INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Abhängigkeitsmodellierung für das IT-Dienstmanagement

Gabriela Patricia Marcu

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Andreas Hanemann
Martin Sailer
David Schmitz

Abgabetermin: 14. Juni 2006

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. Juni 2006

. . . . . .

(Unterschrift der Kandidatin)

Zusammenfassung

In der heutigen Zeit gewinnt der Einsatz von Informationstechnologie (IT) in vielen Produktivbereichen immer mehr an Bedeutung. Davon sind die Durchführung von produktiven Prozessen, die Einhaltung von Vertragsvereinbarungen mit den Kunden und im Allgemeinen das ganze Geschäft abhängig. IT Service Management unterstützt die Verzahnung zwischen der IT und dem gesamten Geschäftskonzept. Außerdem spielt es, in Bezug auf das heutzutage aktuelle Thema des Outsourcings von IT-Diensten zur Steigerung der Qualität, eine große Rolle.

Der Ausfall oder die Störung bestimmter IT-Systeme kann unerwünschte Auswirkungen auf einen Teil oder den gesamten Geschäftsprozess haben. Zur effizienteren Vorbeugung und Behebung solcher Störungen und Probleme ist deshalb eine geeignete Strategie zur Fehlersuche unerlässlich. Um möglichst angemessen reagieren zu können, müssen des Weiteren die Auswirkungen, die ein Ausfall eines IT-Systems hat, eingeschätzt werden. Um Effizienz bzgl. dieser beiden Aspekte zu erreichen, müssen die bestehenden Abhängigkeiten zwischen den Ressourcen untereinander, zwischen den Diensten untereinander und zwischen Diensten und Ressourcen, identifiziert, analysiert und modelliert werden. Zwar existieren Ansätze zur Bestimmung und Modellierung der Abhängigkeiten, die wertvolle Informationen darüber liefern, jedoch existiert noch kein strukturierter Ansatz auf diesem Gebiet

In Rahmen dieser Arbeit wird, basierend auf den bestehenden Ansätzen im Bereich des Fehlermanagements, ein Abhängigkeitsmodell entwickelt. Für die Entwicklung dieses Modells wird u.a. das Composite Design Pattern als Methode des Softwaredesigns benutzt. Dieses ermöglicht einem Management Tool sowohl komplexe als auch einfache Abhängigkeiten in gleicher Weise zu betrachten. Das entwickelte Abhängigkeitsmodell wird anschließend, anhand zweier Beispielszenarien des Leibniz-Rechenzentrums (Web Hosting-Dienst und E-Mail-Dienst), prototypisch realisiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren nimmt die Dienstorientierung in der IT-Branche einen immer höheren Stellenwert ein. Es ist fast selbstverständlich geworden im Internet zu Surfen, eine E-Mail zu verschicken oder mit den Freunden zu chaten. Dass sind alles IT-Dienste die man bewusst oder „unbewusst“ in Anspruch nimmt. Wie kann man eigentlich IT-Dienste definieren? Es sind Leistungen im Bereich der IT, die von einem Dienstanbieter (engl. *Provider*) für einen Kunden erbracht werden, so dass der Kunde einen Nutzen davon hat. Überall findet man Beispiele von IT-Diensten, die erbracht und in Anspruch genommen werden, z.B. E-Mail, WWW, Webmail, Web-Hosting, Dateiübertragung und Internettelefonie. Die Erfahrung hat gezeigt, dass diese Dienste oft sehr viel besser betrieben werden können wenn sie an Drittanbieter ausgelagert sind. Diese Auslagerung von IT-Diensten an Drittanbieter ist als Outsourcing bekannt. Outsourcing stellt in der aktuellen Zeit eine der zentralen Problematiken im Bereich des IT-Dienstmanagement (engl. *IT Service Management*) dar. Dadurch dass die Auslagerung der Dienste sowohl eine Verbesserung der Qualität als auch eine deutliche Vereinfachung für die Kunden hervorbringt, werden diese zunehmend von externen Anbietern erbracht. Das heisst, dass immer mehr Unternehmen ihr Kerngeschäft von den IT-Diensten trennen. Das **Leibniz-Rechenzentrum** (**LRZ**) z.B. stellt für die **Ludwig-Maximilians-Universität** (**LMU**) und die **Technische Universität München** (**TUM**) den E-Mail- und den Web Hosting-Dienst bereit.

In dem oberen Fall ist das LRZ der Dienstleister (engl. *Provider*), der Dienste seinen Kunden (LMU und TUM) anbietet. Um eine geeignete Leistung für ihre Kunden zu erbringen, müssen die Provider organisatorische Fähigkeiten beweisen. Dieser Bereich der Organisation trägt den Namen IT-Dienstmanagement. „IT Service Management (ITSM) und ist die Gesamtheit der Aktionen der IT-Organisation, die notwendig sind, um die Geschäftsprozesse bestmöglich zu unterstützen. Er beschreibt den Wandel der Informationstechnik in Richtung Kunden- und Serviceorientierung.“[ITSS 00] Von Bedeutung ist die Sicherstellung und Überwachung von IT-Diensten. Mit Hilfe des IT Dienstmanagements kann kontinuierlich die Effizienz, die Qualität und die Wirtschaftlichkeit der jeweiligen IT-Organisation verbessert werden.

Ein wichtiger Aspekt von IT Service Management sind Dienstvereinbarungen (engl. **Service Level Agreement** (**SLA**)). Ein Service Level Agreement ist eine vertragliche Vereinbarung zwischen IT-Dienstleister und Kunde. Es werden beispielsweise Reaktionszeiten für Supportleistungen oder maximale Ausfallzeiten von IT-Services und deren quantitative Messung festgelegt (Definition gemäß [ITSS 00]). Das SLA beschreibt, verständlich für den Kunden, die IT-Services in nichttechnischen Begriffen. Die aufgetretenen Störungen, welche gegen das SLA verstoßen (aber auch die, die noch nicht dagegen verstoßen, aber eine kritische Situation verursachen könnten), werden vom Fehlermanagement bzw. Performancemanagement gelöst.

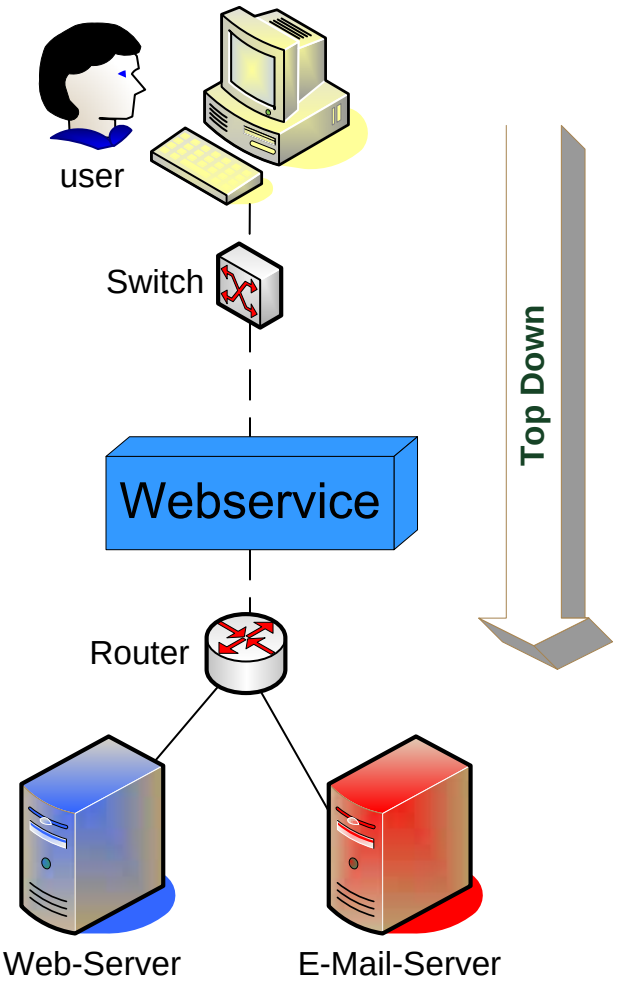

Abbildung 1.1: Top Down-Ansatz

Das Fehlermanagement spielt, für das Herausfinden und Beseitigen von Störungen, die in einem Betrieb auftreten können, eine bedeutende Rolle . Durch ein richtig durchgeführtes und effizientes Fehlermanagement kann man dazu beitragen dass die SLA eingehalten werden. Beispielsweise wird zwischen einem Provider und einem Kunden ein SLA festgelegt dass die Nutzer zwischen 9-17 Uhr zu 99,9% über den E-Mail-Service verfügen können. Das heißt, wenn ein Fehler innerhalb von 10 Tagen auftritt, muss man ihn in höchstens 4,8 Minuten beheben, so dass dieses SLA eingehalten werden kann. Daher sind Strategien zur Fehlerfindung und Fehlerbehebung (Fehlermanagement) wichtig. Bei der Fehlersuche ist insbesondere sehr wichtig, in welcher Beziehung die teilnehmende Komponenten zueinander stehen; um wissen zu können wie sich miteinander interagieren. Diese Beziehungen sind die so genannten Abhängigkeiten. Für den Anbieter ist es, unter diesen Umständen, von bedeutendem Interesse die Abhängigkeiten zwischen den Komponenten bei der Diensterbringung zu kennen.

