.

The approach to actually perform such a complete identification and specification of all degradations and corresponding dependencies of the service im-

specification of $g^\star_{r1}, g^\star_{r1b} \rightarrow g^\star_{s1-1}$ in various steps:

combination of both rises of the mail sending delay:
$g^\star_{s1-1/1}, g^\star_{s1-1/2} \rightarrow g^\star_{s1-1}$,
with $(i = 1, 2)$:
$g^\star_{s1-1/i} \equiv g_{high\_mail\_sending\_delay/i} := degradation($
subject: $f_{\mathrm{mail/use/send}}$,
manner: $gt_{highmailsendingdelay}$,
avg. mail sending delay rise: $value_i\ [min]$,
) with constraint $value_1 + value_2 > 5$ min,

high link utilization entailing the first rise:
$g^\star_{r1} \rightarrow g^\star_{s1-1/1}$, with
$g^\star_{r1} \equiv g_{high\_iplink\_util} := degradation($
subject: $r_{\mathrm{iplink}}(r_{\mathrm{rt\_lrz}}, r_{\mathrm{sw}_2})$,
manner: $gt_{highlinkutilization}$,
avg. link util: $u_{link} > 15\%$,
intermediary context: avg. ratio mail traffic/other traffic: rat_{traf},
intermediary context: avg. mail requests/min: $\#req_{mail}$,
) with constraint $value_1 = link\_util\_to\_mail\_delay(u_{link} \cdot rat_{traf}, \#req_{mail})$,
where $link\_util\_to\_mail\_delay(.,.)$ is an appropriate function describing the estimated relationship between link utilization, number of mail sending requests, and mail sending delay, (e.g., by use of respective historic value comparisons for all three parameters or some heuristics given by experts),

and high DNS response delay entailing the second rise:
$g^\star_{r1b} \rightarrow g^\star_{s1-1/2}$, with
$g^\star_{r1b} \equiv g_{high\_dns\_delay} := degradation($
subject: $r_{\mathrm{dns\_sv1}}$,
manner: $gt_{highdnsdelay}$,
avg. request delay: $d_{dns} > 10$ second,
additional context: min. number of DNS requests / mail sending: 2,
) with constraint $value_2 = 2 \cdot d_{dns}$

Table 5.1: Example of generalized degradation dependency (generalization of example specified in Table 4.18 and Table 4.19)

approach for derivation of entailing service/resource degradations

pact dependency model is based on the prerequisite of an existing modeling of the involved services. That it is assumed that a modeling for all services (and their relevant subservices), for which top-level service degradations have been identified during top-down business impact dependency model instantiation, is already in place, or alternatively can be appropriately instantiated subsequently in parallel with top-down service impact dependency model instantiation, e.g., by use of an appropriate service modeling instantiation methodology.

Here, as one example for such a modeling the MNM service model and its extension for modeling the whole service-view and resource-view by UML use

case diagrams, UML activity diagrams, and UML collaboration diagrams (see Sect. 3.3.1) is assumed and used as example. In general, other service modeling may be used alternatively.

Below, an example instantiation for the mail sending functionality of the example mail service will be performed using a modeling with the MNM service model. In general, the MNM service model has an instantiation methodology attached to it, especially a top-down instantiation methodology (see Sect. 3.3.1), which may be used to create the actual modeling of the services of the given service scenario in parallel with its use by top-down service impact dependency model instantiation.

steps of service impact dependency model instantiation

In the following the particular steps of service impact dependency model instantiation, especially in with respect to three generic subtasks introduced in Sect. 5.2, are discussed:

It is started with the top-level service degradations having been specified in appropriate granularity/accuracy during business impact dependency model instantiation (Sect. 5.3.1), e.g., the service degradation $g^{\star}_{s1-1}$ of the mail sending functionality $f_{\mathrm{mail/use/send}}$ which comprises all violations of the SLA constraint $sla\_pnlty_{\mathrm{mail3}}$.

First, the identification and specification in appropriate granularity/accuracy of any entailing service/resource degradation and corresponding degradation dependencies (service-to-service degradation dependencies, resource-to-service degradation dependencies, resource-to-resource degradation dependencies in Fig. 4.12 on p. 146) takes place, e.g., from $g^{\star}_{s1-1}$ to $g^{\star}_{r1}$ and $g^{\star}_{r1b}$ via $g^{\star}_{s1-0}$ and $g^{\star}_{s1b-0}$. Basically, the granularity and the level of detail for the specification of (source) degradations and corresponding dependencies has to be at least as large as it is necessary for an accurate and appropriate determination of the entailed target degradations, which have been identified and specified previously. A detailed example of how such an identification can be done utilizing the assumed, given modeling of the involved services will be discussed below. Based on the performed identification of degradations, their detailed particular specification takes place in appropriate granularity/accuracy by utilizing existing best practices, and expert knowledge, combined with particular approaches for determining and measuring all aspects of degradation dependencies with respect to the particular technologies (compare p. 84) involved. The resulting specifications may impose requirements on the actual technical implementation concerning any necessary data source in the SP/MI used for determining concrete degradation dependencies during I/RA runs later, i.e., for collecting all necessary information about concrete instantiations of these identified and specified, general degradation dependencies. In general, access to any management database/tool which holds or allows to determine some of the necessary degradation (dependency) details has to be possible. This includes, e.g., access to a service MIB (Sect. 3.3.5) for determining the dependencies between degradation subjects (resources or service functionalities) or even further their detailed QoX degradation (instantiation) dependencies (compare p. 268 in Sect. 4.3.8), or access to QoS/QoR measure-

ment databases/tools (Sect. 3.4, especially Sect. 3.4.1) to determine accurate, current values for the involved QoX parameters.

Second, based on the identification/specification of all service/resource degradations and their degradation dependencies, the details of these specifications are classified as either static model data or additional, dynamic data (compare p. 143 in Sect. 4.2.2.2).

Third, appropriate m/d i/o data modules for exchanging all the specified static model data and dynamic data to derive concrete degradation dependencies are identified and specified. The implementation of these m/d i/o modules has to fulfill all requirements which have been mentioned above: For example, requirements on the contents of the service MIB (Sect. 3.3.5), which may be used mainly for specifying the static model data of service impact dependency models, as well as requirements imposed on QoX measurement approaches which are utilized.

relationship of identification and specification of degradations

Identification and specification of degradations correspond in the following way to the aspects of degradation specification (Fig. 4.7 on p. 135): The identification of degradations and their dependencies is mainly concerned with the restriction to a suitable degradation scope (degradation subjects and degradation manner). Based on this, the specification of the identified degradations and their dependencies - by re-utilizing best practices, expert knowledge, and particular approaches for determining and specification of dependencies for particular technologies and fields (p. 84) involved - is mainly concerned with the degradation specification details degradation time and/or degradation value accuracy.

particular example how to identify entailing service/resource degradations

In the following, an approach how to identify all relevant (QoS/QoR) degradation dependencies of the service impact dependency model in appropriate accuracy and granularity by using the MNM service model is treated. As an example, all service and to the mail sending functionality $f_{\mathrm{mail/use/send}}$ (covering also $g^{\star}_{s1-1}$), are investigated and identified. In general, it is started from the top-level service degradations given by business impact dependency model instantiation (Sect. 5.3.1), i.e., degradations of particular service functionalities, e.g., $g^{\star}_{s1-1}$, which has functionality $f_{\mathrm{mail/use/send}}$ as its degradation subject.

In the MNM service model with its extension for using UML use case/activity/collaboration diagrams (being attached to the MNM service model instantiation methodology) (see p. 88 in Sect. 3.3.1), service functionalities are represented on the one hand use cases in use case diagrams (high-level view), and on the other hand as an (interaction) process specified by an activity or collaboration diagram (low-level view). The low-level view with activity diagram actually has two refinement steps, an interaction process from the common point of view of customer and provider (representing the service-view), a more refined interaction process from the sole point of the provider's realization. The low-level view with collaboration diagram represents a further refinement to the the one with activity diagram(s), as it presents the detailed specification of the interaction process of the functionality with its access to resources and use of subservices.

Specific instances (e.g., invocation, sessions) of a service functionality are called service functionality interactions (compare Sect. 4.3.6.1). Consequently, the general (interaction) process for service interactions of a functionality class is correspondingly specified by the above mentioned activity diagrams Fig. 5.4 and Fig. 5.5 represent an example for such use case and activity diagrams specifying the service functionality interactions of service functionality $f_{\mathrm{mail/use/send}}$. Similarly, the corresponding UML collaboration diagram of a service functionality would specify the realization of the service functionality interactions in great detail.

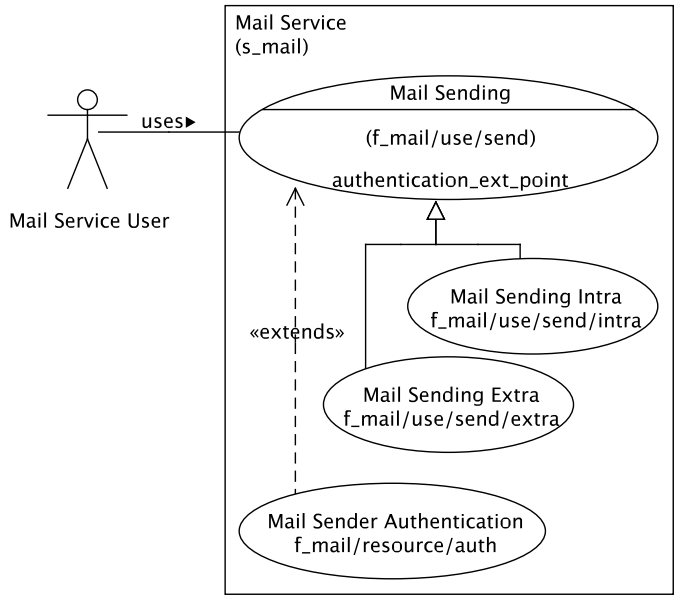

Figure 5.4: Use case diagram for mail sending functionality $f_{\mathrm{mail/use/send}}$

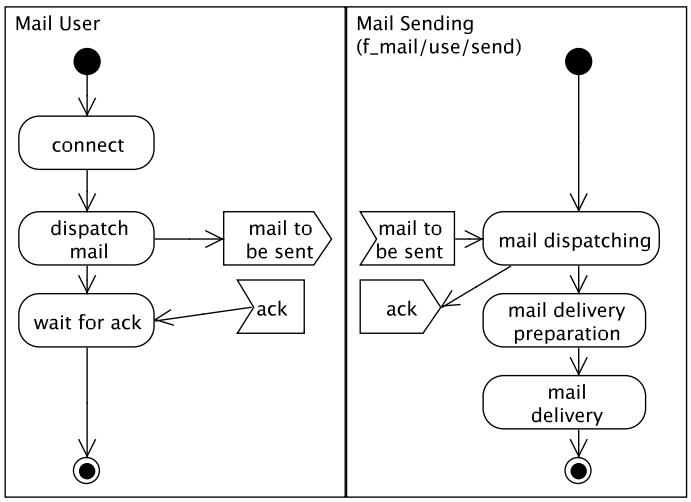

Figure 5.5: Activity diagram for mail sending functionality $f_{\mathrm{mail/use/send}}$ (refinement of Fig. 5.4)

The approach for identifying service/resource degradations and their dependencies is specifically using these UML use case/activity/collaboration diagrams. Roughly speaking, the decomposition of the interaction process of

a given target functionality (here $f_{\mathrm{mail/use/send}}$) into separate phases provides a survey to identify all relevant dependencies from degradations of source degradation subjects to degradations of this target functionality. In this example here only use case and activity diagrams are used. Detailed collaboration diagrams could also have been used to provide more details for the identification of degradation dependencies, especially concerning the dependency on resources which is not completely covered in this example by using only use case/activity diagrams.

interaction phases of the process of a service functionality

In general, for each service functionality which is the degradation subject of one of the identified, top-level service degradations, the existing activity/collaboration diagrams are taken as a survey of the interaction process of this functionality, allowing to identify degradations of the service functionality, here specifically for $f_{\mathrm{mail/use/send}}$ from the activity diagram in Fig. 5.5.

As first step, (sub)phases of the interaction process of the functionality are identified based on the diagram, e.g., exactly corresponding to the activities in the activity diagram, or the diagram partitions in the activity diagram. For example, mail sending functionality $f_{\mathrm{mail/use/send}}$ is composed of the three phases: *mail dispatching* (interaction of mail client and mail server), *mail sending preparation* (mail queuing/routing on the mail server), and *mail delivery* (to destination or at least next hop mail server). This decomposition into the 3 phases can be derived from Fig. 5.5.

As second step, the particular QoS parameters of the functionality (QoS parameters whose scope includes the functionality) are specifically assigned to the separate interaction phases (possibly resulting in an m:n mapping). This is normally done with the help of expert knowledge about the particular realization of the service, and about the particular technologies involved. Partially, the mapping between QoS parameters and interaction phases might also be directly derived from additional annotations in the activity diagrams (compare p. 99 in Sect. 3.5.1) which can be defined as conditions for the execution of the separate activities or sequences thereof in the activity diagram. For $f_{\mathrm{mail/use/send}}$ the resulting mapping between its interaction phases and its QoS parameters may be:

- mail dispatching: mail dispatching delay, mail dispatching rate, overall mail sending delay, overall mail sending rate, availability/reliability of $f_{\mathrm{mail/use/send}}$.
- sending preparation: overall mail sending delay, overall mail sending rate, number of mails lost due to queue overload, availability/reliability of $f_{\mathrm{mail/use/send}}$.
- mail delivery: mail delivery delay, mail delivery rate, overall mail sending delay, overall mail sending rate, availability/reliability of $f_{\mathrm{mail/use/send}}$.

As third step, dependencies between degradations of resources/functionalities used and degradations of the given target service functionality (or more

specifically its particular phase) are derived by using best practices, and expert knowledge about the actual realization and technologies involved. Actually, the service modeling by an activity diagram serves as a survey for experts to identify all dependencies from resources/functionalities to (the phases of) the given target functionality. Alternatively, the detailed collaboration diagram of the target functionality may be used instead of activity diagram to allow to directly identify such dependencies between an interaction phase and another degradation subject (resource or other functionality), as such subjects are directly appearing in the collaboration diagram. Nevertheless, mainly the diagrams itself allow only to derive subject dependencies, potentially combined with degradation manner, i.e., sets of QoS/QoR parameters (in the case of activity diagrams annotated with QoS parameters). The further degradation specification aspects (Fig. 4.7 on p. 135) have to be covered by best practices and detailed expert knowledge. Examples for dependencies between interaction phases and other degradation subjects identified for $f_{\text{mail/use/send}}$ are:

- mail dispatching: mail dispatching delay depends on the IP path delay (of the IP path between client and server), as well as on QoR/QoD parameters (CPU utilization, server restart frequency) of the mail server resource.
- send preparation: overall sending delay here is influenced by DNS response delay, as well as QoR/QoD parameters on the mail server resource
- mail delivery: mail delivery delay here depends e.g., on the IP path delay of the IP path from mail server to destination (here a differentiation of intra domain sending and extra domain sending should be done).

refinement of the interaction phases - differentiating service-view and resource view functionalities

The activity diagram in Fig. 5.5 for $f_{\text{mail/use/send}}$ is actually the activity diagram for the common perspective between provider and customer (service-view). If the activity diagram for $f_{\text{mail/use/send}}$ from sole point of view of provider (realization-view) is used instead, more dependencies can be derived directly from this diagram. To the three identified interaction phases of $f_{\text{mail/use/send}}$ mentioned above, additionally *mail client authorization* (as extension of *mail dispatching*) is to be added, for mail sending interactions which use authorization (e.g., required for mail clients accessing the mail sending functionality from outside the local mail domain). Fig. 5.6 refines Fig. 5.5 by including the functionality $f_{\text{mail/resource/auth}}$ for authenticating the mail sending user, via sub service functionality $f_{\text{afs/use/auth}}$ of service s_{afs}.

This refinement of activity diagrams (from service-view to realization-view) specifically is related to the differentiation of service-view functionalities and resource-view functionalities introduced in Sect. 4.3.6.2. Actually, $f_{\text{mail/resource/auth}}$ in Fig. 5.6 is a resource-view functionality.

Based on the refinement of the phases of $f_{\text{mail/use/send}}$, appropriately refined degradation dependencies of corresponding degradation types (i.e., QoS/QoR parameter sets) can be identified and specified by experts: E.g., authorization delay of $f_{\text{auth/use}}$ adds up to dispatch delay (for

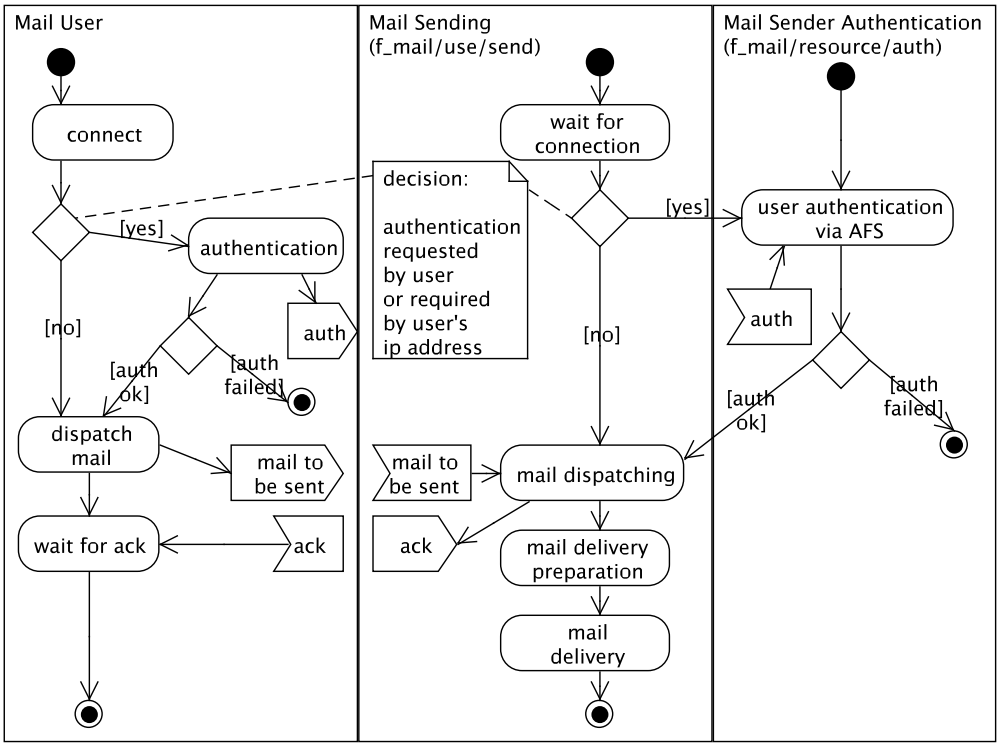

Figure 5.6: Refined activity diagram for mail sending functionality (refinement of Fig. 5.5)

$f_{\mathrm{mail/use/send}}(\textit{authentication}{=}\mathrm{yes})$ only) as well as overall sending delay in the mail dispatch phase, availability of $f_{\mathrm{auth/use}}$ constraints availability of $f_{\mathrm{mail/use/send}}(\textit{authentication}{=}\mathrm{yes})$.

5.3.3 Impact Rating Model Instantiation

For top-down business impact rating model, first, the top-level business degradations identified during business impact dependency model instantiation, are given direct rating definitions based on the initially given business policies concerning the determination and prioritization of relevant types of business degradations. More specifically, each direct rating definition for a type of business degradation has the form of a business degradation characteristic to direct rating mapping, as introduced on p. 295 in Sect. 4.4.1. The sum of all of them represents to rating model.