Da das Fehlermanagement ein so wichtiger Teil des Netz- und Systemmanagements ist, haben die Aktivitäten zur Optimierung der Fehlersuche einen bedeutenden Platz in der Forschung (siehe z.B die Aktivitäten des **Munich Network Management** (**MNM**)-Teams) [GHHK 01]. Eine große Hilfe für das Fehlermanagement ist ein Modell zur Beschreibung der Abhängigkeiten von Teilkomponenten und/oder von Diensten.

Warum ist das Abhängigkeitsmodell für Provider wichtig? In Bezug auf das Fehlermanagement gibt es zwei Ansätze: top-down und bottom-up. Der Top Down-Ansatz ist in der Abbildung 1.1 dargestellt. Ein User meldet dass der Web-Dienst ausgefallen oder fehlerhaft sei. Anhand der vordefinierten Abhängigkeiten kann man die Ursache dieser Störung auf bestimmten Komponenten (z.B. Switch, Router, E-Mail-Server, Web-Server) eingrenzen. Wenn ein gutes Abhängigkeitsmodell vorhanden ist kann die Suche nach der Ursache sehr effizient erfolgen.

Im Falle des Bottom Up-Ansatzes (Abbildung 1.2) ist es umgekehrt. Wenn z.B. einer der Ausgangsrouter eines Rechenzentrums ausfällt, stellt sich die Frage was die Auswirkungen sind die von diesem Ausfall verursacht werden? Sind nur der E-Mail- und der Web-Dienst betroffen oder auch andere Dienste? Wie viele und welche Nutzer sind betroffen, in welchem Ausmaß ? Welche Dienste sind hiermit ausgefallen? Solche Fragestellungen, im Falle einer Störung, lassen die Nützlichkeit eines Abhängigkeitsmodells erkennen. Dieser Ansatz ist auch in der Netzplanung wichtig; da der Ausfall einer Komponente sehr große Auswirkungen haben kann. Um dieses schon in der Entwurfsphase abzufangen, sichert man die Komponenten durch Redundanz, so dass bei fehlerhaften Funktionen

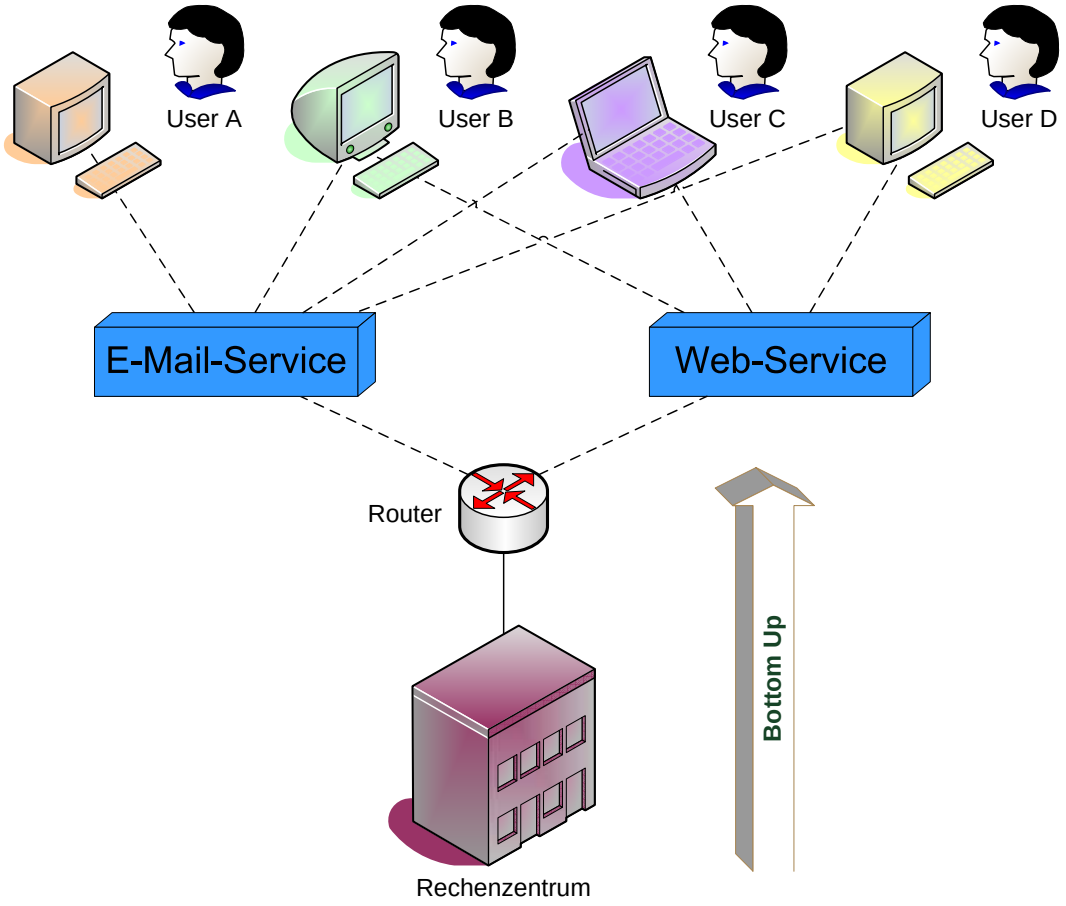

Abbildung 1.2: Bottom Up-Ansatz

eine Ersatzkomponente einspringt und die Funktion der ausgefallenen bisherigen Komponenete übernimmt. Der Benutzer bekommt dann nichts von dem Ausfall mit.

Abhängigkeitsmodellierung ist für den Provider ebenfalls zur Unterstützung von Providerhierarchien (ein Dienstanbieter ist selbst Kunde für einen anderen Dienstanbieter usw.)wünschenswert, obwohl in der Realität das Problem sehr komplex wird. Daher werden die providerinternen Abhängigkeiten zuerst untersucht. Bei den Abhängigkeiten kann man zwischen Abhängigkeiten von Diensten und Subdiensten (z.B. ein E-Mail-Dienst braucht einen **Domain Name Service** (**DNS**)), Abhängigkeiten zwischen Diensten und Ressourcen (z.B. der E-Mail-Dienst ist abhängig von der Funktion des E-Mail-Servers) sowie Abhängigkeiten zwischen den Ressourcen untereinander (z.B. die Verbindung zwischen LMU-Mailserver und LRZ-Mailserver) unterscheiden. Es ist wichtig ein allgemeingültiges Informationsmodell zu besitzen, dass die Abhängigkeiten beschreibt und darstellt. Ein Informationsmodell ist eine abstrakte Abbildung von Objekten mit ihren Eigenschaften und Beziehungen, sowie den Aktionen die sie ausführen können und den Operationen die mit ihnen durchgeführt werden können [BeSc96]. Informationsmodelle werden sehr oft im ITSM benutzt. Es wurden einige Versuche auf diesem Gebiet gemacht, und einige dieser Modelle haben sich als Standard durchgesetzt, insbesondere im Netz- und Systemmanagement, der Ressourcenmodellierung und des Internetmanagements. Für die Dienstmodellierung hat sich allerdings bisher kein allgemein akzeptierter Standard etabliert.

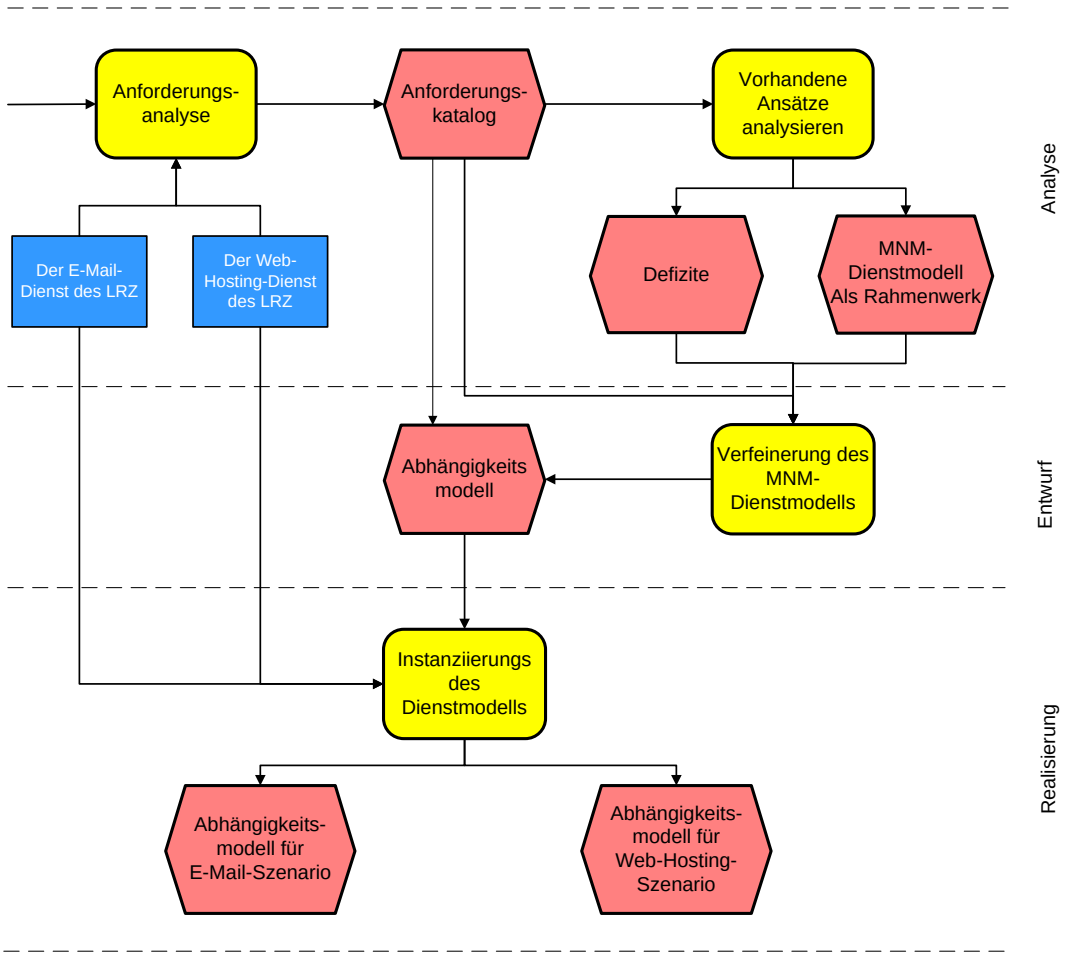

Abbildung 1.3: Vorgehensweise der Arbeit

1.1 Aufgabenstellung

Ziel der Diplomarbeit ist es zu untersuchen, wie man die Abhängigkeiten in geeigneter Weise darstellen kann. Zunächst soll untersucht werden welche Kriterien für die Modellierung wichtig sind. Insbesondere soll es ermöglicht werden, Probleme mit der Erbringung der Dienstqualität, in Zusammenhang mit den Ressourcen zu bringen; und auch umgekehrt, aus Problemen mit Ressourcen, auf betroffene Dienste schließen zu können. In dieser Arbeit sollten verschiedene Aspekte der Abhängigkeitsmodellierung betrachtet werden: die Bestimmung und die tatsächliche Modellierung der Abhängigkeiten. Es soll, nach genauer Analyse der Anforderungen für das Abhängigkeitsmodell, eine Methode zur Bestimmung der Abhängigkeiten gewählt werden. Zur konkreten Modellierung der Abhängigkeiten soll eine Verfeinerung des MNM-Dienstmodells vorgenommen werden.

Als Ergebnis soll ein Abhängigkeitsmodell geliefert, dass die Abhängigkeiten und deren Beziehungen zueinander darstellt. Wichtig, für so ein Abhängigkeitsmodell, ist die Betrachtung der Eigenschaften der Abhängigkeiten in Form von Attributen. Dieses Modell, das entwickelt werden soll, muss auf alle Dienste allgemein anwendbar sein.

1.2 Vorgehensweise

In diesem Abschnitt wird die Vorgehensweise der Arbeit dargestellt (siehe Abbildung 1.3). Die in der Abbildung 1.3 benutzten Konventionen sind: gelbe gerundete Rechtecke (Aktionen die durchgeführt werden), rote Sechsecke (Zwischenergebnisse) und blaue Rechtecke (Informationen die als Input benutzt werden). Am Anfang wird die Anforderungsanalyse durchgeführt, und als Ergebnis dessen entsteht ein Anforderungskatalog. Als Ausgangsinformationen, für die Anforderungsanalyse, werden die zwei Dienst-Szenarien des LRZ benutzt (die blauen Rechtecke in der Abbildung). Die Anforderungsanalyse wird realisiert um die spezifischen Aspekte, die zur Modellierung der Abhängigkeiten notwendig sind, destillieren und analysieren zu können. Dies ist eine unabdingbare Vorausetzung für den Entwurf eines Abhängigkeitsmodells.

Der entstandene Anforderungskatalog wird als Input für die nachfolgende Analyse der vorhandenen Ansätze benutzt. Diese Analyse ist, neben der Anforderungsanalyse, ein zweiter wichtiger Punkt der bei der Modellierung in Betracht gezogen werden muss. Die Erfahrungen, die auf diesem Gebiet gemacht worden sind, sind sehr reich an Informationen - die in die Modellierung einfließen. Daraufhin werden, in Bezug auf den Anforderungskatalog, einige Defizite aber auch Vorteile der vorhandenen Modelle aufgezeigt, und es wird dann das allgemeine MNM-Dienstmodell als Grundlage für die weitere Entwicklung des Abhängigkeitsmodells ausgewählt.

Nach der Analysephase folgt die Entwurfsphase. Mit den schon vorhandenen Daten wird das MNM-Dienstmodell verfeinert und danach das Abhängigkeitsmodell entworfen; als auch eine entsprechende Methodik zur spezifischen Abhängigkeitsmodellierung. Die Abhängikeitsmodellierung muss so allgemein wie möglich durchgeführt werden. Das MNM-Dienstmodell wird demnach insbesondere als Werkzeug zur Bestimmung der Abhängigkeiten benutzt. Nach dem Entwurf wird gezeigt in welchen Ausmaß das Abhängigkeitsmodell die Anforderungen (aus dem Anforderungskatalog) erfüllt.

Als Unterstützung der Realisierung eines Abhängigkeitsmodells werden das E-Mail-Dienst- und das Webhosting-Dienst-Szenario benutzt. In der Realisierungsphase wird, entsprechend der entwickelten Methodik für die zwei Dienste des LRZ, das Abhängigkeitsmodell instantiiert. Die Anwendung auf diese bekannten Szenarien zeigt ob das entwickelte Modell leicht anwendbar und insbesondere korrekt ist.

1.3 Gliederung der Arbeit

In Kapitel 2 wird die Anforderungsanalyse durchgeführt. Das heißt, es werden feste Anforderungen an das Abhängigkeitsmodell identifiziert. Diese sind auf beliebige Dienste allgemein anwendbar . Dabei werden die Daten in einem Anforderungskatalog zusammengefasst.