Second, based on the direct rating definitions of top-level business degradation, indirect rating weighting expressions/algorithms (see p. 295 in Sect. 4.4.1) are added to the business and service impact dependency model for all entailing degradations recursively.

example

In the case of the degradations identified for the example scenario, i.e., $g^{\star}_{b1-1}$ and its entailing degradations, being generalizations of $g^{\star}_{b1-1}$ and respective entailing degradations, this is similarly done as performed for the particular degradations on p. 297 in Sect. 4.4.1. That is, it is started with the SLA penalty

function definition (business degradation characteristic definition) $slp^{\star}_{mail3}$ of $g^{\star}_{b1-1}$ being directly used as a direct rating weighting definition (simple business degradation characteristic unification). Based on this, recursively any related degradations dependencies, which have been also identified before, are followed in reverse direction (from targets to sources) to assign indirect rating weighting expressions/algorithms to all these identified degradation dependencies. Later on during actual I/RA runs these expressions/algorithms will allow the calculation of the indirect rating weighting specifications of actual instantiation of all entailing source degradations ($g^{\star}_{s1-1}$, $g^{\star}_{s1-0}$, $g^{\star}_{s1b-0}$, $g^{\star}_{rb}$, $g^{\star}_{rb1}$).

steps performed

As a whole, for the complete impact rating model instantiation, the following steps are performed: First the specifications described above are performed. Especially the indirect rating weighting expressions/algorithms are added as a part of the respective impact dependency model (compare p. 295 in Sect. 4.4.1). So, as they represent a part of an impact dependency model, their data parts are classified as static model data or additional dynamic data. In contrast, for the direct rating definitions representing the proper rating model such a classification is not necessary, as the whole rating model is regarded as proper, static model data only (compare Fig. 4.36 on p. 191).

Following, all necessary m/d i/o modules for collecting and measuring all the specified model data from the SP/MI are identified and specified: First, this comprises new m/d i/o modules for the rating model definitions (for this purpose they concerned only with model data, no dynamic data). Second, it comprises the adaption of existing m/d i/o modules, or the addition of new m/d i/o modules concerned with degradation dependencies, i.e., specifically with the indirect rating weighting expressions/algorithms.

5.3.4 Recovery Action Dependency Model Instantiation

During recovery action dependency model instantiation, any potential recovery handling, correspondingly particular recovery actions, and their particular influence/effect/execution information (compare p. 173 in Sect. 4.2.3.2) are identified and specified. This is done by using the given modeling of the relevant services, in combination with existing best practices, and expert knowledge for all the above identified, relevant potential resource degradations, in appropriate granularity and accuracy, with respect to their particular resource technologies involved.

example

For each general resource degradation identified before, e.g., for $g^{\star}_{r1}$, handlings and corresponding recovery actions are conceived by experts, e.g., $RecAct^{\star}(g^{\star}_{r1}, 1)$, $RecAct^{\star}(g^{\star}_{r1}, 2)$, $RecAct^{\star}(g^{\star}_{r1}, 3)$, being generalizations of $RecAct(g_{r1}, 1)$, $RecAct(g_{r1}, 2)$, $RecAct(g_{r1}, 3)$ from the example recovery plan $ExRecPlan1$ (compare p. 164 in Sect. 4.2.3.1) respectively, along with appropriately generalized recovery action dependencies.

identification and specification of recovery action dependencies

First, for any resource degradation identified during service impact dependency model instantiation, potential recovery actions are identified and specified with expert knowledge about the actual service implementation, and about concrete technologies involved. Second, all relevant influence/effect/execution dependencies for these recovery actions have to be identified and specified, also mainly with expert knowledge. Nevertheless, an existing modeling of the involved services, as it has been used for service impact dependency model instantiation, may be used in addition for this task to provide the experts with an overview of the interaction process for each relevant service functionality. These specifications may impose requirements on the later, actual implementation for determining concrete instantiations of these dependencies by collecting and measuring the necessary information from the SP/MI during actual I/RA runs later on.

steps performed

Altogether, the following steps have to be done for the instantiation of the recovery action dependency model as a whole: First, the two specification steps mentioned above have to be done. Following, all parts of the specified data are classified as static model data or additional, dynamic data. Last, m/d i/o modules in order to realize the exchange of all specified recovery action dependency model data with the SP/MI, while fulfilling the above mentioned imposed requirements, have to be identified and specified.

5.3.5 Model Adaption Dependency Model Instantiation

For model adaption dependency model instantiation, from the execution information of all potential recovery actions identified before, all potentially necessary model adaption actions, and their dependencies, together with the necessary m/d i/o data modules to perform them are identified and specified.

example

For example, potential necessary model adaption actions for the recovery actions handling of $g^{\star}_{r1}$ are (similarly as done for g_{r2} on p. 322 in Sect. 4.5.2): the update of QoX dependencies in case of changed prioritization of IP traffic on a particular IP link ($RecAct^{\star}(g^{\star}_{r1}, 1)$), the synchronization of all IP path dependencies resulting from changed IP routing ($RecAct^{\star}(g^{\star}_{r1}, 2)$), the re-measuring the available IP path throughput for important, changed IP paths ($RecAct^{\star}(g^{\star}_{r1}, 2)$), the re-measuring the average IP path delay for important, changed IP paths ($RecAct^{\star}(g^{\star}_{r1}, 2)$), the check whether corresponding inventory database have been updated appropriately to reflect changes of IP links ($RecAct^{\star}(g^{\star}_{r1}, 3)$).

steps performed

Model adaption dependency model instantiation as a whole consists of the following steps: First, all potential model adaption actions and their dependencies are identified, and specified, second their data is classified into static or additional dynamic, and third m/d i/o modules for all data parts of the specified model adaption dependency model are conceived. The latter steps comprises actually m/d i/o modules for the planning of model adaption, as

well as m/d i/o modules which are used to actually execute the planned model adaption actions.

5.3.6 Recovery Changes Tracking Dependency Model Instantiation

During recovery changes tracking dependency model instantiation, for all potential model adaption actions identified before, appropriate recovery changes to allow to recognize which model adaption actions have actually to be performed during a concrete I/RA run later on, and their dependencies are identified and specified. This includes a specification of all necessary m/d i/o modules to monitor such recovery changes.

example

Potential individual recovery changes to be monitored during the actual recovery concerning the handling of $g^{\star}_{r1}$ are e.g., monitoring of addition, removal, change of related individual IP routes ($RecAct^{\star}(g^{\star}_{r1}, 2)$), or addition, removal, change of related IP links ($RecAct^{\star}(g^{\star}_{r1}, 3)$).

steps performed

The complete recovery changes tracking dependency model instantiation as a whole comprises the following steps: First, all potential recovery changes and their dependencies are identified, and specified, second their data is classified into static or additional dynamic, and third m/d i/o modules for all data parts of the specified recovery changes tracking dependency model data are specified. The third step comprises m/d i/o modules for the recovery changes tracking dependency model itself, as well as m/d i/o modules for monitoring the recovery changes (represents dynamic data, compare with Fig. 4.36 on p. 191).

5.4 Summary

In this chapter, the instantiation of the generic impact and recovery analysis (I/R analysis) framework developed in Sect. 4 was discussed. Basically, the instantiation comprises component architecture instantiation and model instantiation. Especially, a top-down oriented instantiation methodology was introduced to enable a service provider to establish and appropriately adjust I/R analysis of his real-world services, based on his business policies and requirements derived from the SLAs with his customers. Beyond the top-down oriented instantiation methodology, two other potential instantiation methodologies with another focus were shortly outlined.

For the top-down oriented instantiation methodology particularly the model instantiation was discussed in greater detail in Sect. 5.3, while the component architecture instantiation was only treated roughly in general in Sect. 5.2. A more detailed component architecture instantiation, especially in particular cooperation with the top-down model instantiation is subject to future work.

The next chapter concludes the thesis by assessing the results achieved and summarizing open issues and future work.

Chapter 6

Conclusion and Future Work

Contents

To conclude the thesis, first in Sect. 6.1, the contributions of thesis with the generic framework of Chapter 4 and its instantiation methodology of Chapter 5 are summarized and assessed according to the requirements which have been identified in Chapter 2. Second in Sect. 6.2, open issues left are summarized as well as issues for future further work and potential extensions are presented.

6.1 Contributions

Here the contributions of this thesis with the developed generic framework (Chapter 4) and its instantiation methodology (Chapter 5) are summarized and assessed according to the requirements identified for impact and recovery analysis (I/RA) in Chapter 2.

The requirements of Chapter 2 have been subdivided into the requirement classes R0 (*generic requirements*, Sect. 2.4.1), R1 (*requirements on service modeling in general*, Sect. 2.4.3), R2 (*requirements on service degradation and service impact modeling*, Sect. 2.4.4), R3 (*requirements concerning the modeling of business impact*, Sect. 2.4.5), R4 (*requirements for recovery action modeling*, Sect. 2.4.6), and R5 (*requirements for course of I/R analysis*, Sect. 2.4.7). In the following, for each class the coverage of the requirements by the framework and its instantiation methodology is assessed. First, the class of generic requirements (R0), having been taken directly from the problem statement for impact/recovery analysis (Chapter 1) are treated, then subsequently the other requirement classes (R1 to R5, for an overview see Fig. 2.11 on p. 57 or for details see Fig. 2.13 on p. 67) follow.

Basically, the main contribution of this thesis is the framework for I/R analysis, which first of all covers the generic requirements, R0. The three generic

generic requirements (R0)

requirements are *smooth integration into the service provisioning and service management environment of the service provider* (R0.1), *genericity* (R0.2), and *manageability concerning a changing service provisioning and service management environment* (R0.3). They are covered by the design of the basic component architecture (Sect. 4.2.5, especially, Sect. 4.2.5.2) and the provisioning of an instantiation methodology for the application to concrete service scenarios (Chapter 5): R0.1 is achieved by the basic subdivision of the basic component architecture into the model/dynamic input/output data area and the proper impact/recovery analysis area (compare p. 193 in Sect. 4.2.5.2, and Fig. 4.37). Additionally, R0.1 is supported by the instantiation methodology allowing to instantiate according to a given scenario especially the former area, the model/dynamic input/output data area, which exchanges all model and dynamic data with the service environment. Based on this, R0.2 is achieved by considering multiple, different model/dynamic input/output data modules in the model/dynamic input/output data area, to allow exchange of data with any existing management platform, management tool, management database using the appropriate technology, as far as necessary (compare p. 194 in Sect. 4.2.5.2). Concerning this, R0.2 is also basically supported by the instantiation methodology which comprises the consecutive design of all model/dynamic input/output modules according to their task in the framework. At last, for fulfilling R0.3, the model/dynamic input/output data area as a whole is subdivided into a model/dynamic input/output data engine and a model/-dynamic input/output data access area. The former one is concerned with the control of the synchronization and the exchange of (changed) model and dynamic data between the service environment and the impact/recovery analysis framework, for this task specifically controlling the latter one, which comprises all the above mentioned different model/dynamic input/output data modules (compare p. 194 in Sect. 4.2.5.2).

requirements on service modeling in general (R1)

Regarding the requirement class R1, the contributions of this thesis are the service impact dependency models SIDepMod(Subj:X) (Sect. 4.3.4-4.3.7) and SIDepMod(Coop) (Sect. 4.3.9) being concerned with the requirements of R1. These four requirements on service modeling in general (R1.1-R1.4) are covered in the following way by the framework: Concerning the *consideration of multiple domains* (R1.1), the framework supports the reasoning about (degradation) dependencies on (provider-external) subservices, in addition to the degradation dependencies on resources or provider-internal subservices. Only appropriate model/dynamic input/output modules have to be in place which collect and measure the corresponding dependency information. But, the framework developed so far, is run by a single provider. The operation of the framework in a multi-domain is not explicitly covered yet. That is, the cooperation of multiple providers for I/R analysis in terms of cooperation workflows or data structures for exchanging the relevant management information (i.e., about subservices and their degradation information) is not considered explicitly by the framework. Such a potential extension remains as future work. The *granularity of (service) functionality definition* (R1.2) and the *level of detail regarding realization* (R1.3) is covered by service impact dependency models

SIDepMod(Fcyt) (Sect. 4.3.6) and SIDepMod(FcytInst) (Sect. 4.3.7), which provide a way to specify degradation subjects (resources or service functionalities) in different levels of granularity and accuracy. The coverage of *dependencies with multiplicities* ≥ 1 (R1.4) is generically addressed by the different service impact dependency models (Sect. 4.3.4-Sect. 4.3.10) supporting multiplicities ≥ 1 in general, together with dependency constraints to further restrict or characterize the dependent objects in detail. Furthermore, especially the service impact dependency model SIDepMod(Coop) (Sect. 4.3.9) allows to express complex issues concerning dependencies with multiplicities ≥ 1, like redundancy and load-balancing. In addition to that, the impact rating (Sect. 4.4.1) along such degradation dependencies is supported by a corresponding, recursive derivation of indirect rating weightings from target to source degradations (compare p. 295 in Sect. 4.4.1). But some open issues are still left concerning the indirect rating of dependencies with $m : n$ multiplicity (see p. 296 in Sect. 4.4.1).

requirements on service degradation and quality modeling (R2)

Particular contributions of this thesis with respect to the requirement class R2 are the service impact dependency models SIDepMod(QoX) and SIDepMod(QoXInst) (Sect. 4.3.8), as well as SIDepMod(Coop) (Sect. 4.3.9) and SIDepMod(DynOT) (Sect. 4.3.10). The four requirements on service degradation and quality modeling (R2.1-R2.4) are covered in the following way: In order to cover the requirement *dynamics of dependencies per I/R analysis run* (R2.1), both service impact dependency models SIDepMod(Coop) (Sect. 4.3.9) and SIDepMod(DynOT) (Sect. 4.3.10) were specifically developed. Moreover, the support for multiple *degradation types per resource/-functionality* (R2.2) is explicitly covered by the service impact dependency models SIDepMod(QoX) (Sect. 4.3.8) Based on this, the support for multiple *different value levels/value ranges per degradation type of a resource/functionality* (R2.3) is covered by the service impact dependency model SIDepMod(QoXInst) (Sect. 4.3.8) as well as its extension, the service impact dependency model SIDepMod(Coop) (Sect. 4.3.9). The requirement of supporting multiple *simultaneously occurring degradations per I/RA run* (R2.4) is explicitly covered by the framework. Throughout the whole development of the framework no restrictions were made concerning the number of simultaneously occurring degradations: Multiple resource degradations can be given as essential input to an I/RA run.

requirements concerning the modeling of business impact (R3)

Concerning the requirement class R3, the contribution of this thesis is the particular business impact dependency models BIDepMod(Char) (Sect. 4.3.3). There are three requirements concerning the modeling of business impact (R3.1-R3.3), resulting in the support for different types of business degradation in the framework: Primarily, the *consideration of SLA penalty costs* (R3.1), as a canonical example for business degradations, is required. Therefore, SLA penalties are considered as one basic type of business degradation, e.g., in the business impact dependency model BIDepMod(Char) (Sect. 4.3.3), where they are mainly specified by respective service level penalty functions acting as a particular type of business degradation characteristic function (p. 225 in Sect. 4.3.3). Beyond SLA violation costs, the *integration of*

actual current/future service usage (by users/customers) (R3.2) is demanded. This issue is not explicitly covered fully by the framework, especially concerning e.g., the use of existing estimation techniques (compare Sect. 3.5.2) to specify appropriate types of service degradations and dependent, entailed business degradations. Nevertheless the framework provides a general basis for such dependencies: Service degradations specified with the service impact dependency model SIDepMod(QoXInst) (Sect. 4.3.8) may be used with appropriate degradation type parameter lists (compare p. 268 in Sect. 4.3.8) and corresponding dependencies to business degradations with appropriate business degradation characteristic functions. At last, *further types of potential financial/reputational impact* (R3.3) are supported by allowing to define additional types of business degradations (beyond SLA penalties) in the business impact dependency model BIDepMod(Char) with appropriate business degradation characteristics for their specification (on pages 224–225 in Sect. 4.3.3).

requirements for recovery action modeling (R4)

Regarding the requirement class R4, the contribution of this thesis is the design of impact rating models and recovery action dependency models, including the design of recovery actions, as part of the recovery analysis framework (Sect. 4.4). There are two requirements concerning recovery action modeling (R4.1, R4.2), of which the first one, the *notion of recovery action alternatives* (R4.1) is subdivided into various sub-requirements regarding the recommended recovery plan(s) and its recovery actions: The *granularity of recovery action notion* (R4.1.1) is approached with a generic design of recovery actions (types and instantiations thereof) with individual and coordination parameter list declarations/definitions (p. 299 in Sect. 4.4.2). Support for the *determination of priority/order/scheduling* (R4.1.2) is covered by the coordination parameter list, common to all types of recovery actions (p. 301 in Sect. 4.4.2): For example, this includes the specification of start time, as well as sequential or parallel, and conditional execution of recovery actions within a recommended recovery plan. Specifically, the specification of the demanded *duration needed for each handling of a degradation* (R4.1.3) and of *specific effort necessary per action (staff, extra resources)* (R4.1.4) can be described by appropriate individual recovery action parameter(s) (compare R.4.1.1 above). Last, concerning the *repair costs determination per action* (R4.1.5), repair costs are explicitly introduced as one additional type of a business degradation in the basic framework (p. 158 in Sect. 4.2.3.1), and more specifically in the business impact dependency model BIDepMod(Char) (p. 224 in Sect. 4.3.3) for the particular use within recovery action dependency models.

The second requirement concerning recovery action modeling explicitly demands the *combination of the estimated effects of the recommended recovery with the previously determined pre-recovery business impact* (R4.2). This is explicitly covered by the introduction of particular notions, like recovery action result dependencies, including recovery costs, (p. 163) and reduced impact of a whole recovery plan, as well as their detailed design (compare, e.g., p. 307 in Sect. 4.4.2).

requirements for the course of I/R analysis (R5)

With respect to the requirement class R5 (course of action for I/RA) the contributions of this thesis are a detailed workflow description (basic realized workflow, BRWf, Sect. 4.2.5.6) for all tasks of I/RA, a generic component architecture for the execution of this workflow (basic component architecture, BCArch, Sect. 4.2.5.2), as well as particular implementation guidelines for the main part of this architecture, the I/RA analyzer (Sect. 4.3.11, Sect. 4.4.3, and Sect. 4.5.3). Particularly, there are three requirements concerning the course of I/R analysis (R5.1-5.3): First, the *tasks of impact/recovery analysis* (R5.1) are covered by the (basic) workflow design, i.e., the basic structure of the three refinement steps of the basic workflow BWf, namely BAWf (Sect. 4.2.1), BARWf (Sect. 4.2.2 to 4.2.4), BRWf (Sect. 4.2.5.6). The main tasks for I/R analysis, i.e., impact analysis, recovery analysis, recovery tracking, and corresponding model adaption are fully and explicitly covered. Only the customer notification was not explicitly considered in the development of the framework. Nevertheless, an appropriate customer notification can be based on the different parts of information, which are subsequently determined during the runs of the specified workflows, e.g., the determined actual degradations of services, and the current status of an on-going recovery. The CSM approach (p. 74 in Sect. 3.2) provides a suitable basis for this. The requirement for the *range/depth of I/R analysis* (R5.2) distinguishes the type of degradations, to which the I/R analysis is applied: either to actually current occurring degradations, or to only assumed ones. Basically, the developed framework can be used for both types. But the impact rating and recovery planning, which follow impact analysis, have been designed mainly for targeting at currently occurring degradations. They are mainly suited for reactive incident and problem management. Though, simulation runs with only assumed degradations are possible, but are taken under the assumption that the estimated pre-recovery impact and the correspondingly recommended recovery plan(s) are used for recovering from potential, current degradations. That is, the later usage and above all the initial instantiation of the framework, applied to only assumed degradations, for management areas beyond reactive incident and problem management is not explictly investigated and covered: For example, how to instantiate and use the framework, including all notions introduced and their detailed design, e.g., for the purpose of planning of proactive incident/problem management, or further management areas like availability management. Concerning the coverage of the requirement *urgency level of I/R analysis* (R5.3) by the framework, the following can be said: The use of the framework short-term or long-term is generally possible, concerning the duration of the degradations itself, as well as the duration of their recovery handling, depending on the definitions in the impact dependency models and the recovery action dependency model. The terms short-term and long-term for use with currently occurring degradations correspond to incident management and problem management respectively (compare Fig. 2.12 on p. 66). This is fully covered (see R5.2 above). Regarding the use of only assumed degradations, either short-term or long-term (Fig. 2.12), the explicit instantiation and usage of the framework for related management areas be-

yond reactive incident and problem management, such as proactive incident/problem management are not covered explicitly (compare R5.2 above).