In Kapitel 3 werden die vorhandenen Ansätze (insbesondere das MNM-Dienstmodell) auf diesem Gebiet, in Bezug auf die gefundenen Anforderungen, dargestellt und bewertet.

Kapitel 4 befasst sich mit der generischen Modellierung von Abhängigkeiten. Es wird zusätzlich eine Methodik zur Verfeinerung des MNM-Modells vorgeschlagen. Dazu werden beide Views separat betrachtet, mit dem Ziel die entsprechenden Funktionalitäten zu verfeinern. Durch diese Verfeinerung werden die eigentlichen Abhängigkeiten definiert.

In Kapitel 5 wird, im Rahmen dieser Arbeit, das in Kapitel 4 realisierte Modell an den zwei Szenarien des LRZ (E-Mail und Web Hosting-Dienst) angewandt. Dafür wird eine entsprechende Verfeinerungsmethodik benutzt.

Im letzten Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst und einige Anregungen für weitere Arbeiten gegeben.

Kapitel 2

Anforderungsanalyse

Inhaltsverzeichnis

In diesem Kapitel wird die Anforderungsanalyse für den Entwurf eines Abhängigkietsmodells durchgeführt. Die Anforderungen zu identifizieren und zu analysieren sind wichtige Voraussetzungen für die Realisierung eines allgemein anwendbaren Abhängigkeitsmodells. Erst wenn alle Anforderungen an das Modell vollständig definiert worden sind, kann man es entwerfen. Um die Anforderungen erklären zu können, werden zwei Anwendungsbeispiele (E-Mail- und Web Hosting-Dienst des Leibniz Rechenzentrums) benutzt und die gefundenen Anforderungen werden abstrahiert.

Um die in dieser Arbeit verwendeten Begriffe zu erläutern werden in Abschnitt 2.1 kurz das MNM-Dienstmodell und in Abschnitt 2.2 die zwei Anwendungsszenarien vorgestellt. Danach (Abschnitt 2) wird die Anforderungsanalyse realisiert. In Abschnitt 2.4 wird ein Dimensionsbild der Abhängigkeiten vorgeschlagen und in Abschnitt 2.5 wird zusammenfassend eine Liste der Anforderungen, in Form eines Anfordeungskatalogs, zusammengestellt.

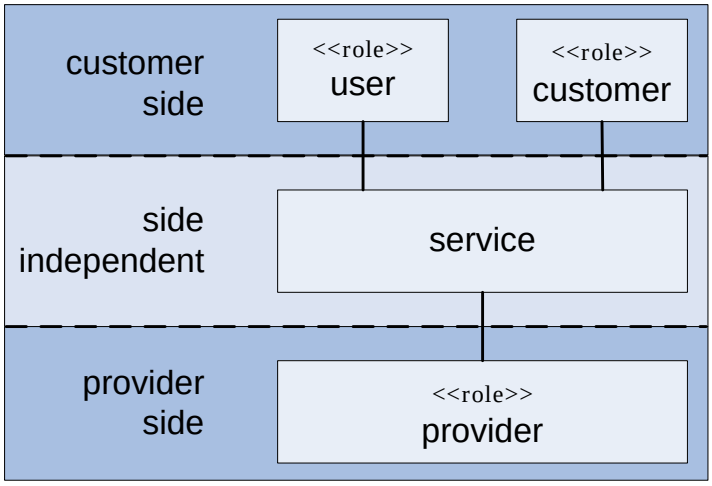

Abbildung 2.1: Basis MNM-Dienstmodell [GHHK 01]

2.1 Begriffserklärung

In diesem Absatz wird das MNM-Dienstmodell kurz beschrieben, während im Kapitel 3 eine detaillierte Beschreibung davon folgen wird. Dies wird gemacht weil diese Begriffe in der Anforderungsanalyse (Abschnitt 2) benutzt werden.

Das MNM-Dienstmodell [GHHK 01] spezifiziert einen formalen Rahmen, der hilft, ein benutzerorientiertes und qualitätsorientiertes Modell eines spezifischen Dienstes zu erstellen. Es unterstützt sowohl die Planung, Bereitstellung, und Betreibung eines Dienstes (service planing, provisioning and operation) als auch das Dienstmanagement an der Kunde-Anbieter-Schnittstelle (customer-provider-interface). Das Modell ist in drei Bereiche eingeteilt (siehe Abbildung 2.1) und weist folgendermaßen Rollen zu:

- auf der Kundenseite (engl. *customer side*) die Rollen `Nutzer` (user) und `Kunde` (customer), die den beiden Funktionalitäten: Nutzungs- und Managementfunktionalität entsprechen.
- auf der Anbieterseite (engl. *provider side*) die Rolle `Anbieter` (provider), die die beiden obengenannte Funktionalitäten anbietet.
- der unabhängige (engl. *side independent*) Bereich beinhaltet keine Rollen, sondern nur den Dienst selbst, d.h. die Dienstfunktionalitäten und die QoS-Parameter (**Quality of Service** (**QoS**)), die zur Realisierung des Dienstes beitragen.

Eine sehr mächtige Eigenschaft des MNM-Dienstmodells ist die Rekursivität, die die Realisierung von Dienst- und Providerhierarchien ermöglicht. Eine wichtige Problematik die das MNM-Dienstmodell hervorbringt ist die Tatsache, dass die Erbringung des Dienstes von der internen Realisierung getrennt werden soll . Das heißt der Kunde, der einen Dienst in Ansprunch nimmt, ist prinzipiell nicht an der internen Realisierung interessiert. Demnach hat das MNM-Modell zwei Sichten: 1. die Dienstsicht (Service View), die die Kunden-Provider-Beziehung (Customer Provider Relationship) darstellt, und 2. die Realisierungssicht (Realization View), die die providerinterne Realisierung des Dienstes darstellt. Diese werden im Abschnitt 3.1.1.4 umfassend beschrieben.

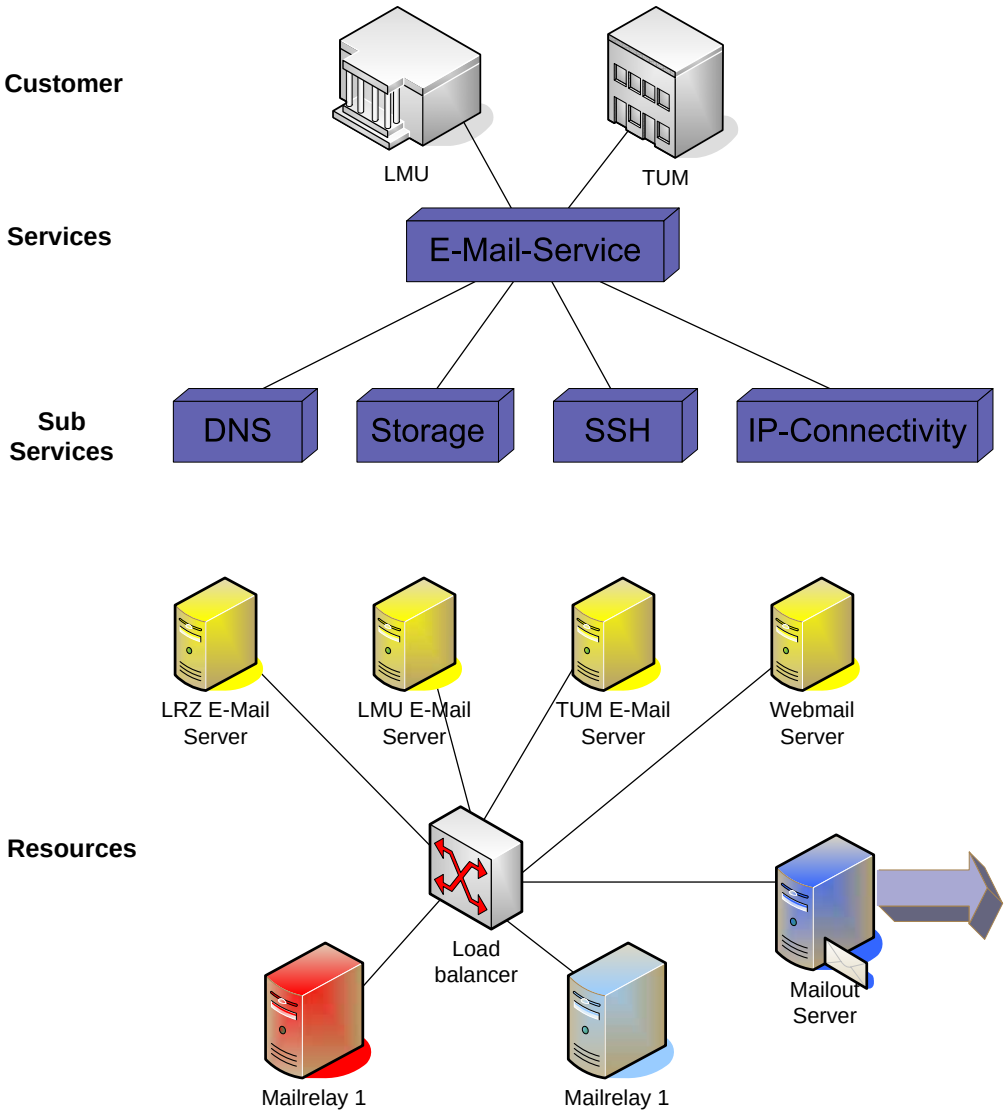

Abbildung 2.2: Der E-Mail-Dienst des LRZ

2.2 Anwendungsszenarien

Als Anwendungsszenarien wurden der E-Mail- und der Web Hosting-Dienst des LRZ ausgewählt; weil sie eine realitätsnahe Situation repräsentieren. In diesem Unterkapitel werden diese nur kurz vorgestellt und erst später wird eine detaillierte Beschreibung folgen (siehe Kapitel 5). Das LRZ, als IT-Dienstanbieter der Hochschulen im Raum München, verwaltet ein Netzwerk mit mehr als 60 Standorten und über 50.000 Benutzern (Studenten und Mitarbeiter), und stellt seinen Kunden (Customers) unterschiedliche Dienste zur Verfügung.

Für die Modellierung des Szenarios, wie in den Abbildungen 2.2 und 2.3 ersichtlich ist, wurde die Realisierung des Dienstes in drei hierarchische Ebenen aufgeteilt: 1. die des Kunden, 2. die des Services und 3. die der Ressourcen (der Infrastruktur). Auf der Service-Ebene sind zwei Arten von Diensten dargestellt. Mit **„Dienst“** bezeichnet man in dieser Arbeit den Dienst, der dem Kunden angeboten

wird. Mit **„Subdienst“** wird einen Dienst bezeichnet, der zur Realisierung des Ersten beiträgt. Das heißt nicht, dass der Subdienst untergeordnet ist oder weniger wichtig als ein Dienst ist. Es sind eigentlich alles gleichwertige Dienste. Nach dem MNM-Dienstmodell werden in der Kategorie der Ressourcen auch die Belegschaft (Mitarbeiter), Software (Anwendungen) und jegliche andere ,,materielle und geistige“ (z.B. Dokumentation) Unterstützungen für die Realisierung des Dienstes zusammengefaßt.

Der E-Mail- und der Web-Hosting-Dienst des LRZ werden mehreren Kunden angeboten, wie z.B. der LMU und der TUM. Für diese Kunden ist die tatsächliche Realisierung nicht relevant, sondern nur die Funktionalität des E-Mail- und/oder des Web-Hosting-Dienstes, die ihnen angeboten werden und die sie in Anspruch nehmen können. Danach wird die Dienstebene beschrieben, in der die tatsächliche Realisierung dieser Dienste dargestellt wird. Auf dieser Ebene werden die unterschiedlichen Subdienste, die für die Realisierung des Dienstes notwendig sind, beschrieben. In der untersten Ebene werden die Beziehungen zwischen den unterschiedlichen Ressourcen definiert.

2.2.1 Der E-Mail-Dienst

Der E-Mail-Dienst wird angeboten, als ein schneller und bequemer Weg, um Nachrichten zwischen unterschiedlichen Rechnern auszutauschen. Es gibt sehr viele Mailserver im heutigen Internet, aber die existierenden können über Gateways miteinander kommunizieren. Innerhalb der Mail-Systeme wird die Weiterleitung der Mails von den **Mail-Relays** übernommen.

Am LRZ wird der E-Mail-Dienst, wie in der Abbildung 2.2, realisiert. Es werden daher einige Unterdienste benötigt, ohne die der E-Mail-Dienst nicht laufen könnte. Diese sind: DNS, SSH, Proxy und Speicher(Storage).

Die Infrastruktur, die zur Realisierung des Dienstes zur Verfügung steht, beinhaltet zwei Mail-Relays. Diese sind durch einen Load-Balancer untereinender redundant und ihrerseits mit den jeweiligen Mail-Servern verbunden (LMU-, TUM-, LRZ-, Webmail Servern). Der virtuelle Rechner ,,mailout“ wird dynamisch auf ,,mailrelay1“ oder ,,mailrelay2“ abgebildet. Aus der Sicht der Mailserver im **Münchner Hochschulnetz** (**MHN**) werden *mailrelay1* und *mailrelay2* für ankommende/empfangene e-Mails genutzt, während *mailout* von den Mailclients für abgehende/gesendete e-Mails genutzt wird.

2.2.2 Der Web Hosting-Dienst

Das LRZ bietet den Kunden des **Münchener Wissenschaftsnetzes** (**MWN**) (ca. 200 Institute und Einrichtungen) die Möglichkeit, Webseiten über die Server am LRZ ins Internet zu stellen. Der Vorteil dabei ist, dass die Institute oder Einrichtungen keinen eigenen Webserver haben müssen, sondern sich nur auf die Erstellung und die Pflege der Webseiten konzentrieren können. Der Begriff ,,virtueller WWW-Server“ wird als Bezeichnung für diese am LRZ betriebene Webpräsenz (inklusive Domainhosting) benutzt. Der interne Aufbau des virtuellen Servers ist für die Institute überhaupt nicht wichtig. Sie nehmen nur die Funktionalität des Web Hosting-Dienst des LRZ in Anspruch. Die Realisierung des Web-Hosting-Dienstes [WEB 05] am LRZ ist in Abbildung 2.3 grafisch dargestellt.

Um überhaupt funktionsfähig zu sein benötigt der Web-Hosting-Dienst mehrere Unterdienste: DNS, IP-, Proxy-, und Storage-Dienste.