6.2 Open Issues and Future Work

Concerning the generic framework (Chapter 4) as well as its instantiation methodology (Chapter 5), there are some open issues as well as various issues for future work and potential extensions left. Partially these issues have already been mentioned in the last section.

As having also been discussed in the previous section, the generic framework and its instantiation methodology are mainly oriented towards the application for reactive incident and problem management, i.e., the application to currently occurring degradations. Nevertheless, the framework can also be applied in general for assumed degradations, and may therefore also be used for proactive incident and problem management, or provide basic input for availability management, even if such usage was not covered explicitly during the development of the framework.

In the following, first the left open issues for the application of the generic framework and its instantiation methodology to reactive incident and problem management are summarized. Second, future further work and potential extensions concerning the explicit coverage of proactive incident and problem management, the support of availability management, and further management areas is discussed.

open issues for the generic framework applied for reactive incident/problem management

There are some open issues left for the generic framework, concerning especially the application of impact/recovery analysis for reactive incident/problem management, i.e., with the usage for really currently occurring degradations. Most of them have already been mentioned in the last section, so the following lists provides a summary. Additionally, the list is complemented with issues not mentioned in the last section:

open issue I1.1: The explicit investigation about how existing techniques for measurement of current service usage and estimation of future service usage (compare Sect. 3.5.2) can be integrated in the specification and determination of types of service degradations and related business degradations beyond SLA violation costs is left open. Nevertheless the framework provides a general basis for the specification of such types of service degradations with the service impact dependency model SIDepMod(QoXInst) (Sect. 4.3.8): For this, appropriate degradation type parameter lists (compare p. 268 in Sect. 4.3.8) for such types of service degradations as well as corresponding dependencies to entailed business degradations with corresponding business degradation characteristic functions have to be identified and specified.

open issue I1.2: The consideration and support for the impact rating of $m : n$ relationships, i.e., degradation dependencies with $m : n$ multiplicities, is left as an open issue. Specifically, the specification and determination of indirect impact rating weighting for such $m : n$ degradation dependen-

cies, i.e., the weighting distribution from their target degradation to their source degradations has to be determined (compare p. 296 in Sect. 4.4.1).

open issue I1.3: In the current framework only recovery actions targeting directly at the handling of given resource degradations are considered in order to (indirectly) handle entailed service and business degradations. The consideration of high-level (recovery) actions targeting directly at the handling of service or business degradations, so called *service degradation recovery actions* and *business degradation recovery actions* is left as future work (compare p. 157 in Sect. 4.2.3.1). Examples for such service/business degradation recovery actions include conciliation with customers after SLA violations instead of paying the (full) amount of the respective penalties the adaption of the service offering, the adaption of costs and pricing. The consideration of such high-level recovery actions, including the instantiation for concrete scenarios, requires an explicit extension of the design of recovery actions and their dependencies on post-recovery degradations.

open issue I1.4: A further elaboration of the design for recovery plans and the coordination of their recovery actions is left as an open issue (compare p. 301 in Sect. 4.4.2): The existing design of recovery plans supports programming concepts like start time specification, sequential, parallel, and conditional execution of recovery actions within a recovery plan. But more complex programming concepts such as while loops or sub program calls are not supported by this design for recovery plans and for the corresponding determination of the resulting reduced post-recovery impact.

open issue I1.5: The investigation concerning the design of the model/dynamic input/output data modules for the automated self-adaption with (changed) model adaption execution dependencies after a recovery is still left open (compare p. 325 in Sect. 4.5.2).

open issue I1.6: The task of customer notification has not been considered explicitly during the design of the framework. Nevertheless, basically, information derived subsequently during I/RA, such as derived impact information or currently known recovery status, can be used for customer notification. It stays open to investigate in detail how to use explicitly such information derived subsequently during I/RA for customer notification. This includes an appropriate extension of the workflows (e.g., integration with CSM approach discussed on p. 74 in Sect. 3.2) for this purpose as well as the specification of further necessary data structures and/or the necessary adaption of existing data structures.

open issue I1.7: The developed I/RA framework is basically operated in a single domain environment with the possibility to consider and integrate degradations of provider-external subservices. But the explicit cooperation of multiple service providers for the purpose of impact and recovery analysis is not covered by the framework yet, and requires an extension:

Basically, the adaption of the specified workflows and the adaption or new definition of data structures for the exchange of degradation (dependency) information, e.g., for the degradation of provider-external subservices, has to be done for such an extension.

Related to this is an explicit extension of the framework to allow the performing of impact and recovery analysis in a hierarchical manner within a (large) single domain, or beyond this in a multi-domain environment (*hierarchical impact/recovery analysis*). The necessary workflows and data structures for exchanging the necessary information, e.g., about degradation types and dependencies between these degradation types, in such an hierarchical manner have to be identified and specified: This may result e.g., in hierarchical impact dependency models, hierarchical impact rating models, as well as hierarchical recovery dependency models. A hierarchical application of impact and recovery analysis may support a higher adaptability and scalability in a a large single-domain environment or in multi-domain environments, e.g., by making use of the benefits of parallelism.

open issues for the framework instantiation used for reactive incident/problem management

There are also open issues left concerning the instantiation of the generic framework, especially when used for reactive incident/problem management, i.e., for use with real, currently occurring degradations. In fact, the instantiation of the framework discussed in Chapter 5 was mainly targeting at the application for reactive incident/problem management, whereas management areas beyond this were not considered explicitly. The open issues for this instantiation of the framework are:

open issue I2.1: In Chapter 5 generically the instantiation of the framework was subdivided in the parts component architecture instantiation and model instantiation. Based, on this a particular top-down oriented instantiation methodology realizing these parts was treated, though with a strong focus on the model instantiation part, which was specified as a sequence of subsequently performed steps (methodology workflow). Whereas, the component architecture instantiation was only introduced generically (Sect. 5.2). That is, the elaboration of a specifically top-down oriented component architecture instantiation, consisting of subsequent steps, explicitly cooperating with the steps of top-down model instantiation methodology (Sect. 5.3), was not performed and is left as open issue.

open issue I2.2: In Chapter 5, beyond the top-down oriented instantiation methodology also two bottom-up oriented instantiation methodologies (p. 338 in Sect. 5.1), each with a different focus, were shortly outlined. An investigation concerning such other instantiation methodologies and their detailed specification by methodology workflows is left as future work.

explicit extensions of the framework for application beyond reactive incident/problem management

Although I/RA framework may be applied to assumed degradations instead of currently occurring ones, such usage has not been considered explicitly during its development in Chapter 4. Extensions for the explicit application of I/RA beyond reactive incident/problem management, i.e., considering also the explicit application to only assumed degradations, are left for further work and corresponding extensions of the framework. These potential extensions are - partly already mentioned in the last section - summarized and outlined in the following:

extension E1: The particular utilization of impact analysis and recovery recommendations for only assumed degradations has not been investigated explicitly. The developed framework in general is capable of using actual or only assumed resource degradations as input: Mainly, the handling of actually currently occurring resource degradations is covered. Nevertheless, in a simulation-like manner the framework can be applied for assumed resource degradations in order to determine their impact and recommend recovery alternatives. But the specific determination about how to utilize such derived impact information and recovery recommendations for only assumed degradations (simulation or "what if?" analysis), for proactive incident/problem management has not been investigated. This is similarly true for further management areas beyond incident/problem management: It is above all not investigated yet how to utilize such simulations as input for e.g., availability management, especially for network planning, for SLA planning, or for the adaption of pricing (compare Fig. 2.12 on p. 66).

extension E2: Beyond the use of the framework for resource degradations caused mainly by chance, hazard or accidents, the framework may be adapted to be used for resource degradations which are potentially caused by scheduled maintenance actions (*maintenance impact and recovery analysis*): Maintenance actions often have, at least a time-limited, impact on resources/services/business during their realization. Especially entailed business degradations can especially arise if these maintenance actions are performed during normal business hours, e.g., due to the importance of the maintenance actions, instead of being performed during defined maintenance intervals agreed with the customer. Thus, maintenance actions can result to some extent in (preferably only time-limited) degradations. Nevertheless, after their execution also often afore-hand existing degradations may be reduced or even eliminated. That is, maintenance actions are similar to recovery actions, and the framework may be adapted to work with maintenance actions. Either, such an adaption of the I/RA framework may be used to determine what potential resulting impact a set of planned maintenance actions is likely to have (*maintenance impact analysis*). Or alternatively, an adaption of the I/RA framework may be also used to recommended directly a scheduling of planned maintenance actions with minimal resulting impact (*maintenance planing* corresponding to the *recovery planning*). Concluding, instead of being applied only to degradations caused

by hazard (for incident/problem management), an adaption of the I/RA framework may be applied to potential degradations caused by given maintenance actions. Furthermore, instead of being used for the recommendation of a recovery plan consisting of recovery actions, such an adaption of the I/RA framework can be used to recommend an appropriate scheduling for given maintenance actions.

extension E3: As a further extension, the developed framework may be adapted to be applied also to security-related events instead of only being applied to functionality and quality degradations (*security impact/recovery analysis*). That is, instead of being used for fault management (incident/problem management), the framework could be used for security management, namely for security events of resources (short-term incidents or long-term problems), their dependencies on the services and the business, as well as for the recommendation of action plans in order to handle the critical ones of the given events.

APPENDIX A - GENERIC NOTATIONS

Set-Theoretic Formal Notations for Degradations and Degradation Dependencies

Common, general notations concerning sets and maps in general:

sets in general:

$$Set := \text{Class of all sets}, \quad \text{i.e., } A \in Set \text{ means that } A \text{ is a set.} \tag{6.1}$$

If $A, B \in Set$, then

$$A \uplus B \in Set \text{ denotes the disjoint union of } A \text{ and } B. \tag{6.2}$$

$$A \times B \in Set \text{ denotes the cartesian product of } A \text{ and } B. \tag{6.3}$$

$$\mathcal{P}(A) \in Set \text{ denotes the power set of } A, \quad \text{i.e., } B \in \mathcal{P}(A) \Longrightarrow B \subset A. \tag{6.4}$$

maps (or functions) between sets:

$$Abb := \text{Class of all maps from sets to sets} \equiv \{h : A \to B \mid A, B \in Set\} \tag{6.5}$$

If $f \in Abb$, then

$src(f)$ denotes the domain of f (source of f),
$tgt(f)$ denotes the codomain of f (target of f),
$im(f)$ denotes the range of f (image of f). (6.6)

Table 6.1: Formal notations for sets and maps in general

SIDepMod(Sv):

subject:

$Res \in Set$ (resources), and $Sv \in Set$ (services),
$Subj_{sv} := Res \uplus Sv$

additional degradation information beyond subject, e.g., time**:**

$$AddInfo \in Set$$

degradation:

$$g \in Dgr_{Sv} := Subj_{sv} \times AddInfo \tag{6.7}$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (Subj_{sv} \rightarrow Subj_{sv}) \times AddInfo \tag{6.8}$$

Table 6.2: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(Sv)

SIDepMod(SvInst):

subject:

$Res \in Set$ (resources), and $Sv \in Set$ (services), and
$SvInst(s) \in Set$ for each $s \in Sv$ (service instances),
$Subj_{sv\_inst} := Res \uplus Sv \uplus (\cup_{s \in Sv} SvInst(s))$

additional degradation information beyond subject, e.g., time**:**

$$AddInfo \in Set$$

degradation:

$$g \in Dgr_{sv\_inst} := Subj_{sv\_inst} \times AddInfo \tag{6.9}$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (Subj_{sv\_inst} \rightarrow Subj_{sv\_inst}) \times AddInfo \tag{6.10}$$

Table 6.3: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(SvInst)

SIDepMod(Fcty):

subject:

$Res \in Set$ (resources), and $Sv \in Set$ (services), and
$Fcty(s) \in Set$ for each $s \in Sv$ (service functionalities),
$Fcty := \cup_{s \in Sv} Fcty(s)$
with inheritance relationship $FctyInherits(s) \subset Fcty(s) \times Fcty(s)$,
$Subj_{fcty} := Res \cup Fcty \equiv Res \cup (\cup_{s \in Sv} Fcty(s))$

additional degradation information beyond subject, e.g., time**:**

$$AddInfo \in Set$$

degradation:

$$g \in Dgr_{fcty} := Subj_{fcty} \times AddInfo \tag{6.11}$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (Subj_{fcty} \rightarrow Subj_{fcty}) \times AddInfo \tag{6.12}$$

Table 6.4: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(Fcty)

Definitions for data types:

There follow some general notations for data types and their data values (used for parameter list declarations and definitions of particular classes and their instantiations):

data types (to be used as parameter types)**:**

$$DataType \in Set \tag{6.13}$$

data values as instantiations of data types
(to be used as parameter values):

$$DataValue \in Set$$
$$dataValue(.) : DataType \rightarrow \mathcal{P}(DataValue)$$

$\Longrightarrow$ for $t \in DataType$, $dataValue(t) \subset DataValue$ denotes all potential data values (i.e., instantiations) of data type t. (6.14)

conformance of a data value to a data type:
If $v \in DataValue, t \in DataType$, then

$$v \text{ is conforming to type } t :\Leftrightarrow$$
$$v \in dataValue(t) \tag{6.15}$$

inheritance relationship $\rightarrow_{sub}$ between data types:
If $t, t' \in DataType$,

$$t' \rightarrow_{sub} t :\Leftrightarrow dataValue(t') \subset dataValue(t) \tag{6.16}$$

Table 6.5: Formal notation of general data types and their data values (to be used for parameter list declarations and definitions)

Definitions for parameter lists:

General notations of parameter list declarations and their
parameter list definitions (used for parameterization of class and their instantiations):

names (to be used for naming parameters)**:**

$$Name \in Set \text{ and } Name \text{ is countable}$$
$$(e.g., Name = \Sigma^+ \text{ for some given alphabet } \Sigma). \quad (6.17)$$

parameter list declaration, i.e., list of typed parameter names**:**

$$ParamListDecl := \{\pi : N \rightarrow DataType \mid N \subset Name, Nfinite\} \quad (6.18)$$

parameter list definitions,
i.e., instantiations of parameter list declarations**:**

$$ParamListDefn := \{p : N \rightarrow DataValue \mid N \subset Name, Nfinite\}$$

parameter list definitions, or paramater list binding, or parameter list assignment, i.e., list of associations between parameter names and assigned parameters.

(6.19)

conformance of a parameter list definition to a parameter list declaration:
If $p \in ParamListDefn$ and $\pi \in ParamListDecl$, then

$$p \text{ is conforming to declaration } \pi :\Leftrightarrow$$
$$N := src(p) = src(\pi) \in Name \wedge \forall_{n \in N} p(n) \in dataValue(\pi(n))$$

The parameter names of p and the declaration π are the same, and for each parameter name the value $p(n)$ is of conforming type, i.e., is a member of the set of potential data values of the type $\pi(n)$.

(6.20)

Table 6.6: Formal notation for parameter lists in general (based on Table 6.5)

Definitions for parameterized classes (part 1):

General notations for parameterization of
a set of classes $X \in Set$ (of a particular kind) and their instantiations
(e.g., classes of $X = Fcty$ of kind *functionality* - see Table 6.4):

parameter list declaration of class set X:

$$paramListDecl(.) : X \rightarrow ParamListDecl$$

$\Longrightarrow$ for $x \in X$, $paramListDecl(x) \in ParamListDecl$ denotes the parameter list declaration of class x.

(6.21)

set of instantiations of X:

$$InstSet(X) \in Set.$$

each $i \in InstSet(X)$ represents a particular instantiation of some element (some class) in X.

(6.22)

class (or type) of an instantiation:

$$classDecl(.) : Inst(X) \rightarrow X$$

$\Longrightarrow$ for $i \in InstSet(X)$, $x := classDecl(i) \in X$ denotes the particular type (declaration), or class (declaration) x of instantiation i.

(6.23)

parameter list definition of an instantiation:

$$paramListDefn(.) : InstSet(X) \rightarrow ParamListDefn$$

$\Longrightarrow$ for $i \in InstSet(X)$, $p := paramListDefn(i)$ denotes the particular parameter list definition of instantiation i.

(6.24)

Table 6.7: Formal notation for parameterization of a set of classes in general (based on Table 6.6) - part 1 (continuation in Table 6.8)

Definitions for parameterized classes (part 2):

General notations for parameterization of
a set of classes $X \in Set$ (of a particular kind) and their instantiations
(continuation of Table 6.7):

conformance of an instantiation to its class (declaration),
i.e., conformance of its parameter list definition to the parameter list declaration of its class declaration:
If $i \in InstSet(X)$ with
$x := classDecl(i) \in X$, $p := paramListDefn(i) \in ParamListDefn$,
$\pi := paramListDecl(x) \in ParamListDecl$, then

$$i \text{ is conforming to its (class) declaration } x \equiv classDecl(i) :\Leftrightarrow \quad (6.25)$$
$$p \equiv paramListDefn(i) \text{ is conforming to the declaration } \pi \equiv paramListDecl(classDecl(i)).$$

The parameter list definition of an instantiation i has to be conforming to the parameter list declaration of the class $x \equiv classDecl(i)$ of i;
This is equivalent to
$N := src(p) = src(\pi) \subset Name \wedge \forall_{n \in N} p(n) \in dataValue(\pi(n))$,
compare defintion in Table 6.6.

(6.26)

Table 6.8: Formal notation for parameterization of a set of classes in general - part 2 (continuation of Table 6.7)

SIDepMod(FctyInst):

subject:

$Res \in Set$ (resources), and $Sv \in Set$ (services), and
$Fcty(s) \in Set$ for each $s \in Sv$ (service functionalities),
with inheritance relationship $FctyInherits(s) \subset Fcty(s) \times Fcty(s)$,
$Subj_{fcty\_inst} := InstSet(Subj_{fcty}) \equiv InstSet(Res \cup (\cup_{s \in Sv} Fcty(s)))$

(for $InstSet(.)$ see Table 6.7/6.8)

additional degradation information beyond subject, e.g., time**:**

$$AddInfo \in Set$$

degradation:

$$g \in Dgr_{fcty\_inst} := Subj_{fcty\_inst} \times AddInfo \quad (6.27)$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (Subj_{fcty\_inst} \rightarrow Subj_{fcty\_inst}) \times AddInfo \quad (6.28)$$

Table 6.9: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(FctyInst)

SIDepMod(QoX):

subject:

reusing subject specification $Subj_X$
with $X \in \{Sv, SvInst, Fcty, FctyInst\}$

manner:

$$QoXDgrType \in Set$$

scope:

$$Scope_{qox} := Subj_X \times QoXDgrType$$

additional degradation information beyond scope, e.g., time**:**

$$AddInfo \in Set$$

degradation:

$$g \in Dgr_{qox} := Scope_{qox} \times AddInfo \tag{6.29}$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (Scope_{qox} \rightarrow Scope_{qox}) \times AddInfo \tag{6.30}$$

Table 6.10: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(QoX)

SIDepMod(QoXInst):

subject:

reusing subject specification $Subj_X$
with $X \in \{Sv, SvInst, Fcty, FctyInst\}$

manner:

$$QoXDgrType \in Set$$

scope + value accuracy $\times$ time:

$$Dgr_{qox\_inst} := Subj_X \times InstSet(QoXDgrType),$$

(note: $q \in InstSet(QoXDgrType)$ can e.g., be specified as a function of time)

(for $InstSet(.)$ see Table 6.7/6.8)

degradation:

$$g \in Dgr_{qox\_inst} \equiv Subj_X \times InstSet(QoXDgrType) \quad (6.31)$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in Dgr_{qox\_inst} \rightarrow Dgr_{qox\_inst} \quad (6.32)$$

Table 6.11: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(QoXInst)

SIDepMod(Coop):

subject:

reusing subject instantiation specification $Subj_X$
with $X \in \{FctyInst\}$

manner:

$$QoXDgrType \in Set$$

scope + value accuracy $\times$ time:

$$Dgr_{coop} := InstSet(Subj_X \times QoXDgrType),$$

(note: $InstSet(QoXDgrType)$ can e.g., be specified as a function of time)

(for $InstSet(.)$ see Table 6.7/6.8)

degradation:

$$g \in Dgr_{coop} \equiv InstSet(Subj_X \times QoXDgrType) \quad (6.33)$$

degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in Dgr_{coop} \rightarrow Dgr_{coop} \quad (6.34)$$

Table 6.12: Formal notation for degradation (dependencies) of impact dependency model SIDepMod(Coop)

BIDepMod(Char):

degradation degradation characteristic:

$$AbbSignature := \{sign = (D, C) | D, C \in Set\},$$

(the set of map signatures, or function signatures, each $sign$ declaring a map/function with domain D and codomain C).

(6.35)

$$BizDegrCharDecl := \{char \in AbbSignature | \ char \text{ specifies a business degradation characteristic declaration } \},$$

$$BizDegrChar := InstSet(BizDegrCharDecl)$$

(for $InstSet(.)$ in general see Table 6.7/6.8, here specifically $InstSet(BizDegrCharDecl)$ means the set of potential function of time (definitions) conforming to some function of time declaration of BizDegrCharDecl, i.e., $InstSet(BizDegrCharDecl) \equiv BizDegrChar = \{char \in Abb | \ char$ specifies a business degradation characteristic$\})$

(6.36)

additional degradation information beyond business degradation characteristic:

$$AddInfo \in Set$$

degradation:

$$g \in BizDgr_{char} := BizDegrChar \times AddInfo \equiv Inst(BizDegrCharDecl) \times AddInfo. \quad (6.37)$$

service-to-business degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in SvDgr \times (BizDegrChar \times AddInfo)$$

with $SvDgr := (\cup_{X \in \{Sv, SvInst, Fcty, FctyInst, QoX, QoXInst, Coop, DynOT\}} Dgr_X)$

(SvDgr comprising all potential degradation specifications of any service impact dependency model, compare Table 6.2 to Table 6.12)

(6.38)

business-to-business degradation dependency:

$$g_{src} \rightarrow g_{tgt} \in (BizDegrChar \rightarrow BizDegrChar) \times AddInfo \quad (6.39)$$

Table 6.13: Formal notation for degradation (dependencies) of impact dependency model BIDepMod(Char)

APPENDIX B - EXAMPLE CODE

Example Implementation for Impact Dependency Models

In the following, in Code Listing 6.1, an example implementation for impact dependency models (Sect. 4.3.2 to Sect. 4.3.10) of the impact analysis framework is given. Actually this example implementation comprises the definition of logical facts and rules for the *Flora2* system.

Flora2 [YK00, YKZ03], which is based on XSB prolog [xsb], is a deductive database approach (see Sect. 3.6.1). Particularly, it implements an extension of the object-oriented specification language F-Logic (p. 107 in Sect. 3.6.1), which is often used in combination with the deductive database approach. But beyond being a pure query system, which deductive databases usually are only, Flora2 is a full-featured (logical) programming language. In addition to this, it also integrates the reasoning by second-order logic (syntactically second-order, semantically first order) in HiLog [CKW93] as well as the reasoning by backtrackable logical updates based on Transaction Logic (see Sect. 3.6.2). Especially the latter feature, which among other features is suitable for planning tasks, will allows for a later easy extension of recovery planning with this example implementation for impact analysis.

Code Listing 6.1: Flora2 code for impact analysis and impact rating, i.e., specification and derivation of degradations, their dependencies, and their ratings

```
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% example implementation in flora2 DDB system
//% for degradation dependencies using the impact dependency models
//% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
//%
//% for some degradation subjects and related degradation dependencies
//% of the example scenario
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% main file "ira.flr"
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% 0. load flora2 system module predicates

?- [prettyprint >> pp].

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% generic definitions for dependencies

#include "dependency_generic.flr"

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% generic definitions for (degradation) subjects and their dependencies
#include "subject_generic.flr"

//% examples of subject (dependencies)
//% from scenario and example queries
#include "subject_example.flr"

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% generic definitions for degradations and their dependencies
#include "degradation_generic.flr"

//% examples of
//% degradation dependencies, initial resource degradations, direct rating
//% for scenario and exmaple queries
#include "degradation_example.flr"

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% example implementation in flora2 DDB system
//% for degradation dependencies using the impact dependency models
//% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
//%
//% for some degradation subjects and related degradation dependencies
//% of the example scenario
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% file "dependency_generic.flr"
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

```

```
//% I.1 totaly generic definitions:
//% reflexive closure, transitive closure, product, etc.
//% pf relations (predicates)

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% add a log (as base of a derivation proof) to unary ?rel
log1(?rel)(?S, ?log) :- ?rel(?S), ?log = ?rel(?S).

//% add a log (as base of a derivation proof) to binary ?rel
log2(?rel)(?S, ?T, ?log) :- ?rel(?S, ?T), ?log = ?rel(?S, ?T).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% reflexive closure of elements true with unary predicate ?test
Refl(?test)(?S, ?T) :- ?test(?S), ?test(?T), ?S = ?T.
//
Refl_log(?test)(?S, ?T, ?log) :- ?test(?S), ?test(?T), ?S = ?T, ?log=(?S=?T).
//
Refl_log2(?test)(?S, ?T, ?log) :- ?test(?S, ?log1), ?test(?T, ?log2), ?S = ?T
     , ?log=Refl_log2(?log1, ?log2, ?S=?T).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% transitive closure of ?rel
Trans(?rel)(?S, ?T) :- ?rel(?S, ?T).
Trans(?rel)(?S, ?T) :- ?rel(?S, ?R), Trans(?rel)(?R, ?T).
//
Trans_log(?rel)(?S, ?T, ?log) :- ?rel(?S, ?T), ?log=?rel(?S, ?T).
//Trans_log(?rel)(?S, ?T, ?log) :- ?rel(?S, ?R), Trans_log(?rel)(?R, ?T, ?
     log2), ?log = Trans_log(?rel(?S, ?R), ?log2).
Trans_log(?rel)(?S, ?T, ?log) :- ?rel(?S, ?R), Trans_log(?rel)(?R, ?T, ?log2)
     , ?log = [?rel(?S, ?R) | ?log2].
//
Trans_log2(?rel)(?S, ?T, ?log) :- ?rel(?S, ?T, ?log).
Trans_log2(?rel)(?S, ?T, ?log) :- ?rel(?S, ?R, ?log1), Trans_log2(?rel)(?R, ?
     T, ?log2), ?log = [?log1, ?log2].
//Trans_log2(?rel)(?S, ?T, ?log) :- ?rel(?S, ?R, ?log1), Trans_log2(?rel)(?R,
      ?T, ?log2), ?log = Trans_log2(?log1, ?log2).
//Trans_log2(?rel)(?S, ?T, ?log) :- ?rel(?S, ?R, ?log1), Trans_log2(?rel)(?R,
      ?T, ?log2), append(?log1, ?log2, ?log)@_prolog(lists).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% transitive, reflexive closure of ?rel
TransRefl(?_, ?test)(?S, ?T) :- Refl(?test)(?S, ?T).
TransRefl(?rel,?_)(?S, ?T) :- Trans(?rel)(?S, ?T).
//
TransRefl_log(?_, ?test)(?S, ?T, ?log) :- Refl_log(?test)(?S, ?T, ?log).
TransRefl_log(?rel,?_)(?S, ?T, ?log) :- Trans_log(?rel)(?S, ?T, ?log).
//
TransRefl_log2(?_, ?test)(?S, ?T, ?log) :- Refl_log2(?test)(?S, ?T, ?log).
TransRefl_log2(?rel,?_)(?S, ?T, ?log) :- Trans_log2(?rel)(?S, ?T, ?log).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% cartesian product of ?rela and ?relb
Prod(?rela, ?relb)(?S, ?T) :- ?rela(?S, ?R), ?relb(?R, ?T).
//
Prod_log(?rela, ?relb)(?S, ?T, ?log) :- ?rela(?S, ?R), ?relb(?R, ?T), ?log =
     Prod_log(?rela(?S, ?R), ?relb(?R, ?T)).
//
Prod_log2(?rela, ?relb)(?S, ?T, ?log) :- ?rela(?S, ?R, ?log1), ?relb(?R, ?T,
     ?log2), ?log = Prod_log2(?log1, ?log2).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% union of all relations in list ?Lrel
Union(?Lrel)(?S, ?T) :- member(?rel, ?Lrel)@_prolog(basics), ?rel(?S, ?T).
//
Union_log(?Lrel)(?S, ?T, ?log) :- member(?rel, ?Lrel)@_prolog(basics), ?rel(?
    S, ?T), ?log = ?rel(?S, ?T).
//
Union_log2(?Lrel)(?S, ?T, ?log) :- member(?rel, ?Lrel)@_prolog(basics), ?rel
    (?S, ?T, ?log).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% I.2 generic definitions for dependencies

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% I.2.a) generic definitions concerning combination
//% of dependencies and inheritance

spec_tgt(?dep, ?subs)(?S, ?T) :- ?subs(?Tp, ?T), ?dep(?S, ?Tp).
spec_src(?dep, ?subs)(?S, ?T) :- ?subs(?S, ?Sc), ?dep(?Sc, ?T).
spec(?dep, ?subs)(?S, ?T) :- spec_tgt(?dep, ?subs)(?S, ?T).
spec(?dep, ?subs)(?S, ?T) :- spec_src(?dep, ?subs)(?S, ?T).
//
spec_tgt_log(?dep, ?subs)(?S, ?T, ?log) :- ?subs(?Tp, ?T), ?dep(?S, ?Tp, ?
    log1), ?log = [?log1, spec_tgt(?Tp, ?T)].
spec_src_log(?dep, ?subs)(?S, ?T, ?log) :- ?subs(?S, ?Sc), ?dep(?Sc, ?T, ?
    log2), ?log = [spec_src(?Sc, ?S), ?log2].
spec_log(?dep, ?subs)(?S, ?T, ?log) :- spec_tgt_log(?dep, ?subs)(?S, ?T, ?log
    ).
spec_log(?dep, ?subs)(?S, ?T, ?log) :- spec_src_log(?dep, ?subs)(?S, ?T, ?log
    ).

genr_tgt(?dep, ?subs)(?S, ?T) :- ?subs(?T, ?Tc), ?dep(?S, ?Tc).
genr_src(?dep, ?subs)(?S, ?T) :- ?subs(?Sp, ?S), ?dep(?Sp, ?T).
genr(?dep, ?subs)(?S, ?T) :- genr_tgt(?dep, ?subs)(?S, ?T).
genr(?dep, ?subs)(?S, ?T) :- genr_src(?dep, ?subs)(?S, ?T).
//
genr_tgt_log(?dep, ?subs)(?S, ?T, ?log) :- ?subs(?T, ?Tc), ?dep(?S, ?Tc, ?
    log1), ?log = [?log1, genr_tgt(?T, ?Tc)].
genr_src_log(?dep, ?subs)(?S, ?T, ?log) :- ?subs(?Sp, ?S), ?dep(?Sp, ?T, ?
    log2), ?log = [genr_src(?Sp, ?S), ?log2].
genr_log(?dep, ?subs)(?S, ?T, ?log) :- genr_tgt_log(?dep, ?subs)(?S, ?T, ?log
    ).
genr_log(?dep, ?subs)(?S, ?T, ?log) :- genr_src_log(?dep, ?subs)(?S, ?T, ?log
    ).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

spec_or_genr_tgt_log(?_)(?T, ?Tsub, ?log) :- ?T = ?Tsub, ?log=nop.
spec_or_genr_tgt_log(?subs)(?T, ?Tsub, ?log) :- ?subs(?T, ?Tsub), ?log =
    spec_tgt(?T, ?Tsub).
spec_or_genr_tgt_log(?subs)(?T, ?Tsub, ?log) :- ?subs(?Tsub, ?T), ?log =
    genr_tgt(?T, ?Tsub).

spec_or_genr_src_log(?_)(?Ssub, ?S, ?log) :- ?Ssub = ?S, ?log=nop.
spec_or_genr_src_log(?subs)(?Ssub, ?S, ?log) :- ?subs(?Ssub, ?S), ?log =
    spec_src(?Ssub, ?S).
spec_or_genr_src_log(?subs)(?Ssub, ?S, ?log) :- ?subs(?S, ?Ssub), ?log =
    genr_src(?Ssub, ?S).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% I.2.b) definition of recursive dependency tracing
//% (combined with following inheritance/subsumption relationships where
    necessary)