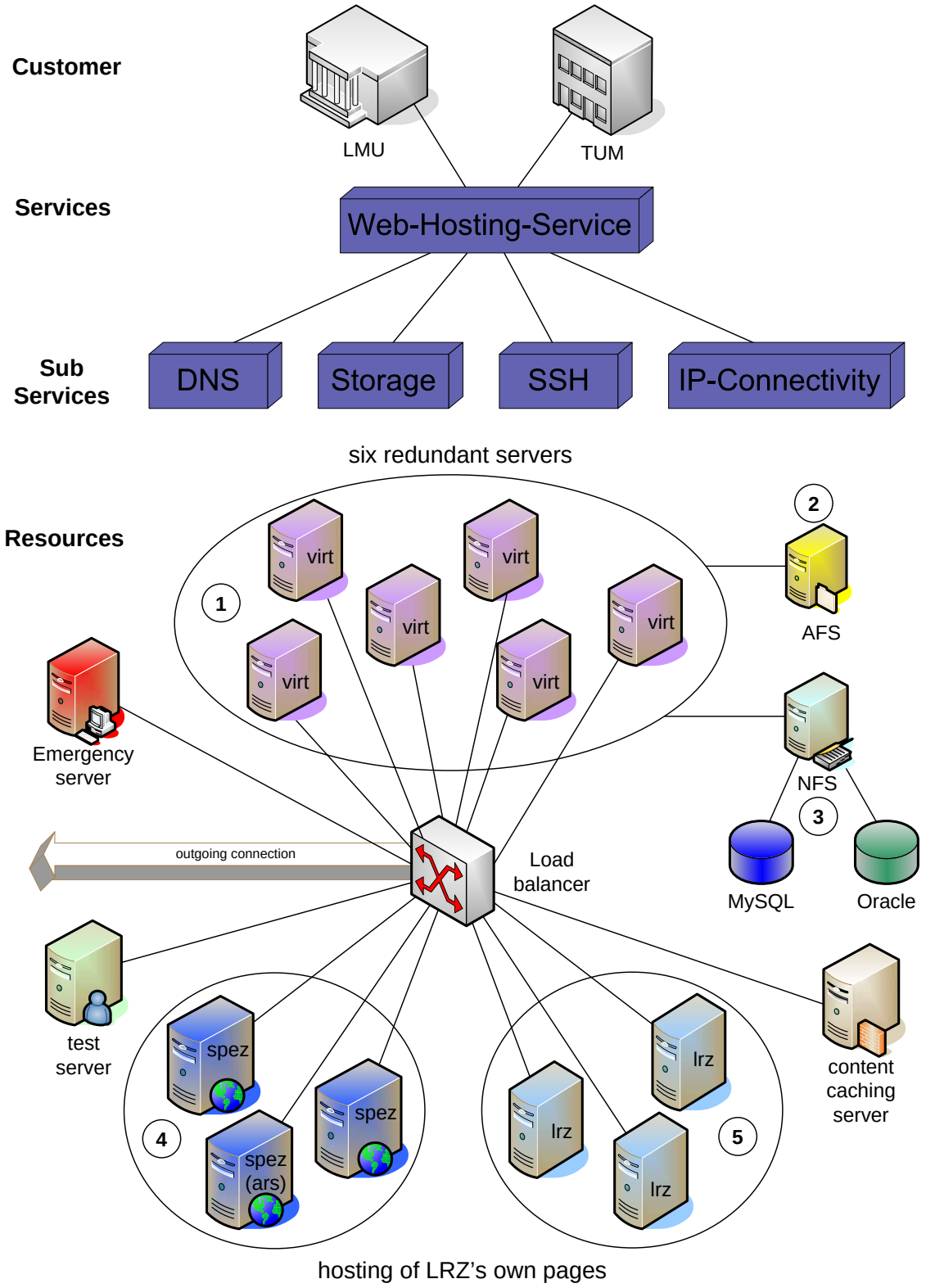

Abbildung 2.3: Der Web Hosting-Dienst des LRZ

Wie Bitterich in [Bitt 05] beschreibt, gibt es vier unterschiedliche Konfigurationen der Webserver, je nachdem für welchen Zweck sie eingesetzt werden:

lrz ist auf die Anforderungen der LRZ-Webseiten eingerichtet (statisch kompiliert).

virt ist so eingerichtet, dass er die virtuellen Webserver der Kunden darstellt (statisch kompiliert).

spez bietet eine spezielle Konfiguration für die Funktionalitäten die eine spezielle Umgebung benötigen, wie z.B. der Webmaildienst und das Webinterface ,,phpMyAdmin“, die auf den Datenbankserver zugreifen (nicht statisch kompiliert, sondern per ,,Dynamic Shared Objects“(DSO)).

ars steht für Action Request System der Firma Remedy und wird für das Fehlermanagement und die Dokumentation von Hard- und Software benötigt.

Auf der Ebene der Ressourcen ist zu erkennen dass die Infrastruktur weitaus komplexer ist als bei der Realisierung des E-Mail-Dienstes. Es werden sechs redundante Server (**virt**), die die virtuellen Webserver als auch deren Webseiten verwalten, über einen Load-Balancer verbunden (siehe ① im Abbildung 2.3). Die wichtigste Rolle des Load-Balancers ist die der Lastverteilung. Das AFS (Andrew File System)-Cluster (siehe ② im Abbildung 2.3) ist mit den sechs redundanten Servern verbunden und bietet eine größere Sicherheit, insbesondere gegenüber unbefugten Zugriffen und der Replizierfähigkeit, Auf den AFS-Servern werden die Webseiten derjenigen, die eine AFS-Kennung besitzen (Master-User), gespeichert. Das Network File System (NFS) ermöglicht den Zugriff auf ein vorhandenes Dateiesystem, über ein Netzwerk (siehe ③ im Abbildung 2.3). Im AFS werden die statischen Seiten gespeichert oder die dynamischen im Safe-Modus und im NFS die dynamischen **cgi**-Skripte. Dabei werden die Dateien nicht, wie z.B. bei FTP, jedesmal vollständig übertragen, sondern die Benutzer können auf Dateien, die sich auf einem entfernten Rechner befinden, so zugreifen, als wenn sie auf ihrer lokalen Festplatte abgespeichert wären. Daraufhin wird der NFS-Server an die Datenbanken gekoppelt. Das heißt, dass die dynamischen Skripte, die auf NFS laufen können, die Oracle und MySql Datenbanken nutzen (siehe ③ im Abbildung 2.3). Zusätzlich zur Speicherung wird AFS auch für Autorisierung benutzt, das heißt die Passwörter sind dort gespeichert.

Über die Load-Balancer (graphisch nur einer dargestellt), die auch als *BIP-IP* oder *L4/7 Switch* oder *Server Load Balancer(SLB)* bezeichnet werden, werden auch die sechs redundanten Server, die die LRZ- eigenen Webseiten verwalten, angeschlossen. Auf drei der redundant verbunden Maschinen ist die **lrz**-Konfiguration eingespielt (siehe ④ im Abbildung 2.3), die für die Verwaltung der LRZ internen Webseiten zuständig ist. Die anderen drei Server sind mit der **spez**-Konfiguration ausgestattet (siehe ⑤ im Abbildung 2.3). Auf diesen laufen die LRZ internen Tools, die Webmail Funktionalität und das PhpMyAdmin Tool. Die **ars**-Konfiguration ist zusammen mit der **spez**-Konfiguration auf einem der Servern eingerichtet.

Der Load-Balancer ist dann mit dem Emergency-Server, dem Test-Server und dem Content-Caching-Server verbunden. Der Emergency-Server wird im Falle eines größeren Ausfalles erforderlich, indem zumindest eine allgemeine Benachrichtigungsseite dargestellt werden kann. Der Test-Server wird für Versuchszwecke verschieder Konfigurationen benutzt. Der Content-Caching-Server ist für die dynamischen Seiten zuständig. Der Load-Balancer stellt aber auch die Verbindung ,,nach außen“ her. Das heißt dass alle 14 Rechner (Server) über den Load-Balancer an das Hochschulnetz angeschlossen sind.

2.3 Anforderungsanalyse für das Abhängigkeitsmodell

Am Anfang jedes Entwurfes eines Modells werden alle wichtigen Informationen für das zukünftige Modell zusammenstellt. Das wird durch eine genaue Untersuchung der Anforderungen, die man an das Modell stellt, bewerkstelligt. Dies wird wiederum aufgrund vorher definierter Szenarien am MNM-Dienstmodell durchgeführt. Alle Arten von Abhängigkeiten werden dabei aufgelistet und analysiert. Das Ziel ist die Erstellung eines Anforderungskatalogs, der das Obengenannte beschreibt. Für diese Arbeit sind die von Dombach [Domb 97] und Keller [Kell 98] dargestellten Methodiken zur Realisierung des Kriterienkatalogs wichtig, da einige Anforderungen auch auf das Abhängigkeitsmodell im allgemeinen zutreffen. In [Bitt 05] erstellt Bitterich einen Anforderungskatalog für ein Dienstmanagement-Informationsmodell.

Im Abschnitt 2.3.1 werden die allgemeinen (dienstbezogenen) Anforderungen erläutert. Die abhängigkeitsbezogenen Anforderungen folgen dann mit einer Darstellung der Arten von Abhängigkeiten in Abschnitt 2.3.2, einem Einblick in die Dynamik der Abhängigkieten im Abschnitt 2.3.3 und einige Aspekte bzgl. Redundanzen im Abschnitt 2.3.4.

2.3.1 Dienstbezogene Anforderungen

Die dienstbezogenen Anforderungen beziehen sich auf die Allgemeinheit des Modells; wie die Einhaltung allgemeiner Regeln bzgl. der Dienstdefinition, der Funktionalitäten, der Dienstparameter usw. bzw auf seine Fähigkeit mit schon bestehenden Ansätzen zu interagieren.