```

```
207 isparent(?P, ?C) :- has_parent_functionality(?C, ?P).
208
209 comprises(?P, ?C) :- is_comprised_by(?C, ?P).
210
211 //% dependency tracing with attached ?log (proof)
212 //% based on the predicates:
213 //% has_dependency_to(?src, ?tgt) for specifying direct dependencies
214 //% comprises(?parent, ?child) for specifying inheritance/subsumption
215 dep_all_log(?s, ?t, ?log) :-
216   //Prod_log2(
217    Prod_log2(
218     spec_or_genr_src_log(Trans(comprises)),
219     Trans_log2(Union_log2([log2(has_dependency_to), Prod_log2(Prod_log2(log2(
          has_dependency_to), spec_or_genr_tgt_log(Trans(comprises))), log2(
          has_dependency_to)) ]))
220    )
221     //,spec_or_genr_tgt_log(Trans(comprises)))
222   (?s, ?t, ?log).
223
224 //% dependency tracing with ?log removed
225 dep_all(?s, ?t) :- dep_all_log(?s, ?t, ?_).
226
227 //?- dep_all_log(?s, ?t, ?log).
228 //?- dep_all_log(?src, ?tgt:f_mail_use_send[isAuthenticated->yes], ?_).
229
230 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
231 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
232 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
233 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
234
235 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
236 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
237 //%
238 //% example implementation in flora2 DDB system
239 //% for degradation dependencies using the impact dependency models
240 //% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
241 //%
242 //% for some degradation subjects and related degradation dependencies
243 //% of the example scenario
244 //%
245 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
246 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
247 //%
248 //% file "subject_generic.flr"
249 //%
250 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
251 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
252
253 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
254 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
255 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
256 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
257 //% II.1.a) general definitions for subjects
258
259 // flora types for subject
260 functionality::subject.
261 resource::subject.
262
263 // subject kinds
264 is_subject_kind(?K) :- ?K == functionality.
265 is_subject_kind(?K) :- ?K == resource.
266
267 // subject classes
268 is_subject_class(?S) :- ?S:resource.
269 is_subject_class(?S) :- ?S:functionality.
270 // subject classes together with their kind
271 is_subject_class(?S, ?K) :- is_subject_class(?S), ?S:?K, is_subject_kind(?K).
272
273 // subject instantiations
274 is_subject_instance(?S) :- is_subject_class(?SC),?S:?SC.
275 // subjects instantiations together with their class
```

```
276 is_subject_instance(?S, ?SC) :- is_subject_instance(?S), ?S:?SC,
        is_subject_class(?SC).
277 // subjects instantiations together with their class and kind
278 is_subject_instance(?S, ?SC, ?K) :- is_subject_instance(?S), ?S:?SC,
        is_subject_class(?SC, ?K).
279
280 // subjects in general
281 is_subject(?S) :- is_subject_class(?S).
282 is_subject(?S) :- is_subject_instance(?S).
283
284 is_subject(?SC, ?SC) :- is_subject_class(?SC).
285 is_subject(?S, ?SC) :- is_subject_instance(?S, ?SC).
286
287 is_subject(?SC, ?SC, ?K) :- is_subject_class(?SC, ?K).
288 is_subject(?S, ?SC, ?K) :- is_subject_instance(?S, ?SC, ?K).
289
290 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
291 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
292 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
293 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
294 //% II.1.b) general definitions for subject dependencies
295
296 pertains_to_service(?F, ?S) :- is_top_functionality(?F, ?S).
297 pertains_to_service(?F, ?S) :- pertains_to_service(?P, ?S),
        has_parent_functionality(?F, ?P).
298
299 has_parent_functionality_rec(?F, ?P) :- has_parent_functionality(?F, ?P).
300 has_parent_functionality_rec(?F, ?P) :- has_parent_functionality_rec(?F, ?B),
        has_parent_functionality(?B, ?P).
301
302 is_comprised_by(?B, ?A) :- has_parent_functionality_rec(?B, ?A).
303 is_comprised_by(?B, ?A) :- ?A:?C, ?C::subject, ?B:?A.
304 //%is_comprised_by(?B, ?A) :- ?A::subject, ?B:?A.
305
306 // % for using flogic inheritance for fcty class hierarchies
307 // is_comprised_by(?B, ?A) :- ?B::?A, ?A:functionality.
308
309 // % <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
310 // % <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
311
312 the_any_instance(?F):?F :- ?F:functionality.
313 the_any_instance(?F):?F :- ?F:resource.
314
315 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
316 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
317 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
318 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
319 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
320 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
321 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
322 //%
323 //% example implementation in flora2 DDB system
324 //% for degradation dependencies using the impact dependency models
325 //% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
326 //%
327 //% for some degradation subjects and related degradation dependencies
328 //% of the example scenario
329 //%
330 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
331 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
332 //%
333 //% file "subject_example.flr"
334 //%
335 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
336 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
337
338 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
339 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
340 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
341 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
342 //% II.2.a) definitions for subject dependencies of scenario
```

```

s_mail:service.
s_web:service.

s_ip:service.
s_dns:service.
s_afs:service.
s_nfs:service.
s_db:service.
s_ldap:service.
s_mailext:service.
s_db:service.

s_new:service.

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

is_top_functionality(f_new:functionality, s_new).


//% use flogic internal inheritance for subject class hierarchies
f_new_use::f_new.
f_new_use:functionality.

//% the next line must be first:
//test1:f_new_use[customer -> lmu].
?X[customer -> undef:customer] :- ?X:f_new, not( ?X[customer -> ?_] ).

f_new_use_func1::f_new_use.
f_new_mgmt::f_new.

has_dependency_to(r_new1:resource, f_new_use).

has_dependency_to(r_new1_lmu:resource, test1:f_new_use[customer -> lmu]).

has_parent_functionality(?C, ?P) :- ?C:functionality, ?P:functionality,
    pertain_to_same_service(?C, ?P), ?C::?P.

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

is_top_functionality(f_mail:functionality, s_mail).

has_parent_functionality(f_mail_use, f_mail).
has_parent_functionality(f_mail_mgmt:functionality, f_mail).
has_parent_functionality(f_mail_use_send:functionality, f_mail_use).
has_parent_functionality(f_mail_use_send_intra:functionality, f_mail_use_send).
has_parent_functionality(f_mail_use_send_extra:functionality, f_mail_use_send).
has_parent_functionality(f_mail_use_recv:functionality, f_mail_use).

//?X[customer -> undef:customer] :- ?X:f_mail_use_recv.
?X[customer -> undef:customer] :- ?X:f_mail_use_recv, not( ?X[customer -> ?_] ).

//%test5:f_mail_use_recv.
has_parent_functionality(f_mail_use_recv_intra:functionality, f_mail_use_recv).
has_parent_functionality(f_mail_use_recv_extra:functionality, f_mail_use_recv).
has_parent_functionality(f_mail_use_mailboxaccess:functionality, f_mail_use).
has_parent_functionality(f_mail_use_customize:functionality, f_mail_use).
has_parent_functionality(f_mail_use_webmail:functionality, f_mail_use).

is_top_functionality(f_ip, s_ip).
has_parent_functionality(f_ip_use, f_ip).
has_parent_functionality(f_ip_use_connectivity, f_ip_use).
has_parent_functionality(f_ip_use_loadbalance, f_ip_use).
has_parent_functionality(f_ip_mgmt, f_ip).

is_top_functionality(f_dns, s_dns).
```

```
408 has_parent_functionality(f_dns_use, f_dns).
409 has_parent_functionality(f_dns_mgmt, f_dns).
410
411 is_top_functionality(f_web, s_web).
412 has_parent_functionality(f_web_use, f_web).
413 has_parent_functionality(f_web_use_special, f_web_use).
414 has_parent_functionality(f_web_use_special_webmail, f_web_use_special).
415 has_parent_functionality(f_web_mgmt, f_web).
416
417 is_top_functionality(f_afs, s_afs).
418 has_parent_functionality(f_afs_use, f_afs).
419 has_parent_functionality(f_afs_use_auth, f_afs_use).
420 has_parent_functionality(f_afs_use_store, f_afs_use).
421 has_parent_functionality(f_afs_mgmt, f_afs).
422
423 is_top_functionality(f_nfs, s_nfs).
424 has_parent_functionality(f_nfs_use, f_nfs).
425 has_parent_functionality(f_nfs_mgmt, f_nfs).
426
427 is_top_functionality(f_db, s_db).
428 has_parent_functionality(f_db_use, f_db).
429 has_parent_functionality(f_db_mgmt, f_db).
430
431 is_top_functionality(f_ldap, s_ldap).
432 has_parent_functionality(f_ldap_use, f_ldap).
433 has_parent_functionality(f_ldap_mgmt, f_ldap).
434
435 is_top_functionality(f_mailext, s_mailext).
436 has_parent_functionality(f_mailext_use, f_mailext).
437
438 r_mailout:resource.
439 r_mailrelay1:resource.
440 r_mailrelay2:resource.
441 r_blacklist:resource.
442 r_spamcheck:resource.
443 r_viruscheck:resource.
444 r_graylist:resource.
445 r_mailin:resource.
446 r_mailin_studlmu:resource.
447 r_mailin_lmu:resource.
448 r_mailin_tu:resource.
449 r_confsv:resource.
450
451 has_dependency_to(f_ip_use_connectivity, f_mail_use).
452 has_dependency_to(f_dns_use, f_mail_use).
453 has_dependency_to(f_afs_use_auth, f_mail_use_mailboxaccess).
454
455 has_dependency_to(r_mailin, f_mail_use_mailboxaccess).
456 has_dependency_to(r_mailout, f_mail_use_send).
457 has_dependency_to(f_mailext_use, f_mail_use_send_extra).
458
459 _#:f_mail_use_send[isAuthenticated->yes].
460 has_dependency_to(f_afs_use_auth, ?X) :- ?X:f_mail_use_send[isAuthenticated->
        yes].
461 //% test:
462 has_dependency_to(r_mailsec:resource, ?X) :- ?X:f_mail_use_send[
        isAuthenticated->yes].
463
464 has_dependency_to(r_mailin, f_mail_use_recv).
465 has_dependency_to(r_mailin_tum, f_mail_use_recv1:f_mail_use_recv[customer ->
        tum:customer]).
466 has_dependency_to(r_mailin_lmu, f_mail_use_recv2:f_mail_use_recv[customer ->
        lmu:customer]).
467
468 has_dependency_to(f_mailext_use, f_mail_use_recv_extra).
469
470 has_dependency_to(r_mailrelay1, f_mail_use_recv).
471 has_dependency_to(r_mailrelay2, f_mail_use_recv).
472
473 has_dependency_to(r_spamcheck, f_mail_use_recv).
474 has_dependency_to(r_viruscheck, f_mail_use_recv).
```

```

has_dependency_to(r_blacklist, f_mail_use_recv_extra).
has_dependency_to(r_graylist, f_mail_use_recv_extra).

has_dependency_to(f_mail_use_mailboxaccess, f_mail_use_customize).
has_dependency_to(r_confsv, f_mail_use_customize).

has_dependency_to(f_mail_use_mailboxaccess, f_mail_use_webmail).
has_dependency_to(f_mail_use_send, f_mail_use_webmail).
has_dependency_to(f_mail_use_recv, f_mail_use_webmail).
has_dependency_to(f_web_use_special_webmail, f_mail_use_webmail).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

has_dependency_to(r_rtcore:resource, f_ip_use).
has_dependency_to(r_lbswitch:resource, f_ip_use_loadbalance).

s_test1:service.
is_top_functionality(f_test1_use:functionality, s_test1).
has_dependency_to(f_ip_use, f_test1_use).

has_dependency_to(f_ip_use_connectivity, f_dns_use).

has_dependency_to(f_ip_use_connectivity, f_afs_use).
has_dependency_to(f_dns_use, f_afs_use).

has_dependency_to(f_ip_use_connectivity, f_nfs_use).

has_dependency_to(f_ip_use_connectivity, f_db_use).
has_dependency_to(f_dns_use, f_db_use).

has_dependency_to(f_ip_use_connectivity, f_ldap_use).
has_dependency_to(f_dns_use, f_ldap_use).

has_dependency_to(f_ip_use_connectivity, f_mailext_use).
has_dependency_to(f_dns_use, f_mailext_use).

//

r_rt_lrz:resource.
r_sw2:resource.
r_ip_link(r_rt_lrz, r_sw2):resource.
r_ip_link1 :=: r_ip_link(r_rt_lrz, r_sw2).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% II.2.b) exmaple queries for subject dependencies

//?- dep_all_log(?s:subject, ?t:subject, ?log).
?- dep_all_log(?src, ?tgt:f_mail_use_send[isAuthenticated->yes], ?_).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% example implementation in flora2 DDB system
//% for degradation dependencies using the impact dependency models
//% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
//%
//% for some degradation subjects and related degradation dependencies
//% of the example scenario
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//%
//% file "degradation_generic.flr"
```

```
//%
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% III.1) general definitions for degradations and degradation dependencies

is_subsumed(?a) :- is_subsumed_by(?a, ?_).

dep_all_nosubsumption(?S, ?T) :- dep_all(?S, ?T), not is_subsumed(?T).

//% definitions for any degradation type
//% for convenience purposes only

?GS[ target -> ?GT ] :- ?GS:dgr, ?GT:dgr, has_dependency_to(?GS, ?GT).
?GT[ source -> ?GS ] :- ?GS:dgr, ?GT:dgr, has_dependency_to(?GS, ?GT).

?GS[ target_recursive -> ?GT ] :- ?GS:dgr, ?GT:dgr, dep_all(?GS, ?GT), not is_subsumed_by(?GS, ?_).
?GT[ source_recursive -> ?GS ] :- ?GS:dgr, ?GT:dgr, dep_all(?GS, ?GT), not is_subsumed_by(?GS, ?_).

//% subsumed degradations are only intermediate degradations
//% being subsumed by a (more superior) one
//% e.g. two particular portions of the rise of mail sending delay
//% (both rises caused from different source degradations)
//%
//% such subsumed degradations are not taken into account
//% for indirect rating of source degradations
//% as their subsuming degradation already propagtes
//% the respective rating portions to the sources
?GS[
  is_subsumed_by -> ?GT
] :- ?GT:dgr, ?GS:dgr, is_subsumed_by(?GS, ?GT).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% degradations and degradation dependencies of SIDepMod(Subj)
dgr_subj::dgr.

//% derive degradation(Subj) dependencies from subject dependencies
g_subj(?GS, ?tgt_subj):dgr_subj[
  subject -> ?tgt_subj
] :- ?GS:dgr_subj[
  subject -> ?src_subj
], dep_all(?src_subj, ?tgt_subj).
has_dependency_to(?GS, ?GT) :- ?GS:dgr_subj, ?GT:dgr_subj = g_subj(?GS, ?_).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% degradations and degradation dependencies of SIDepMod(QoX)
//dgr_qox::dgr.
//is_subsumed_by(?dgr_subj, ?dgr_qox) :-
//?dgr_subj:dgr_subj[ subject -> ?subj ], ?dgr_qox:dgr_qox[ subject -> ?subj ].

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

//% degradations and degradation dependencies of SIDepMod(QoXInst)
dgr_qoxinst::dgr.

is_subsumed_by(?dgr_subj, ?dgr_qoxinst) :-
?dgr_subj:dgr_subj[ subject -> ?subj ], ?dgr_qoxinst:dgr_qoxinst[ subject -> ?subj ].

```

```
613 //% derive degradation(QoXInst) dependencies from subject dependencies
614 g_subj(?GS, ?tgt_subj):dgr_subj[
615   subject -> ?tgt_subj
616 ] :- ?GS:dgr_qoxinst[
617   subject -> ?src_subj
618 ], dep_all(?src_subj, ?tgt_subj).
619 has_dependency_to(?GS, ?GT) :- ?GS:dgr_subj, ?GT:dgr_qoxinst = g_subj(?GS, ?_
      ).
620
621 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
622 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
623
624 //% degradations of BIDepMod(Char)
625 dgr_char::dgr.
626
627 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
628 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
629 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
630 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
631 //% general code for direct/indirect rating of degradations from business
      degradations ?dgr
632 //% for which the predicate direct_rating(?dgr, ?rating) provides values;
633 //% direct_rating in this respect specifies the direct rating model;
634
635
636 //% copy direct rating as meta information into degradation
637 ?GT[
638   direct_rating -> ?rating,
639   rating -> ?rating
640 ] :- ?GT:dgr, direct_rating(?GT, ?rating).
641
642 //
643
644 //% derive indirect ratings (as meta information) from target degradations
645
646 //% here ?rating is considered only to be a single (arithemetic) value only
647 //% (only one point in time considered)
648
649 /*?GS[
650   indirect_rating -> ?rating,
651   rating -> ?rating
652 ] :- ?GS:dgr, ?rating =
653  sum{ ?rating_part | ?GT:dgr[ source(?rat_weigth) -> ?GS, rating -> ?
       tgt_rating ],
654         not is_subsumed_by(?GT, ?_), ?rating_part is ?rat_weigth * ?
              tgt_rating }.*/
655
656 //% also make visible how much each individual target contributed to the
      indirect rating
657 /*?GS[
658   indirect_rating_part(?GT, ?rat_weigth, ?tgt_rating) -> ?rating_part
659 ] :- ?GS:dgr, ?GT:dgr[ source(?rat_weigth) -> ?GS, rating -> ?tgt_rating ],
660 not is_subsumed_by(?GT, ?_), ?rating_part is ?rat_weigth * ?tgt_rating.*/
661
662 //
663
664 //% here ?rating is considered only to be a function (of time) term
665
666 ?GS[
667   indirect_rating -> ?rating,
668   rating -> ?rating
669 ] :- ?GS:dgr, not ?GS[ direct_rating -> ?_ ],
670  mysum(?rating, collectset{ ?rating_part | ?GT:dgr[ source(?rat_weigth) -> ?
       GS, rating -> ?tgt_rating ],
671         not is_subsumed_by(?GT, ?_), mymult(?rat_weigth, ?tgt_rating, ?
              rating_part), not ?rating_part = 0  }).
672
673 // simple arthmetic of function (of time) terms:
674 mysum(0, []).
675 mysum(?elem, [?elem]).
676 mysum(sum([ ?elem1 | [ ?elem2 | ?L ] ]), [ ?elem1 | [ ?elem2 | ?L ] ]).
```

```
677
678 mymult(0, ?_, 0).
679 mymult(?_, 0, 0).
680 mymult(?fact, ?elem, ?elem) :- ?fact = 1.0, not ?elem = 0.
681 mymult(?fact, ?fcty, ?fcty2) :- not ?fact = 0, not ?fact = 1.0, not ?fcty =
        0, ?fcty2 = ?fact(?fcty).
682
683 //% also make visible how much each individual target contributed to the
        indirect rating
684 ?GS[
685   indirect_rating_part(?GT, ?rat_weigth, ?tgt_rating) -> ?rating_part
686 ] :- ?GS:dgr, not ?GS[ direct_rating -> ?_ ],
687 ?GT:dgr[ source(?rat_weigth) -> ?GS, rating -> ?tgt_rating ],
688 not is_subsumed_by(?GT, ?_), mymult(?rat_weigth, ?tgt_rating, ?rating_part).
689
690 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
691 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
692 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
693 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
694
695
696 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
697 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
698 //%
699 //% example implementation in flora2 DDB system
700 //% for degradation dependencies using the impact dependency models
701 //% SIDepMod(Subj), SIDepMod(QoXInst), BIDepMod(Char)
702 //%
703 //% for some degradation subjects and related degradation dependencies
704 //% of the example scenario
705 //%
706 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
707 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
708 //%
709 //% file "degradation_example.flr"
710 //%
711 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
712 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
713
714 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
715 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
716 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
717 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
718 //% III.2.a) definitions for degradation dependencies of scenario
719 //% detailed degradation dependencies using SIDepMod(Subj/Fcty), SIDepMod(QoX
        ), SIDepMod(QoXInst)
720
721 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
722 //% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
723 //% examples for degradation dependencies(QoXInst) of scenario
724
725 //% degradation dependency g_r1 -> g_s1-1
726 g_mailsend_delay(?G):dgr_qoxinst[
727  subject -> f_mail_use_send,
728  manner -> gt_high_mailsend_delay,
729  avg_mailsend_delay_rise_minutes -> ?rise,
730
731  //% meta information: entailing source degradation(s) with weighting rating
732  source(1.0) -> ?G
733 ] :- ?G:dgr_qoxinst[
734  subject -> r_ip_link1,
735  manner -> gt_high_link_utilization,
736  avg_val_percent -> ?pcnt
737 ], ?pcnt > 40, ?rise is ((?pcnt - 40) / 10 * 2.5/2).
738
739 //
740
741 //% degradation dependency g_r1b -> g_s1-1
742 g_mailsend_delay(?G):dgr_qoxinst[
743  subject -> f_mail_use_send,
744  manner -> gt_high_mailsend_delay,
```

```
745  avg_mailsend_delay_rise_minutes -> ?rise,
746
747  //% meta information: entailing source degradation(s) with weighting rating
748  source(1.0) -> ?G
749 ] :- ?G:dgr_qoxinst[
750  subject -> r_dnssv1,
751  manner -> gt_high_dns_delay,
752  avg_request_delay_sec -> ?dns_delay
753  ], ?dns_delay > 1, ?rise is ?dns_delay * 2 / 60.
754
755 has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_qoxinst =
        g_mailsend_delay(?GS).
756
757 //% example of degradation dependency as combination of two degradations:
758 g_mailsend_delay_add(?Set):dgr_qoxinst[
759  subject -> ?subject,
760  manner -> ?manner,
761  avg_mailsend_delay_rise_minutes -> ?rise,
762
763  //% meta information: entailing source degradation(s) with weighting rating
764  source(?rat1) -> ?GS1,
765  source(?rat2) -> GS2
766 ] :- ?GS1:dgr_qoxinst, ?GS2:dgr_qoxinst, ?GS1 \= ?GS2,
767 ?GT1 = g_mailsend_delay(?GS1)[subject -> ?subject, manner -> ?manner,
        avg_mailsend_delay_rise_minutes -> ?rise1 ],
768 ?GT2 = g_mailsend_delay(?GS2)[subject -> ?subject, manner -> ?manner,
        avg_mailsend_delay_rise_minutes -> ?rise2 ],
769 ?rise is ?rise1 + ?rise2,
770 ?rat1 is ?rise1 / ?rise, ?rat2 is ?rise2 / ?rise,
771 list_to_ord_set([?GT1, ?GT2], ?Set)@_prolog(ordsets).
772 has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_qoxinst =
        g_mailsend_delay_add(?L), ?L:_list, member(?GS, ?L)@_prolog(basics).
773 is_subsumed_by(?GC, ?GP) :- ?GC:dgr_qoxinst, ?GP:dgr_qoxinst =
        g_mailsend_delay_add(?L), ?L:_list, member(?GC, ?L)@_prolog(basics).
774
775 //% degradation dependencies from g_r2
776 g_mboxaccess(?GS):dgr_qoxinst[
777  subject -> _#:f_mail_use_mboxaccess[ user -> ?grpMail ],
778  manner -> gt_mbox_unavailability,
779  unavailability -> total,
780
781   //% meta information: entailing source degradation(s) with weighting rating
782  source(1.0) -> ?GS
783 ] :- ?GS:dgr_qoxinst[
784 subject -> ?_:r_afs_sv1[ path -> ?pathListAfs ],
785  manner -> gt_afs_cell_outage,
786  unavailability -> total
787 ], user_group_afs_to_mbox(?pathListAfs, ?grpMail).
788 has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_qoxinst =
        g_mboxaccess(?GS).
789 // here only one example:
790 user_group_afs_to_mbox(PathListAfs1, GrpMail).
791
792 g_webpageaccess(?GS):dgr_qoxinst[
793  subject -> _#:f_web_use_accesspage[ user -> ?grpWeb ],
794  manner -> gt_webpage_access_unavailability,
795  unavailability -> total,
796
797   //% meta information: entailing source degradation(s) with weighting rating
798  source(1.0) -> ?GS
799 ] :- ?GS:dgr_qoxinst[
800 subject -> ?_:r_afs_sv1[ path -> ?pathListAfs ],
801  manner -> gt_afs_cell_outage,
802  unavailability -> total
803 ], user_group_afs_to_webpage(?pathListAfs, ?grpWeb).
804 has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_qoxinst =
        g_webpageaccess(?GS).
805 // here only one example:
806 user_group_afs_to_webpage(PathListAfs1, GrpWeb).
807
808
```

```
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% examples business degradation dependency (SIDepMod(QoXInst) -> BIDepMod(
    Char)) of scenario

g_slp_mail3(?GS):dgr_char[
  business_degradation_characteristic -> slp_mail3,

 //% meta information: entailing source degradation(s) with weighting rating
 source(1.0) -> ?GS
] :- ?GS:dgr_qoxinst[
  subject -> f_mail_use_send,
  manner -> gt_high_mailsend_delay,
  avg_mailsend_delay_rise_minutes -> ?rise
], not is_subsumed(?GS), ?rise > 2.
has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_char = g_slp_mail3(?
    GS).

//

//% for testing of multiple degradations entailed by a source degradation:

g_slp_test(?GS):dgr_char[
  business_degradation_characteristic -> slp_test,

 //% meta information: entailing source degradation(s) with weighting rating
 source(0.5) -> ?GS
] :- ?GS:dgr_qoxinst[
  subject -> r_ip_link1,
  manner -> gt_high_link_utilization,
  avg_val_percent -> ?pcnt
], ?pcnt > 10.
has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_char = g_slp_test(?GS
    ).


//%
g_slp_mail1(?GS):dgr_char[
  business_degradation_characteristic -> slp_mail1,
  ratio -> ?ratio,

 //% meta information: entailing source degradation(s) with weighting rating
 source(1.0) -> ?GS
] :- ?GS:dgr_qoxinst[
 subject -> ?_:f_mail_use_mboxaccess[ user -> ?UserGroup ],
 manner -> gt_mbox_unavailability,
 unavailability -> total
], ratio_mail1(?UserGroup, ?ratio).
has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_char = g_slp_mail1(?
    GS).
// only exemplified here:
ratio_mail1(GrpMail, ?ratio) :- ?ratio is 1/3 + 1/10.

g_slp_prem_web_1_and_2(?GS):dgr_char[
  business_degradation_characteristic -> slp_prem_web_1_and_2,
  ratio -> ?ratio,

 //% meta information: entailing source degradation(s) with weighting rating
 source(1.0) -> ?GS
] :- ?GS:dgr_qoxinst[
 subject -> ?_:f_web_use_accesspage[ user -> ?UserGroup ],
 manner -> gt_webpage_access_unavailability,
 unavailability -> total
], ratio_prem_web_1_and_2(?UserGroup, ?ratio).
has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_char =
    g_slp_prem_web_1_and_2(?GS).
// only exemplified here:
ratio_prem_web_1_and_2(GrpWeb, ?ratio) :-?ratio is 10/300 * 2.

g_slp_norm_web_1_and_2(?GS):dgr_char[
  business_degradation_characteristic -> slp_norm_web_1_and_2,
```

```
  ratio -> ?ratio ,

 //% meta information: entailing source degradation(s) with weighting rating
 source(1.0) -> ?GS
] :- ?GS:dgr_qoxinst[
 subject -> ?_:f_web_use_accesspage[ user -> ?UserGroup ],
 manner -> gt_webpage_access_unavailability ,
 unavailability -> total
], ratio_norm_web_1_and_2(?UserGroup, ?ratio).
has_dependency_to(?GS, ?GT) :- ?GS:dgr_qoxinst, ?GT:dgr_char =
    g_slp_norm_web_1_and_2(?GS).
// only exemplified here:
ratio_norm_web_1_and_2(GrpWeb, ?ratio) :-?ratio is 5.

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% direct rating model example

//% only taking into account one instance of time here

// direct_rating(?GT, ?rating) :- ?GT = g_slp_mail3(?_):dgr_char, ?rating is
    5.

// direct_rating(?GT, ?rating) :- ?GT = g_slp_test(?_):dgr_char, ?rating is 2.

//

// direct_rating(?GT, ?rating) :- ?GT = g_slp_mail1(?_):dgr_char[ ratio -> ?
    ratio ], ?rating is 1 * ?ratio.

// direct_rating(?GT, ?rating) :- ?GT = g_slp_prem_web_1_and_2(?_):dgr_char[
    ratio -> ?ratio ], ?rating is 10 * ?ratio.

// direct_rating(?GT, ?rating) :- ?GT = g_slp_norm_web_1_and_2(?_):dgr_char[
    ratio -> ?ratio ], ?rating is 2 * ?ratio.

//

//% using function (of time) terms based on the
    business_degradation_characteristics (and ratio)

direct_rating(?GT, ?rating) :- ?GT = g_slp_mail3(?_):dgr_char[
    business_degradation_characteristic -> ?bdchar ], ?rating = ?bdchar.

//

direct_rating(?GT, ?rating) :- ?GT = g_slp_mail1(?_):dgr_char[
    business_degradation_characteristic -> ?bdchar, ratio -> ?ratio ], ?
    rating = ?ratio(?bdchar).

direct_rating(?GT, ?rating) :- ?GT = g_slp_prem_web_1_and_2(?_):dgr_char[
    business_degradation_characteristic -> ?bdchar, ratio -> ?ratio ], ?
    rating = ?ratio(?bdchar).

direct_rating(?GT, ?rating) :- ?GT = g_slp_norm_web_1_and_2(?_):dgr_char[
    business_degradation_characteristic -> ?bdchar, ratio -> ?ratio ], ?
    rating = ?ratio(?bdchar).


//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% III.2.b) initial resource degradations of scenario

//% example of a degradation(Subj)
// g_mailsec:dgr_subj[
//   subject -> r_mailsec
// ].
// initial_resource_degradation(g_mailsec).