Technologieunabhängigkeit Man versteht darunter dass die Lösung unabhängig vom Hersteller und der Technologie sein muss. Das Modell muss unabhängig von den meisten hestellerabhängigen Technologien und muss mit bestehenden Produkten kompatibel sein.

Beim LRZ werden beispielsweise mehrere Tools für Netzmanagement, Anwendungsmanagement, Accountmanagement, Fehlemangement usw. eingesetzt. Dadurch dass sie technologieunabhängig sind, können sie alle untereinander interagieren.

Allgemeinheit des Dienstbegriffes Der Dienstbegriff muss ganz allgemein definiert werden. Definitionen, die nur auf einige Dienste anwendbar sind, sind sehr weit entfernt vom Bild eines idealen, gemeinsamen Dienstmodells. Dienste existieren in letzter Zeit nicht mehr einzeln; sondern oft ist es so, dass ein Provider ein Bündel (engl. *Bundle*) von Diensten anbietet. Und auch darauf muss der Dienstbegriff anwendbar sein. Der Dienst muss sowohl aus dem Gesichtspunkt des Kunden als auch aus dem des Providers definiert werden können.

Lebenszyklus Ein Modell muss alle Phasen des Lebenszykluses eines Dienstes (design, negotiation, provisioning, usage und deinstallation) [Dreo 02] in Betracht ziehen. Alle Phasen sind für ein Modell sehr wichtig, denn es kommt öfter vor dass Dienste, aufgrund erneuter Anforderungen, von Kunden geändert werden.

Der Lebenszyklus kann auf das Fehlermanagement angewendet werden. Sein eigener Lebenszyklus ist in den Zuständen eines Trouble Tickets (siehe die Bearbeitung eines TT auf der Seite 98) ausgedrückt.

Beispielsweise zeigt der Zustand *Erfasst* dass ein Kunde ein Problem gemeldet hat, und der Zustand *in Bearbeitung* gibt an dass der Provider mit der Fehlersuche angefangen hat. Der Zustand *Fehler gefunden* zeigt das der Fehler gefunden worden ist und der Zustand *Geschlossen* zeigt das der Fehler aufgehoben und das Problem für den Kunden gelöst worden ist [Dreo 95].

Dienstfunktionalitäten Die Funktionalität eines Systems ist das wichtigste für die Erbringung des Dienstes. Mehrere Kategorien von Funktionalitäten werden definiert, die auf jeden Dienst anwendbar sind. Diese müssen gleichzeitig den Nutzer, der eine Funktionalität in Anspruch nimmt, als auch den Anbieter, der die Funktionalität bereitstellt, in Betracht ziehen.

Beispielsweise sind für den E-Mail-Dienst die folgenden drei Funktionalitäten sehr wichtig: Mail senden, Mail empfangen, Webmail. Diese stellt er separat, aber auch zusammengefasst, zur Verfügung. jede der Funktionalitäten separat zu betrachten, wäre z.B. für das Einschränken der Fehlersuche notwendig. Wenn einen Kunde Mails versenden aber keine Mails erhalten kann, dann ist die Fehlersuche auf die Komponenten, die den Maileingang realisieren, eingeschränkt. Es wird dann der *mailrelay1* und der *mailrelay2* überprüft, oder die zwei *resolver*; aber auf keinen Fall der *mailout*-Server.

Dienstparameter Das sind Kennzahlen, die für die Diensterbringung benötigt werden. Man unterscheidet zwischen allgemeinen Parametern und QoS-Parametern. Die QoS- Parameter sind auf beiden Seiten – Kunde und Provider – bekannt und müssen von den letzteren eingehalten werden. Schmidt unterscheidet in [Schm 01] zwischen Primärkennzahlen (QoS-Parameter) und Sekundärkennzahlen (hier so genannte Dienstparameter). Beispielsweise kann man die Auslastung der Komponenten in einem System betrachten. Solange die QoS-Parameter eingehalten werden, ist die Auslastung eigentlich nicht von Bedeutung. Aber man kann anhand dieser Dienstparameter erkennen, wie gut der Provider einen Dienst unter Kontrolle hat. Es sind Parameter, die intern für den Provider wichtig sind.

Ein anderer sehr einfachen Dienstparameter, für den Web Hosting-Dienst beim LRZ, ist der Kunde. Eine Webseite, für ein Institut oder für einen Studenten, wird auf zwei verschiedene Weisen erstellt. Der Student könnte einen User des Instituts, aber auch direkt ein Kunde des LRZ sein; indem er seine Webseite (`www.stud.lrz-muenchen.de/ vorname.nachname`) dort speichert.

Erweiterbarkeit Ein Modell muss vom vorhandenen Zustand aus erweiterbar sein. Das heißt, auf dem bestehenden Modell sollte man weiter aufbauen können. Wenn das nicht der Fall sein sollte, ist es kein geeignetes Modell, weil im Modellierungsprozess immer wieder neue Elemente hinzu kommen. Diese müssen als Erweiterung des schon bestehenden Modell betrachtet werden. Hier wird auch der Zeitaufwand in Betracht gezogen; das bedeutet: ein leicht erweiterbares Modell ist viel wertvoller wie ein Modell, das schwierig zu erweitern ist.

Wenn das LRZ den Web Hosting-Dienst eines Tages nicht nur den Instituten der LMU und der TUM anbieten würde, sondern auch anderen externen Kunden, dann müsste man diesen Dienst erweitern können, bzw. das Modell dieses Dienstes muss dies zulassen können. Wenn das Modell neu ausgedacht und implementiert werden müsste, wäre es nicht so effizient, als wenn es nur um einige Teile erweitert werden würde.

Anwendbarkeit Wünschenswert an einem Modell ist eine einfache Benutzbarkeit und die Anwendbarkeit auf die tatsächlichen Dienste. Je mehr Komplexitäten in Betracht gezogen werden

müssen, umso mehr sinkt das Interesse an dem Ansatz. Die Elemente des Modells müssen einfach definiert werden, mit genügend Eigenschaften, aber trotzdem nicht zu vielen, um den Umgang damit leicht zu halten.

Bereitstellung der Dienstgüte Das bezieht sich auf die Festlegung der QoS-Parameter durch die SLAs, und die Kontrolle sowie deren Einhaltung während des Ablaufs eines Dienstes. Dass die QoS-Parameter eingehalten werden ist eine Voraussetzung für die angemessene Funktionalität des Dienstes. Hier kann man zwischen den QoS-Parametern, aus der Sichtweise des Kunden und denen des Providers unterscheiden. Aus der Sicht des Kunden messen die QoS-Parameter die Qualität des angebotenen Dienstes (anwendungsorientiert), wie beispielsweise die Verfügbarkeit des IP-Dienstes bei dem Service Access Point (SAP). Aus Sicht des Providers messen die QoS-Parameter die Qualität des Dienstes von einem operationalen Gesichtspunkt, wie z.B. die Verfügbarkeit des IP-Dienstes für alle Kunden (d.h. alle SAPs).

2.3.2 Arten von Abhängigkeiten

Eine noch wichtigere Einteilung der Abhängigkeiten erfolgt nach den Komponenten, zwischen denen diese bestehen. Es werden dadurch mehrere Kategorien identifiziert: Abhängigkeiten zwischen Ressourcen untereinander, zwischen Ressourcen und Diensten, zwischen Diensten und Subdiensten und Dienstfunktionalität-spezifische Abhängigkeiten. Hauptsächlich wird das MNM-Dienstmodell (Beschreibung in Kapitel 3.1.1) als Referenz für diese Einteilung der Abhängigkeiten benutzt.

Zunächst werden in Abschnitt 2.3.2.1 die Abhängigkieten zwischen den Ressourcen untereinander beschrieben. Es folgt dann eine Beschreibung der Abhängigkieten zwischen Ressourcen und Diensten in Abschnitt 2.3.2.2 und zwischen den Diensten und Subdiensten in Abschnitt 2.3.2.3. Anschließend werden in Abschnitt 2.3.2.4 die dienstfunktionalitätspezifischen Abhängigkeiten betrachtet.