```

```
//?- dep_all(g_mailsec, ?GT:dgr, ?HL),?GT[%pp_self]@pp.

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% examples for degradation(QoXInst)

//% degradation g_r1 of scenario
g_r1:dgr_qoxinst[
 subject -> r_ip_link(r_rt_lrz, r_sw2),
 manner -> gt_high_link_utilization,
 avg_val_percent -> 60
].

//% degradation g_r1b of scenario
g_r1b:dgr_qoxinst[
 subject -> r_dnssv1,
 manner -> gt_high_dns_delay,
 avg_request_delay_sec -> 15
].

//

//% degradation g_r2 of scenario
g_r2:dgr_qoxinst[
subject -> _#:r_afs_sv1[ path -> PathListAfs1 ],
 manner -> gt_afs_cell_outage,
 unavailability -> total
].

initial_resource_degradation(g_r1).
initial_resource_degradation(g_r1b).
initial_resource_degradation(g_r2).

//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% III.2.c) example queries for degradation dependencies of scenario

//?- ?G:dgr.

//?- ?G:dgr_qoxinst, ?G[%pp_self]@pp.

//?- has_dependency_to(?GS:dgr, ?GT:dgr), ?GS[%pp_self]@pp,?GT[%pp_self]@pp.

//?- dep_all(?GT:dgr, ?GS:dgr), ?GS[%pp_self]@pp, ?GT[%pp_self]@pp.

//?- dep_all(?GS:dgr, ?GT:dgr, ?L), ?GS[%pp_self]@pp, ?GT[%pp_self]@pp.

//?- dep_all(g_r1, ?GT:dgr, ?L), ?GT[%pp_self]@pp.
//?- dep_all(g_r1b, ?GT:dgr, ?L), ?GT[%pp_self]@pp.

//?- dep_all_nosubsumption(g_mailsec, ?GT:dgr), ?GT[%pp_self]@pp.

//?- g_r1[%pp_self]@pp.

//?- g_r1b[%pp_self]@pp.

//?- ?GT:dgr[ source(?rat_weighting) -> g_r1 ], ?GT[%pp_self]@pp.

//?- ?GI = g_r2, dep_all_nosubsumption(?GI, ?GT:dgr), ?GT[%pp_self]@pp, ?GI[%
    pp_self]@pp.

//?- ?GI = g_r2, ?GI[ rating -> ?rating ].

//% find all degradations entailed from the initial resource degradations
//% and print out them
?- initial_resource_degradation(?GI), dep_all_nosubsumption(?GI, ?GT:dgr), ?
    GT[%pp_self]@pp, ?GI[%pp_self]@pp.

//% print the rating of all initial resource degradations
```

```
?- initial_resource_degradation(?GI), ?GI[ rating -> ?rating ].


//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//% <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Code Listing 6.2: Output of example Flora2 session

```
:~/flora2> ./runflora -e ['ira'].

[xsb_configuration loaded]
[sysinitrc loaded]

XSB Version 3.0.1 (Sagres) of August 7, 2006
[i686-pc-linux-gnu; mode: optimal; engine: slg-wam; gc: indirection;
    scheduling: local]


Evaluating command line goal:
| ?-  asserta(library_directory('/home/soft/XSB/packages/flora2')). [flora2].
     flora_shell. ['ira'].

| ?-
yes
| ?- [flora2 loaded]
[floraInstallMode loaded]

yes
| ?- [flrversion loaded]
[p2h_config loaded]
[prolog2hilog loaded]
[flrshell loaded]
[flrundefhook loaded]
[flrutils loaded]
[flranswer loaded]
[flrload loaded]
[flrdisplay loaded]
[flrimportedcalls loaded]
[trailer$eq=none+inh=flogic+cus=none_main loaded]
[patch_main loaded]

FLORA-2 Version 0.95 (Androcymbium) of September 2007

Type '_help.' to display help.
Type '_demo(demoName).' to run a demo.

flora2 ?-
[FLORA: Loading /local/home/a2824al/soft/XSB/packages/flora2/ira.flr into
    module main]
[ira_main loaded]
[flraggcolset loaded]
[flrclause loaded]
[FLORA: Dynamically loading /local/home/a2824al/soft/XSB/packages/flora2/ira.
    fdb into module main]
[Preprocessing /local/home/a2824al/soft/XSB/packages/flora2/ira.fdb]
[FLORA: Done! CPU time used: 0.0080 seconds]
[FLORA: Dynamically loading /local/home/a2824al/soft/XSB/packages/flora2/ira.
    fld into module main]
[Preprocessing /local/home/a2824al/soft/XSB/packages/flora2/ira.fld]
[FLORA: Done! CPU time used: 0.0040 seconds]

[FLORA: Loading /local/home/a2824al/soft/XSB/packages/flora2/pkgs/prettyprint
    .flr into module pp]
[prettyprint_pp loaded]
[flrexport loaded]
[flrcontrol loaded]
[flrmetaops loaded]
[flrdynmod loaded]
```

```
55  [FLORA: loading FLORA system module '_io']
56  [flrio__io loaded]
57  [FLORA: loading FLORA system module '_system']
58  [flrsystem__system loaded]
59  [FLORA: Dynamically loading /local/home/a2824al/soft/XSB/packages/flora2/lib/
        flrsystem.fld into module _system]
60  [Preprocessing /local/home/a2824al/soft/XSB/packages/flora2/lib/flrsystem.fld
        ]
61  [FLORA: Done! CPU time used: 0.0000 seconds]
62  [FLORA: Dynamically loading /local/home/a2824al/soft/XSB/packages/flora2/lib/
        flrio.fld into module _io]
63  [Preprocessing /local/home/a2824al/soft/XSB/packages/flora2/lib/flrio.fld]
64  [FLORA: Done! CPU time used: 0.0000 seconds]
65  [FLORA: Dynamically loading /local/home/a2824al/soft/XSB/packages/flora2/pkgs
        /prettyprint.fld into module pp]
66  [Preprocessing /local/home/a2824al/soft/XSB/packages/flora2/pkgs/prettyprint.
        fld]
67  [FLORA: Done! CPU time used: 0.0000 seconds]
68
69  Yes
70
71
72  ?src = f_afs
73  ?tgt = _#'1'2
74
75  ?src = f_afs_use
76  ?tgt = _#'1'2
77
78  ?src = f_afs_use_auth
79  ?tgt = _#'1'2
80
81  ?src = f_dns
82  ?tgt = _#'1'2
83
84  ?src = f_dns_use
85  ?tgt = _#'1'2
86
87  ?src = f_ip
88  ?tgt = _#'1'2
89
90  ?src = f_ip_use
91  ?tgt = _#'1'2
92
93  ?src = f_ip_use_connectivity
94  ?tgt = _#'1'2
95
96  ?src = r_mailsec
97  ?tgt = _#'1'2
98
99  ?src = r_rtcore
100 ?tgt = _#'1'2
101
102 ?src = the_any_instance(r_mailsec)
103 ?tgt = _#'1'2
104
105 ?src = the_any_instance(r_rtcore)
106 ?tgt = _#'1'2
107
108 12 solution(s) in 0.8720 seconds on lxdsz02
109
110 Yes
111
112 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)])) : 'dgr_char'.
113 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)])) : '_object'.
114 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)])) : '_object'.
115 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)]))[
116     business_degradation_characteristic -> {'slp_mail3'},
```

```
117     direct_rating -> {'slp_mail3'},
118     rating -> {'slp_mail3'},
119     source -> {g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay
            (g_r1b)])},
120     source_recursive -> {'g_r1','g_r1b',g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])},
121     source(1.0000) -> {g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)])}
122 ].
123
124 'g_r1'  :  'dgr_qoxinst'.
125 'g_r1'  :  '_symbol'.
126 'g_r1'  :  '_object'.
127 'g_r1'[
128     avg_val_percent -> {60},
129     indirect_rating -> {0.8333(slp_mail3)},
130     manner -> {'gt_high_link_utilization'},
131     rating -> {0.8333(slp_mail3)},
132     subject -> {'r_ip_link1',r_ip_link(r_rt_lrz,r_sw2)},
133     target -> {g_mailsend_delay(g_r1),g_slp_test(g_r1)},
134     target_recursive -> {g_mailsend_delay(g_r1),g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]),g_slp_mail3(
            g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b
            )])),g_slp_test(g_r1)},
135     indirect_rating_part(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]),0.8333,slp_mail3) -> {0.8333(slp_mail3)},
136     indirect_rating_part(g_slp_test(g_r1),0.5000,0) -> {0}
137 ].
138
139 g_slp_test(g_r1)        :  'dgr_char'.
140 g_slp_test(g_r1)        :  '_object'.
141 g_slp_test(g_r1)        :  '_object'.
142 g_slp_test(g_r1)[
143     business_degradation_characteristic -> {'slp_test'},
144     indirect_rating -> {0},
145     rating -> {0},
146     source -> {'g_r1'},
147     source_recursive -> {'g_r1'},
148     source(0.5000) -> {'g_r1'}
149 ].
150
151 'g_r1'  :  'dgr_qoxinst'.
152 'g_r1'  :  '_symbol'.
153 'g_r1'  :  '_object'.
154 'g_r1'[
155     avg_val_percent -> {60},
156     indirect_rating -> {0.8333(slp_mail3)},
157     manner -> {'gt_high_link_utilization'},
158     rating -> {0.8333(slp_mail3)},
159     subject -> {'r_ip_link1',r_ip_link(r_rt_lrz,r_sw2)},
160     target -> {g_mailsend_delay(g_r1),g_slp_test(g_r1)},
161     target_recursive -> {g_mailsend_delay(g_r1),g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]),g_slp_mail3(
            g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b
            )])),g_slp_test(g_r1)},
162     indirect_rating_part(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]),0.8333,slp_mail3) -> {0.8333(slp_mail3)},
163     indirect_rating_part(g_slp_test(g_r1),0.5000,0) -> {0}
164 ].
165
166 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) :  '
        dgr_qoxinst'.
167 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) :  '
        _object'.
168 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) :  '
        _object'.
169 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])[
170     avg_mailsend_delay_rise_minutes -> {3.0000},
171     indirect_rating -> {'slp_mail3'},
172     manner -> {'gt_high_mailsend_delay'},
173     rating -> {'slp_mail3'},
```

```
174     source -> {g_mailsend_delay(g_r1),g_mailsend_delay(g_r1b)},
175     source_recursive -> {'g_r1','g_r1b'},
176     subject -> {'f_mail_use_send'},
177     target -> {g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]))},
178     target_recursive -> {g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(
            g_r1), g_mailsend_delay(g_r1b)]))},
179     source(0.1667) -> {'GS2','g_r1b'},
180     source(0.8333) -> {'GS2','g_r1'},
181     indirect_rating_part(g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(
            g_r1), g_mailsend_delay(g_r1b)])),1.0000,slp_mail3) -> {'slp_mail3'}
182 ].
183
184 'g_r1'  :  'dgr_qoxinst'.
185 'g_r1'  :  '_symbol'.
186 'g_r1'  :  '_object'.
187 'g_r1'[
188     avg_val_percent -> {60},
189     indirect_rating -> {0.8333(slp_mail3)},
190     manner -> {'gt_high_link_utilization'},
191     rating -> {0.8333(slp_mail3)},
192     subject -> {'r_ip_link1',r_ip_link(r_rt_lrz,r_sw2)},
193     target -> {g_mailsend_delay(g_r1),g_slp_test(g_r1)},
194     target_recursive -> {g_mailsend_delay(g_r1),g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]),g_slp_mail3(
            g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b
            )])),g_slp_test(g_r1)},
195     indirect_rating_part(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]),0.8333,slp_mail3) -> {0.8333(slp_mail3)},
196     indirect_rating_part(g_slp_test(g_r1),0.5000,0) -> {0}
197 ].
198
199 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)]))     :  'dgr_char'.
200 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)]))     :  '_object'.
201 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)]))     :  '_object'.
202 g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(
        g_r1b)]))[
203     business_degradation_characteristic -> {'slp_mail3'},
204     direct_rating -> {'slp_mail3'},
205     rating -> {'slp_mail3'},
206     source -> {g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay
            (g_r1b)])},
207     source_recursive -> {'g_r1','g_r1b',g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])},
208     source(1.0000) -> {g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)])}
209 ].
210
211 'g_r1b' :  'dgr_qoxinst'.
212 'g_r1b' :  '_symbol'.
213 'g_r1b' :  '_object'.
214 'g_r1b'[
215     avg_request_delay_sec -> {15},
216     indirect_rating -> {0.1667(slp_mail3)},
217     manner -> {'gt_high_dns_delay'},
218     rating -> {0.1667(slp_mail3)},
219     subject -> {'r_dnssv1'},
220     target -> {g_mailsend_delay(g_r1b)},
221     target_recursive -> {g_mailsend_delay(g_r1b),g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]),g_slp_mail3(
            g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b
            )]))},
222     indirect_rating_part(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]),0.1667,slp_mail3) -> {0.1667(slp_mail3)}
223 ].
224
225 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) :  '
        dgr_qoxinst'.
```

```
226 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) : '
        _object'.
227 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]) : '
        _object'.
228 g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])[
229     avg_mailsend_delay_rise_minutes -> {3.0000},
230     indirect_rating -> {'slp_mail3'},
231     manner -> {'gt_high_mailsend_delay'},
232     rating -> {'slp_mail3'},
233     source -> {g_mailsend_delay(g_r1),g_mailsend_delay(g_r1b)},
234     source_recursive -> {'g_r1','g_r1b'},
235     subject -> {'f_mail_use_send'},
236     target -> {g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]))},
237     target_recursive -> {g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(
            g_r1), g_mailsend_delay(g_r1b)]))},
238     source(0.1667) -> {'GS2','g_r1b'},
239     source(0.8333) -> {'GS2','g_r1'},
240     indirect_rating_part(g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(
            g_r1), g_mailsend_delay(g_r1b)])),1.0000,slp_mail3) -> {'slp_mail3'}
241 ].
242
243 'g_r1b' :  'dgr_qoxinst'.
244 'g_r1b' :  '_symbol'.
245 'g_r1b' :  '_object'.
246 'g_r1b'[
247     avg_request_delay_sec -> {15},
248     indirect_rating -> {0.1667(slp_mail3)},
249     manner -> {'gt_high_dns_delay'},
250     rating -> {0.1667(slp_mail3)},
251     subject -> {'r_dnssv1'},
252     target -> {g_mailsend_delay(g_r1b)},
253     target_recursive -> {g_mailsend_delay(g_r1b),g_mailsend_delay_add([
            g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)]),g_slp_mail3(
            g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b
            )]))},
254     indirect_rating_part(g_mailsend_delay_add([g_mailsend_delay(g_r1),
            g_mailsend_delay(g_r1b)]),0.1667,slp_mail3) -> {0.1667(slp_mail3)}
255 ].
256
257 g_mboxaccess(g_r2)      :  'dgr_qoxinst'.
258 g_mboxaccess(g_r2)      :  '_object'.
259 g_mboxaccess(g_r2)      :  '_object'.
260 g_mboxaccess(g_r2)[
261     indirect_rating -> {0.4333(slp_mail1)},
262     manner -> {'gt_mbox_unavailability'},
263     rating -> {0.4333(slp_mail1)},
264     source -> {'g_r2'},
265     source_recursive -> {'g_r2'},
266     subject -> {'_#'2'_h0'},
267     target -> {g_slp_mail1(g_mboxaccess(g_r2))},
268     target_recursive -> {g_slp_mail1(g_mboxaccess(g_r2))},
269     unavailability -> {'total'},
270     source(1.0000) -> {'g_r2'},
271     indirect_rating_part(g_slp_mail1(g_mboxaccess(g_r2)),1.0000,0.4333(
            slp_mail1)) -> {0.4333(slp_mail1)}
272 ].
273
274 'g_r2'  :  'dgr_qoxinst'.
275 'g_r2'  :  '_symbol'.
276 'g_r2'  :  '_object'.
277 'g_r2'[
278     indirect_rating -> {sum([0.4333(slp_mail1), sum([0.0667(
            slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])])},
279     manner -> {'gt_afs_cell_outage'},
280     rating -> {sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
            slp_norm_web_1_and_2)])])},
281     subject -> {'_#'4'2'},
282     target -> {g_mboxaccess(g_r2),g_webpageaccess(g_r2)},
283     target_recursive -> {g_mboxaccess(g_r2),g_slp_mail1(g_mboxaccess(g_r2)),
```

```
        g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),g_slp_prem_web_1_and_2
        (g_webpageaccess(g_r2)),g_webpageaccess(g_r2)},
    unavailability -> {'total'},
    indirect_rating_part(g_mboxaccess(g_r2),1.0000,0.4333(slp_mail1)) ->
        {0.4333(slp_mail1)},
    indirect_rating_part(g_webpageaccess(g_r2),1.0000,sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])) -> {sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])}
].

g_webpageaccess(g_r2)   :  'dgr_qoxinst'.
g_webpageaccess(g_r2)   :  '_object'.
g_webpageaccess(g_r2)   :  '_object'.
g_webpageaccess(g_r2)[
    indirect_rating -> {sum([0.0667(slp_prem_web_1_and_2), 5(
        slp_norm_web_1_and_2)])},
    manner -> {'gt_webpage_access_unavailability'},
    rating -> {sum([0.0667(slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])},
    source -> {'g_r2'},
    source_recursive -> {'g_r2'},
    subject -> {'_#'3'_h0'},
    target -> {g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),
        g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))},
    target_recursive -> {g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),
        g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))},
    unavailability -> {'total'},
    source(1.0000) -> {'g_r2'},
    indirect_rating_part(g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))
        ,1.0000,5(slp_norm_web_1_and_2)) -> {5(slp_norm_web_1_and_2)},
    indirect_rating_part(g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))
        ,1.0000,0.0667(slp_prem_web_1_and_2)) -> {0.0667(
        slp_prem_web_1_and_2)}
].

'g_r2'  :  'dgr_qoxinst'.
'g_r2'  :  '_symbol'.
'g_r2'  :  '_object'.
'g_r2'[
    indirect_rating -> {sum([0.4333(slp_mail1), sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])])},
    manner -> {'gt_afs_cell_outage'},
    rating -> {sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
        slp_norm_web_1_and_2)])])},
    subject -> {'_#'4'2'},
    target -> {g_mboxaccess(g_r2),g_webpageaccess(g_r2)},
    target_recursive -> {g_mboxaccess(g_r2),g_slp_mail1(g_mboxaccess(g_r2)),
        g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),g_slp_prem_web_1_and_2
        (g_webpageaccess(g_r2)),g_webpageaccess(g_r2)},
    unavailability -> {'total'},
    indirect_rating_part(g_mboxaccess(g_r2),1.0000,0.4333(slp_mail1)) ->
        {0.4333(slp_mail1)},
    indirect_rating_part(g_webpageaccess(g_r2),1.0000,sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])) -> {sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])}
].

g_slp_mail1(g_mboxaccess(g_r2)) :  'dgr_char'.
g_slp_mail1(g_mboxaccess(g_r2)) :  '_object'.
g_slp_mail1(g_mboxaccess(g_r2)) :  '_object'.
g_slp_mail1(g_mboxaccess(g_r2))[
    business_degradation_characteristic -> {'slp_mail1'},
    direct_rating -> {0.4333(slp_mail1)},
    rating -> {0.4333(slp_mail1)},
    ratio -> {0.4333},
    source -> {g_mboxaccess(g_r2)},
    source_recursive -> {'g_r2',g_mboxaccess(g_r2)},
    source(1.0000) -> {g_mboxaccess(g_r2)}
].

'g_r2'  :  'dgr_qoxinst'.
'g_r2'  :  '_symbol'.
```

```
'g_r2'  :  '_object'.
'g_r2'[
    indirect_rating -> {sum([0.4333(slp_mail1), sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])])},
    manner -> {'gt_afs_cell_outage'},
    rating -> {sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
        slp_norm_web_1_and_2)])])},
    subject -> {'_#'4'2'},
    target -> {g_mboxaccess(g_r2),g_webpageaccess(g_r2)},
    target_recursive -> {g_mboxaccess(g_r2),g_slp_mail1(g_mboxaccess(g_r2)),
        g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),g_slp_prem_web_1_and_2
        (g_webpageaccess(g_r2)),g_webpageaccess(g_r2)},
    unavailability -> {'total'},
    indirect_rating_part(g_mboxaccess(g_r2),1.0000,0.4333(slp_mail1)) ->
        {0.4333(slp_mail1)},
    indirect_rating_part(g_webpageaccess(g_r2),1.0000,sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])) -> {sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])}
].

g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))  :  'dgr_char'.
g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))  :  '_object'.
g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))  :  '_object'.
g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))[
    business_degradation_characteristic -> {'slp_norm_web_1_and_2'},
    direct_rating -> {5(slp_norm_web_1_and_2)},
    rating -> {5(slp_norm_web_1_and_2)},
    ratio -> {5},
    source -> {g_webpageaccess(g_r2)},
    source_recursive -> {'g_r2',g_webpageaccess(g_r2)},
    source(1.0000) -> {g_webpageaccess(g_r2)}
].

'g_r2'  :  'dgr_qoxinst'.
'g_r2'  :  '_symbol'.
'g_r2'  :  '_object'.
'g_r2'[
    indirect_rating -> {sum([0.4333(slp_mail1), sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])])},
    manner -> {'gt_afs_cell_outage'},
    rating -> {sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
        slp_norm_web_1_and_2)])])},
    subject -> {'_#'4'2'},
    target -> {g_mboxaccess(g_r2),g_webpageaccess(g_r2)},
    target_recursive -> {g_mboxaccess(g_r2),g_slp_mail1(g_mboxaccess(g_r2)),
        g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),g_slp_prem_web_1_and_2
        (g_webpageaccess(g_r2)),g_webpageaccess(g_r2)},
    unavailability -> {'total'},
    indirect_rating_part(g_mboxaccess(g_r2),1.0000,0.4333(slp_mail1)) ->
        {0.4333(slp_mail1)},
    indirect_rating_part(g_webpageaccess(g_r2),1.0000,sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])) -> {sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])}
].

g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))  :  'dgr_char'.
g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))  :  '_object'.
g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))  :  '_object'.
g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))[
    business_degradation_characteristic -> {'slp_prem_web_1_and_2'},
    direct_rating -> {0.0667(slp_prem_web_1_and_2)},
    rating -> {0.0667(slp_prem_web_1_and_2)},
    ratio -> {0.0667},
    source -> {g_webpageaccess(g_r2)},
    source_recursive -> {'g_r2',g_webpageaccess(g_r2)},
    source(1.0000) -> {g_webpageaccess(g_r2)}
].

'g_r2'  :  'dgr_qoxinst'.
'g_r2'  :  '_symbol'.
'g_r2'  :  '_object'.
```

```
'g_r2'[
    indirect_rating -> {sum([0.4333(slp_mail1), sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])])},
    manner -> {'gt_afs_cell_outage'},
    rating -> {sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
        slp_norm_web_1_and_2)])])},
    subject -> {'_#'4'2'},
    target -> {g_mboxaccess(g_r2),g_webpageaccess(g_r2)},
    target_recursive -> {g_mboxaccess(g_r2),g_slp_mail1(g_mboxaccess(g_r2)),
        g_slp_norm_web_1_and_2(g_webpageaccess(g_r2)),g_slp_prem_web_1_and_2
        (g_webpageaccess(g_r2)),g_webpageaccess(g_r2)},
    unavailability -> {'total'},
    indirect_rating_part(g_mboxaccess(g_r2),1.0000,0.4333(slp_mail1)) ->
        {0.4333(slp_mail1)},
    indirect_rating_part(g_webpageaccess(g_r2),1.0000,sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])) -> {sum([0.0667(
        slp_prem_web_1_and_2), 5(slp_norm_web_1_and_2)])}
].


?GI = g_r1
?GT = g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])

?GI = g_r1
?GT = g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1),
    g_mailsend_delay(g_r1b)]))

?GI = g_r1
?GT = g_slp_test(g_r1)

?GI = g_r1b
?GT = g_mailsend_delay_add([g_mailsend_delay(g_r1), g_mailsend_delay(g_r1b)])

?GI = g_r1b
?GT = g_slp_mail3(g_mailsend_delay_add([g_mailsend_delay(g_r1),
    g_mailsend_delay(g_r1b)]))

?GI = g_r2
?GT = g_mboxaccess(g_r2)

?GI = g_r2
?GT = g_slp_mail1(g_mboxaccess(g_r2))

?GI = g_r2
?GT = g_slp_norm_web_1_and_2(g_webpageaccess(g_r2))

?GI = g_r2
?GT = g_slp_prem_web_1_and_2(g_webpageaccess(g_r2))

?GI = g_r2
?GT = g_webpageaccess(g_r2)

10 solution(s) in 0.3920 seconds on lxdsz02

Yes


?GI = g_r1
?rating = 0.8333(slp_mail3)

?GI = g_r1b
?rating = 0.1667(slp_mail3)

?GI = g_r2
?rating = sum([0.4333(slp_mail1), sum([0.0667(slp_prem_web_1_and_2), 5(
    slp_norm_web_1_and_2)])])

3 solution(s) in 0.0040 seconds on lxdsz02

Yes

```

```
455
456 Yes
457
458
459 flora2 ?- _halt.
460
461 End XSB (cputime 1.56 secs, elapsetime 5.82 secs)
462
463 :~/flora2>
```

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LITERATURE

[AAG+04a] M. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem determination using dependency graphs and run-time behavior models. In *Proceedings of the 15th International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, pages 171–182, Davis, California, USA, November 2004. IFIP.

[AAG+04b] M. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem Determination Using Dependency Graphs and Run-Time Behavior Models. Technical Report RI04004, IBM Research Report, 2004.

[ADAD01] R.H. Arpaci-Dusseau and A.C. Arpaci-Dusseau. Fail-stutter fault tolerance. In *Eighth Workshop on Hot Topics in Operating Systems*, page 33, Washington, DC, USA, 2001.

[AGK+04] M. Agarwal, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Mining Activity Data for Dynamic Dependency Discovery in E-Business Systems. *eTransactions on Network and Service Management (eTNSM)*, September 2004.

[AMW+03] M. Aguilera, J. Mogul, J. Wiener, P. Reynolds, and A. Mutchitacharoen. Performance Debugging for Distributed Systems of Black Boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 329–344, Bolton Landing, New York, USA, October 2003.

[AP94] A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AICom - Artificial Intelligence Communications, IOS Press*, 7(1):39–59, 1994.

[Apr] Spectrorx. http://www.aprisma.com/products/.

[Apr04] Event correlation and root cause analysis in spectrum. http://www.aprisma.com/literature/white-papers/wp0536.pdf, March 2004.

[ARM98] Systems Management: Application Response Measurement (ARM). Technical report, OpenGroup, July 1998.

[BC03] A. Bansal and A. Clemm. Auto-Discovery at the Network and Service Management Layer. In *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, pages 365–378, Colorado Springs, Colorado, USA, March 2003.

[BCS99] P. Bhoj, S. Chutani, and S. Singhal. Sla management in federated environments. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM 1999)*, pages 293–308, Boston, Massassuchets, USA, May 1999. IFIP/IEEE.

[BGSS06] M. Brenner, M. Garschhammer, M. Sailer, and T. Schaaf. CMDB - Yet another MIB? On Reusing Management Model Concepts in ITIL Configuration Management. In *Proceedings of the 17th International IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, pages 269–280, Dublin, Ireland, October 2006.

[BHM+01] L. Burns, J. Hellerstein, S. Ma, C. Perng, D. Rabenhorst, and D. Taylor. Towards discovery of event correlation rules. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, pages 345–359, Seattle, Washington, USA, May 2001. IFIP/IEEE.

[BJ00] A. Bierman and K. Jones. Physical Topology MIB. Technical Report RFC 2922, IETF Network Working Group, 2000.

[BK96] A. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint International Conference and Symposium on Logic Programming*, pages 142–156, Bonn, Germany, 1996.

[BKH01] S. Bagchi, G. Kar, and J. Hellerstein. Dependency analysis in distributed systems using fault injections: Application to problem determination in an e-commerce environment. In *Proceedings of the 12th International IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2001)*, Nancy, France, October 2001.

[BKK01] A. Brown, G. Kar, and A. Keller. An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Application Environment. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, pages 377–390, Seattle, Washington, USA, May 2001.

[BS04] C. Bartolini and M. Salle. Business driven prioritization of service incidents. In *Proceedings of the 15th International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, pages 64–75, Davis, California, USA, November 2004. IFIP.

[Bus] Business continuity impact analysis. http://www.sorm.state.tx.us/-Risk_Management/Business_Continuity/bus_impact.php.

[Can03] G. Candea. The basics of dependability. http://www.pld.ttu.ee:81/-IAF0030/basics-1.pdf, September 2003.

[CFK+02] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In *Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP 2002)*, pages 153–183, Edinburgh, Scotland, July 2002.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin/New York, 1990.

[CIM] CIM Standards. http://www.dmtf.org/standards/cim.

[CJ05] T. Chatain and C. Jard. Models for the Supervision of Web Services Orchestration with Dynamic Changes. In *Proceedings of the IARIA International Conference on Service Assurance with Partial and Intermittent Resources (SAPIR 2005)*, Lisbon, Portugal, July 2005. IEEE press.

[CKW93] W. Chen, M. Kifer, and D.S. Warren. Hilog: A foundation for high-order logic programming. *Journal of Logic Programming*, 15(3):187–230, Feb 1993.

[CMRW96] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2). Technical Report RFC 1902, IETF, January 1996.

[COB] Common Objectives for Information and related Technology (COBIT 4.0). http://www.isaca.org/cobit.

[CR99] D. Caswell and S. Ramanathan. Using service models for management of internet services. In *HP Technical Report HPL-1999-43, HP Laboratories*, Palo Alto, California, USA, March 1999.

[DgFS07] V. Danciu, N. gentschen Felde, and M. Sailer. Declarative Specification of Service Management Attributes. In *Proceedings of the 10th IFIP/IEEE International Conference on Integrated Network Management (IM 2007)*, Munich, Germany, May 2007.

[DHHS06] V. Danciu, A. Hanemann, H.-G. Hegering, and M. Sailer. IT Service Management: Getting the View. In *Managing Development and Application of Digital Technologies*, pages 109–130, Munich, Germany, June 2006. CDTM, Springer Verlag.

[DR02] G. Dreo Rodosek. *A Framework for IT Service Management*. Habilitation, Ludwig–Maximilians–Universität München, June 2002.

[DR03] G. Dreo Rodosek. A Generic Model for IT Services and Service Management. In *Integrated Network Management VIII — Managing It All*, volume 2003, pages 171–184, Colorado Springs, USA, March 2003. IFIP/IEEE, Kluwer Academic Publishers.

[EK02] C. Ensel and A. Keller. An Approach for Managing Service Dependencies with XML and the Resource Description Framework. *Journal of Network and Systems Management*, 10(2), June 2002.

[Ens01a] C. Ensel. A Scalable Approach to Automated Service Dependency Modeling in Heterogeneous Environments. In *5th International Enterprise Distributed Object Computing Conference (EDOC ´01)*, Seattle, USA, September 2001. IEEE, IEEE Publishing.

[Ens01b] C. Ensel. New Approach for Automated Generation of Service Dependency Models. In *Network Management as a Strategy for Evolution and Development; Second Latin American Network Operation and Management Symposium*

(LANOMS 2001), Belo Horizonte, Brazil, August 2001. IEEE, IEEE Publishing.

[eTO] enhanced Telecom Operations Map. http://www.tmforum.org.

[FJP99] Svend Frolund, Mudita Jain, and Jim Pruyne. SoLOMon: Monitoring End-ser Service Levels. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99), Boston, MA*, May 1999.

[Gar04] M. Garschhammer. *Dienstgütebehandlung im Dienstlebenszyklus — von der formalen Spezifikation zur rechnergestützten Umsetzung*. Dissertation, Ludwig–Maximilians–Universität München, July 2004.

[GB 05] enhanced Telecom Operations Map (eTOM), The Business Process Framework For The Information and Communications Services Industry. Technical Report GB 921 Approved Version 5.0, TeleManagement Forum, April 2005.

[GHH+01] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, M. Langer, M. Nerb, I. Radisic, H. Roelle, and H. Schmidt. Towards generic Service Management Concepts — A Service Model Based Approach. In G. Pavlou, N. Anerousis, and A. Liotta, editors, *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, pages 719–732, Seattle, Washington, USA, May 2001. IFIP/IEEE, IEEE Publishing.

[GHH+02] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. A Case–Driven Methodology for Applying the MNM Service Model. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th International IFIP/IEEE Network Operations and Management Symposium (NOMS 2002)*, pages 697–710, Florence, Italy, April 2002. IFIP/IEEE, IEEE Publishing.

[GHK+01] M. Garschhammer, R. Hauck, B. Kempter, I. Radisic, H. Roelle, and H. Schmidt. The MNM Service Model — Refined Views on Generic Service Management. *Journal of Communications and Networks*, 3(4):297–306, December 2001.

[GNAK03] M. Gupta, A. Neogi, M. Agarwal, and G. Kar. Discovering dynamic dependencies in enterprise environments for problem determination. In *Proceedings of the 14th IFIP/IEEE Workshop on Distributed Sytems: Operations and Management*. IFIP/IEEE, October 2003.

[GRM97] Information Technology - Open Systems Interconnection - Structure of Management Information - Part 7: General Relationship Model, IS 10165-7, ISO and International Electrotechnical Committee, 1997.

[Gru98] B. Gruschke. A New Approach for Event Correlation based on Dependency Graphs. In *Proceedings of the 5th Workshop of the OpenView University Association: OVUA98́*, Rennes, France, April 1998.

[Gru99] B. Gruschke. *Entwurf eines Eventkorrelators mit Abhängigkeitsgraphen*. Dissertation, Ludwig–Maximilians–Universität München, November 1999.

[Guo04] J. Guo. Fault tolerant computing. http://www.engin.umd.umich.edu/vi/-w3_workshops/VI_winter04_guo.pdf, 2004.

[HAN99] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems — Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1–55860–571–1, January 1999. 651 p.

[Han07] A. Hanemann. *Automated IT Service Fault Diagnosis Based on Event Correlation Techniques*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[Has01] P. Hasselmeyer. Managing dynamic service dependencies. In *Proceedings of the 12th International IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2001)*, Nancy, France, October 2001. IFIP/IEEE.

[Hei04] E. Heidinger. Ein CIM–basiertes Modell der Rechnernetzepraktikum–Infrastruktur. Systementwicklungsprojekt, Technische Universität München, January 2004.

[HKS99] H. Hazewinkel, C. Kalbfleisch, and J. Schoenwaelder. Definition of Managed Objects for WWW Services. Technical Report RFC 2594, IETF, May 1999.

[HMP02] J. Hellerstein, S. Ma, and C. Perng. Discovering Actionable Patterns from Event Data. *IBM Systems Journal*, 41(3):475–493, 2002.

[HMSS06] A. Hanemann, P. Marcu, M. Sailer, and D. Schmitz. Specification of Dependencies for IT Service Fault Management. In *Proceedings of the 1st International Conference on Software and Data Technologies*, pages 257–260, Setubal, Portugal, September 2006. INSTICC press.

[HP] HP OpenView ServiceCenter, Hewlett Packard Corporation. http://h20229.www2.hp.com/products/ovsc/index.html.

[HSS04] A. Hanemann, M. Sailer, and D. Schmitz. Variety of QoS — the MNM Service Model Applied to Web Hosting Services. In *11th International Workshop of the HP OpenView University Association (HPOVUA 2004)*, volume 2004, Paris, France, June 2004.

[HSS05a] A. Hanemann, M. Sailer, and D. Schmitz. Towards a Framework for Failure Impact Analysis and Recovery with Respect to Service Level Agreements. In *Proceedings of the 9th IFIP/IEEE International Conference on Integrated Network Management (IM 2005)*, Nice, France, May 2005. IFIP/IEEE.

[HSS05b] A. Hanemann, M. Sailer, and D. Schmitz. Towards a Framework for IT Service Fault Management. In *Proceedings of the European University Information Systems Conference (EUNIS 2005)*, Manchester, England, June 2005. EUNIS.

[IBM] IBM Tivoli Application Dependency Discovery Manager, International Business Machines Corporation. http://www-306.ibm.com/software/tivoli/-products/taddm/.

[IRM02] A risk management standard. Technical report, Institute of Risk Management (IRM), The Association of Insurance and Risk Managers (AIRMIC) and

ALARM (The National Forum for Risk Management in the Public Sector), 2002.

[ITI] IT Infrastructure Library. http://www.itil.co.uk.

[itS] IT Service Management Forum. http://www.itsmf.com.

[JA01] Editor J.S. Armstrong. *Principles of Forecasting - A Handbook for Researchers and Practitioners (International Series in Operations Research & Management Science)*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[JLB04] G. Jakobson, L. Lewis, and J. Buford. An Approach to Integrated Cognitive Fusion. In *Proceedings of the 7th ISIF International Conference on Information Fusion (FUSION2004)*, pages 1210–1217, Stockholm, Sweden, June 2004.

[JW93] G. Jakobson and M. Weissman. Alarm correlation. *IEEE Network*, 7(6), November 1993.

[KHL+99] M. Katchabaw, S. Howard, H. Lufiyya, A. Marshall, and M. Bauer. Making Distributed Applications Manageable through Instrumentation. *The Journal of Systems and Software*, 45(2):81–97, 1999.

[KK01] A. Keller and G. Kar. Determining Service Dependencies in Distributed Systems. In *Proceedings of the IEEE International Conference on Communications (ICC 2001)*, Helsinki, Finland, June 2001.

[KKC00] G. Kar, A. Keller, and S. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In J. W. Hong and R. Weihmayer, editors, *NOMS 2000 IEEE/IFIP Network Operations and Managment Symposium — The Networked Planet: Management Beyond 2000*, pages 61–74, Honolulu, Hawaii, USA, April 2000. IEEE.

[KL02] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. In *IBM Research Report, RC22456 (W0205-171)*, Yorktown Heights, New York, USA, May 2002.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *ACM Journal*, 42(4):741–843, May 1995.

[KN96] R. Kaplan and D. Norton. *The Balanced Scorecard: Translating Strategy into Action*. Harvard Business School Press, Cambridge, Massachusetts, USA, 1996.

[Kol93] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, California, USA, 1993.

[KRI] Kriex.org - home of kri library and services. http://www.kriex.org.

[KS86] R. Kowalski and M. Sergot. A logic-based calculus of events. In *New Generation Computing 4 (1986)*, pages 67–94. Ohmsha, Ltd., 1986.

[KT03] C. Kruegel and T. Toth. Using decision trees to improve signature-based intrusion detection. In *Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection (RAID 03)*, Pittsburgh, Pennsylvania, USA, September 2003.

[Lam94] L. Lampert. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, May 1994.

[Lan01] M. Langer. *Konzeption und Anwendung einer Customer Service Management Architektur*. Dissertation, Technische Universität München, March 2001.

[Lan06] S. Lange. Formalisierung von Aggregationsvorschriften für Dienstinformationen. Diplomarbeit, Technische Universität München - in German. Diploma thesis, Technical University of Munich Germany, May 2006.

[Lew93] L. Lewis. A case-based reasoning approach for the resolution of faults in communication networks. In *Proceedings of the Third IFIP/IEEE International Symposium on Integrated Network Management*, pages 671–682, San Francisco, California, USA, April 1993. IFIP/IEEE.

[Lew95] L. Lewis. *Managing Computer Networks - A Case-Based Reasoning Approach*. Artech House, Inc., 1995.

[Lew99] L. Lewis. *Service Level Management for Enterprise Networks*. Artech House, Inc., 1999.

[LLN98] M. Langer, S. Loidl, and M. Nerb. Customer Service Management: A More Transparent View To Your Subscribed Services. In A. S. Sethi, editor, *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, October 1998.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, New York, 2 edition, 1987.

[LSE03] D. Lamanna, J. Skene, and W. Emmerich. SLAng: A Language for Defining Service Level Agreements. In *9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, San Juan, Puerto Rico, May 2003.

[Mar06] P. Marcu. Modellierung von Abhängigkeiten in IT-Diensten - in German. Diploma thesis, University of Munich, Department of Computer Science, Munich, Germany, June 2006.

[MWH98] S. Makridakis, S. Wheelwright, and R. Hyndman. *Forecasting: Methods and Applications*. John Wiley and Sons, Inc, 1998.

[Ner01] M. Nerb. *Customer Service Management als Basis für interorganisationales Dienstmanagement*. Dissertation, Technische Universität München, March 2001.

[Net] Netcool/impact. http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/-index.jsp?toc=/com.ibm.netcool_impact.doc/toc.xml.

[Net04] Managing Today's Mission-Critical Infrastructures: Discovery, Collection, Correlation, and Resolution with the Netcool Suite, Micromuse Incorporated. http://www.micromuse.com/downloads/pdf_lit/wps/Muse_Discovery_-Correlation_Resolution_Jan04.pdf, January 2004.

[NT89] S.A. Naquvi and S. Tsur. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, 1989.

[Off99] Office of Government Commerce (OGC), editor. *Security Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 1999.

[Off00] Office of Government Commerce (OGC), editor. *Service Support*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2000.

[Off01] Office of Government Commerce (OGC), editor. *Service Delivery*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2001.

[Off02a] Office of Government Commerce (OGC), editor. *Application Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.

[Off02b] Office of Government Commerce (OGC), editor. *ICT Infrastructure Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.

[Off02c] Office of Government Commerce (OGC), editor. *Planning to Implement Service Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.

[Off03] Office of Government Commerce (OGC), editor. *Software Asset Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2003.

[Off04] Office of Government Commerce (OGC), editor. *Business Perspective: The IS View on Delivering Services to the Business*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2004.

[Off07a] Office of Government Commerce (OGC), editor. *Continual Service Improvement*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2007.

[Off07b] Office of Government Commerce (OGC), editor. *Service Design*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2007.

[Off07c] Office of Government Commerce (OGC), editor. *Service Operation*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2007.

[Off07d] Office of Government Commerce (OGC), editor. *Service Strategy*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2007.

[Off07e] Office of Government Commerce (OGC), editor. *Service Transition*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2007.

[OMG07a] OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. TC Document formal/07-11-01, Object Management Group, November 2007.

[OMG07b] OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. TC Document formal/07-11-03, Object Management Group, November 2007.

[OSI92] Information Technology - Open Systems Interconnection - Structure of Management Information. Technical Report IS 10165, ISO Electrotechnical Committee, 1992.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[RCN99] S. Ramanathan, D. Caswell, and S. Neal. Auto-Discovery Capabilities for Service Management: An ISP Case Study. Technical Report HPL-1999-68, HP Laboratories, Palo Alto, California, USA, May 1999.

[RDF] Resource Description Framework, World Wide Web Consortium. http://www.w3.org/RDF.

[Ris] Risk analysis - overview. http://www.solver.com/risk-analysis/.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[RSM+07] R. Reboucas, J. Save, A. Moura, C. Bartolini, and D. Trastour. A decision support tool to optimize scheduling of it changes. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, Munich, Germany, May 2007. IFIP/IEEE, IEEE Publishing.

[Sai05] M. Sailer. Towards a Service Management Information Base. In *IBM PhD Student Symposium at ICSOC05*, Amsterdam, Netherlands, December 2005.

[Sai07] M. Sailer. *Konzeption einer Service-MIB - Analyse und Spezifikation dienstorientierter Managementinformation*. Dissertation, Ludwig–Maximilians–Universität München, July 2007.

[SB04] M. Salle and C. Bartolini. Management by contract. In *Proceedings of the 9th IEEE/IFIP International Network Operations and Management Symposium (NOMS 2004)*, pages 787–800, Seoul, Korea, April 2004. IFIP/IEEE, IEEE Publishing.

[Sch00] H. Schmidt. Service Contracts based on Workflow Modeling. In A. Ambler, S.B. Calo, and G. Kar, editors, *Proceedings of the 11th Annual IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2000)*, number 1960 in Lecture Notes in Computer Science (LNCS), Austin, Texas, USA, December 2000. Springer.

[Sch01] H. Schmidt. *Entwurf von Service Level Agreements auf der Basis von Dienstprozessen*. Dissertation, Ludwig–Maximilians–Universität München, July 2001.

[SGF02] G. Stoneburner, A. Goguen, and A. Feringa. Risk management guide for information technology systems. Nist special publication 800-30, National Institute of Standards and Technology, July 2002.

[Sma] Smarts buisness impact manager. http://www.emc.com/products/-software/smarts/bim/.

[Sma04] The InCharge Common Information Model, Smarts Corporation. http://www.smarts.com/resources/ICIM_white_paper.pdf, September 2004.

[SS04] M. Steinder and A. Sethi. A Survey of Fault Localization Techniques in Computer Networks. *Science of Computer Programming, Elsevier*, 53(1):165–194, 2004.

[SS06] S. Shankar and O. Satyanarayanan. An Automated System fo Analyzing Impact of Faults in IP Telephony Networks. In *Proceedings of the 10th IFIP/IEEE International Network Management and Operations Symposium (NOMS 2006)*, Vancouver, British Columbia, Canada, April 2006.

[SSK+06] M. Schmid, J. Schaefer, R. Kroeger, K. Sotnikov, and Y. Karpov. Combining application instrumentation and simulation to forecast costs and revenue in application service provisioning scenarios. In *The 13th International Workshop of the HP OpenView Univeristy Association (HPOVUA 2006)*, June 2006.

[TJ01] K.-D. Tuchs and K. Jobmann. Intelligent search for correlated alarm events in databases. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, pages 285–288, Seattle, Washington, USA, May 2001. IFIP/IEEE.

[TMF04a] Shared Information/Data (SID) Model, Addendum 4S0 - Service Overview Business Entity Definitions. NGOSS Release 4.0, TeleManagement Forum, August 2004.

[TMF04b] The NGOSS Technology-Neutral Architecture. NGOSS Release 4.5, TeleManagement Forum, November 2004.

[WSA05] Web Services Agreement Specification (WS-Agreement). Technical Report Version 7, OGF, Grid Resource Allocation Agreement Protocol WG, June 2005.

[xsb] Xsb project. http://xsb.sourceforge.net/.

[YK00] G. Yang and M. Kifer. Implementing an efficient dood system using a tabling logic engine. In *Intl. Conference on Computational Logic*, July 2000.

[YKZ03] G. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *In Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, Catania, Sicily, Italy, November 2003.

ABBREVIATIONS

-Appr ..... approach
-Fw ..... framework
-Inst ..... instantiation
-InstMeth ..... instantiation methodology
-Wf ..... workflow
AFS ..... Andrew Filesystem
ARM ..... Application Response Measurement
BAWf ..... basic abstract workflow
BCArch ..... basic component architecture
BDgrChar ..... business degradation characteristic
BExtIfcs ..... basic external interfaces
BFw ..... basic framework
BFwAppr ..... approach for development of basic framework
BI ..... business impact
BIA ..... business impact analysis
BIAFw ..... business impact analysis framework
BIDepMod ..... business impact dependency model
BIntComps ..... basic internal components
BRAWf ..... basic refined abstract workflow
BRWf ..... basic realized workflow
BWf ..... basic workflow
CDL ..... Contract Definition Language
CIM ..... Common Information Model
CIMOM ..... CIM Object Manager

CMDB . . . . . Configuration Management Database

Cobit . . . . . Common Objectives for Information and related Technology

CORBA . . . . . Common Object Request Broker Architecture

CSM . . . . . Customer Service Management

DDB . . . . . Deductive Database

DegDep . . . . . degradation dependency

DMTF . . . . . Distributed Management Task Force

eTOM . . . . . Enhanced Telecom Operations Map

F-Logic . . . . . Frame Logic

FCAPS . . . . . fault, configuration, accounting, performance, security

HTTP . . . . . Hypertext Transfer Protocol

i/o . . . . . input/output

I/R . . . . . impact/recovery

I/RA . . . . . impact and recovery analysis

I/RA Inst . . . . . impact and recovery analysis instantiation

I/RAFw . . . . . impact and recovery analysis framework

I/RAFw Inst . . . . . impact/recovery analysis instantiation

I/RAFw InstMeth . . . . . impact/recovery analysis instantiation methodology

I/RAFw TDInstMeth impact/recovery analysis top-down instantiation methodology

I/RAFwAppr approach for development of impact and recovery analysis framework

IA . . . . . impact analysis

IA Inst . . . . . impact analysis instantiation

IAFw . . . . . impact analysis framework

IAFwAppr . . . . . approach for development of impact analysis framework

IDepMod . . . . . impact dependency model

IETF . . . . . Internet Engineering Task Force

IMAP . . . . . Internet Message Access Protocol

IQL . . . . . identical query language

ISP . . . . . Internet Service Provider

ITIL . . . . . IT Infrastructure Library

itSMF . . . . . IT Service Management Forum

KPI . . . . . Key Performance Indicator
KRI . . . . . Key Risk Indicator
LDAP . . . . . Lightweight Directory Access Protocol
LDL . . . . . Logical Data Language
LMU . . . . . Ludwig-Maximilians University
LRZ . . . . . Leibniz Supercomputing Center
m/d i/o . . . . . model/dynamic input/output
MBO . . . . . Management by Business Objectives
MIB . . . . . Management Information Base
MOF . . . . . Managed Object Format
MWN . . . . . Munich Scientific Network
NGOSS . . . . . Next Generation Operation Support and Software
OGC . . . . . British Office of Government Commerce
OSI . . . . . Open Systems Interconnection
POP . . . . . Post Office Protocol
QML . . . . . Quality Management Language
QoD . . . . . quality of device
QoR . . . . . quality of resource
QoS . . . . . quality of service
QoX . . . . . subsumption of QoS and QoR/QoD
R0 . . . . . generic requirements for I/R analysis
R1 . . . . . requirements on service modeling in general
R2 . . . . . requirements on service degradation and quality modeling
R3 . . . . . requirements concerning the modeling of business impact
R4 . . . . . requirements for recovery action modeling
R5 . . . . . requirements for the course of action of I/R analysis
RA . . . . . recovery analysis
RA Inst . . . . . recovery analysis instantiation
RAct . . . . . recovery action
RAFw . . . . . recovery analysis framework
RAFwAppr . . . . . approach for development of recovery analysis framework

RecChgTrckDepMod . . . . . recovery changes tracking dependency model
RPlan . . . . . recovery plan
RT . . . . . recovery tracking
RT Inst . . . . . recovery tracking instantiation
RTFw . . . . . recovery tracking framework
RTFwAppr . . . . . approach for development of recovery tracking framework
SI . . . . . service impact
SIA . . . . . service impact analysis
SIAFw . . . . . service impact analysis framework
SID . . . . . Shared Information/Data Model
SIDepMod . . . . . service impact dependency model
SISL . . . . . Service Information Specification Language
SLA . . . . . service level agreement
slp . . . . . SLA penalty
slv . . . . . SLA violation
SMIB . . . . . Service MIB
SMONA . . . . . Service Monitoring Architecture
SMTP . . . . . Simple Mail Transfer Protocol
SNMP . . . . . Simple Network Management Protocol
SP/MI . . . . . IT service provisioning and management infrastructure
TMF . . . . . TeleManagement Forum
TUM . . . . . Technical University of Munich
UML . . . . . Unified Modeling Language
WBEM . . . . . Web Based Enterprise Management
WSLA . . . . . Web Service Level Agreement
XML . . . . . eXtensible Markup Language

CONVENTIONS FOR EXAMPLES

s_{x} ..... service x

f_{x} ..... service functionality x

$f_{\mathrm{x}}(param_1,...,param_n)$ ..... service functionality x with parameters

r_{x} ..... resource x

$r_{\mathrm{x}}(param_1,...,param_n)$ ..... resource x with parameters

c_{x} ..... service client x (as a special kind of resource)

q_{x} ..... quality parameter x

g_{x} ..... degradation x

$g_x(param_1,...,param_n)$ ..... degradation x with parameters

$src \rightarrow tgt$ ..... dependency from source(s) src to target(s) tgt

examples for dependencies between single objects:

$r_{\mathrm{x}} \rightarrow r_{\mathrm{y}}$ ..... dependency from resource x to resource y

$r_{\mathrm{x}} \rightarrow f_{\mathrm{y}}$ ..... dependency from resource x to functionality y

$f_{\mathrm{x}} \rightarrow f_{\mathrm{y}}$ ..... dependency from functionality x to functionality y

$q_{\mathrm{x}} \rightarrow q_{\mathrm{y}}$ ..... dependency from quality parameter x to quality paramater y

$g_{\mathrm{x}} \rightarrow g_{\mathrm{y}}$ ..... dependency from degradation x to degradation y

analogously dependencies with multiple sources, multiple targets,

and with objects with parameters

$dgrat(g_{\mathrm{x}})$ ..... rating for degradation g_{x}

$DgrHndl(g_{\mathrm{x}})$ ..... recovery handling (task) for pre-recovery degradation g_{x}

$RecAct(g_{\mathrm{x}})$ ..... recovery action (concretization of a recovery handling task) for pre-recovery degradation g_{x}

$ModAdp_{\mathrm{x}}$ ..... model adaption (task) x

$ModAdpAct_{\mathrm{x}}$ ... model adaption action (concretization of a model adaption task) x

$sla\_cnstr_{\text{mail<i>}}$ . . . . . SLA constraint $< i >$ for e-mail service

$sla\_norm\_cnstr_{\text{web<i>}}$ . . . SLA constraint $< i >$ for web hosting service (concerning normal customers/users)

$sla\_prem\_cnstr_{\text{web<i>}}$ . . . SLA constraint $< i >$ for web hosting service (concerning premium customers/users)

$sla\_pnlty_{\text{mail<i>}}$ . . . . . SLA penalty definition $< i >$ for e-mail service

$sla\_norm\_pnlty_{\text{web<i>}}$ . . . . . SLA penalty definition $< i >$ for web hosting service (concerning normal customers/users)

$sla\_prem\_pnlty_{\text{web<i>}}$ . . . . . SLA penalty definition $< i >$ for web hosting service (concerning premium customers/users)

slp_{x}, slp_norm_{y}, slp_prem_{z} . . . . SLA penalty functions x, y, z, corresponding to the respective SLA penalty definitions (see above)

INDEX