Wenn mehrere Abhängigkeiten gleicher Art existieren, ist es möglich eine Skala der Wichtigkeit, des Abhängigkeitsgrads oder der Stärke der Abhängigkeit zu erstellen; so dass, bei einer hohen Anzahl von Abhängigkeiten, die unwichtigsten weggelassen werden können.

2.3.2.1 Abhängigkeiten zwischen Ressourcen

Hier werden die Beziehungen zwischen Ressourcen, also Abhängigkeiten auf der untersten Ebene der Diensthierarchie, und zwar auf der Ebene der Infrastruktur behandelt. Allerdings, wie im allgemeinen MNM-Dienstmodell ([GHHK 01] und [GHKR 01]) beschrieben, gehört zu den Ressourcen nicht nur die Infrastruktur, sondern auch Software, Mitarbeiter, Expertenwissen, d.h. alles, was zur Implementierung des Services beiträg. Darunter fallen beispielsweise die Abhängigkeit zwischen zwei Komponenten, wie Router und Switch. Diese physische Verbindung zwischen Router und Switch ist eine Abhängigkeit, weil beim Ausfall des Routers die Funktionalität des Switches ebenfalls beeinträchtigt sein kann. Die Tatsache, dass der Switch nicht erreichbar ist, kann zur Nichterreichbarkeit des Routers führen. In diesem Fall ist die Erreichbarkeit ein QoS-Parameter. Ein anderes Beispiel sind die regelmäßigen Software-Updates, die auf dem Router eingespielt werden.

Beispiel: In dem E-Mail-Szenario gilt das Beispiel der direkten Verbindung zwischen dem MailServer der LMU und dem Loadbalancer. Wenn diese physische Verbindung unterbrochen wird, dann ist

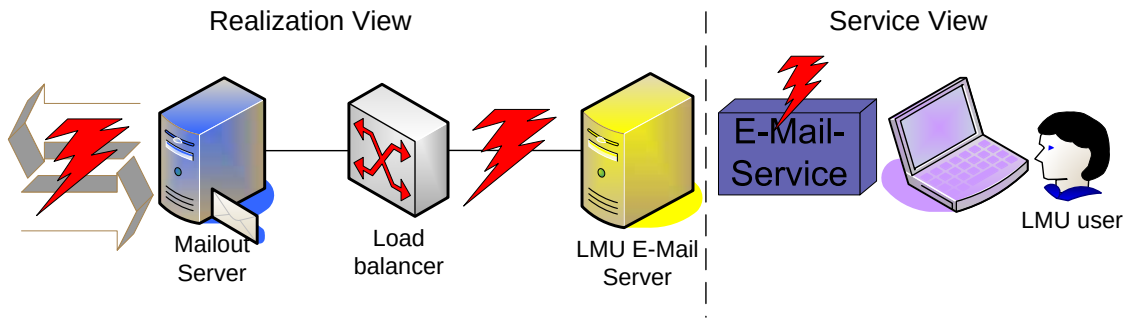

Abbildung 2.4: Abhängigkeiten zwischen den Ressourcen für den E-Mail-Dienst

jeglicher E-Mail-Verkehr bei der LMU ausgeschlossen (siehe Abbildung 2.4). Die Regel ist: dunkle Pfeile repräsentieren die Funktionalitäten, die die Kunden in Anspruch nehmen, und helle Pfeile die ausgehende bzw. eingehende Verbindung. Im Fall des Web-Hosting-Szenarios könnte die Verbindung zwischen dem Load-Balancer und den Web-Mail-Servern unterbrochen sein. Dadurch hat keines der Institute des Hochschulnetzes Zugriff auf seine Webseiten und diese sind von außen auch nicht erreichbar.

Das Erkennen dieser Abhängigkeiten im Vorfeld führt für die Kunden des LRZ., im Falle eines totalen Ausfalls des E-Mail-Dienstes oder des Web-Hosting-Dienstes, zu einer Steigerung der Effizienz bei der Fehlersuche

2.3.2.2 Abhängigkeiten zwischen Ressourcen und Diensten

Diese Art von Abhängigkeiten ist nicht nur Infrastruktur- bzw. Ressourcen-bedingt, sondern sie ist auch von der genauen Realisierung des Dienstes abhängig. Bei Abhängigkeiten zwischen Ressourcen wurden die Verbindungen auf der Ebene der Ressourcen betrachtet. In diesem Fall werden die Verbindungen zwischen der Ebene der Ressourcen und der der Dienste untersucht, wie z.B. Komponenten, die für die Realisierung eines Dienstes zuständig sind. Wenn eine Komponente ausfallen sollte, oder irgendeine Ressource sich im kritischen Bereich befindet, ist der Dienst gestört, er funktioniert nicht mehr standardmäßig, ist fehlerhaft.

Beispiel: Wenn der LMU-Mail-Server ausfällt, nicht unbedingt wegen des Ausfalls einer niedrigeren Komponente, gibt es keinen funktionierenden E-Mail-Dienst für die LMU. Wenn eine solche Situation auftreten sollte, würde man sehr schnell die Fehlerursache finden; wenn die Abhängigkeiten bekannt sind. Ein anderer Vorteil ergibt sich bei Bekanntheit der Abhängigkeiten für die Netzplanung. Hier kann man vorhersehen welche Dienste beim Ausfall dieses bestimmten Servers beeinträchtigt werden.

2.3.2.3 Abhängigkeiten zwischen Diensten und Subdiensten

Einige Dienste brauchen zur Realisierung Subdienste. Diese können sowohl statische Abhängigkeiten (siehe Abschnitt 2.3.3.1) als auch dynamische Abhängigkeiten (siehe Abschnitt 2.3.3.2) für Dienste auf einem höheren Level darstellen. Es gibt immer eine Menge von Diensten, die ständig dazu beitragen dass der Dienst, der jetzt und hier angeboten wird, reibungslos abläuft (statisch). Es gibt aber

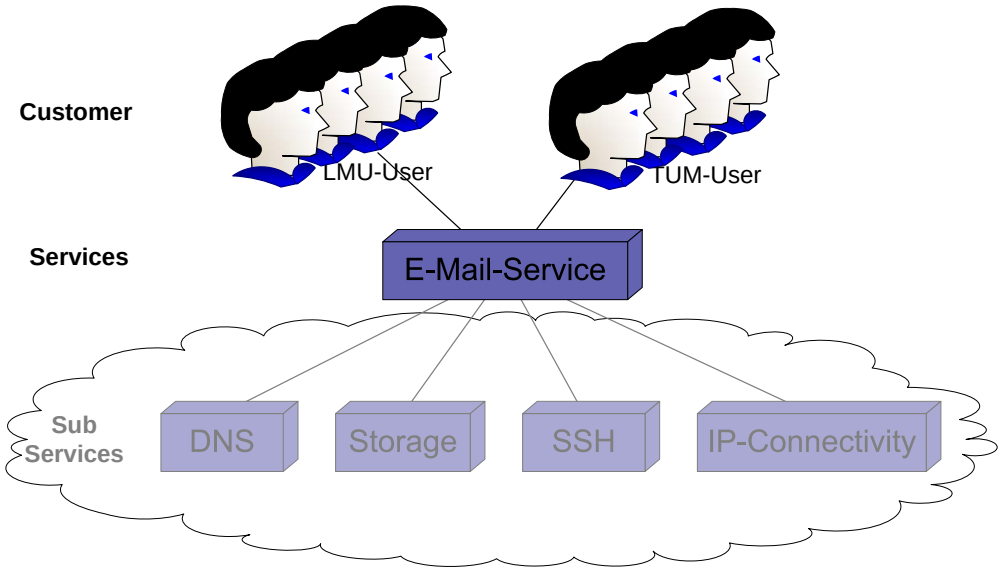

Abbildung 2.5: Transparenzen im E-Mail-Dienst

auch den Fall, in dem ein Dienst mehrere ähnliche (oder identische) Subdienste dynamisch auswählen kann.

Diese Subdienste sind, je nach Situation, für den Kunden sichtbar (intransparent) oder unsichtbar (transparent). Das Problem der Transparenz in der Diensterbringung wurde in [GHHK 02] ausführlich beschrieben. Wenn die User transparent für ihre Anbieter sind, weiß der Anbieter nicht, wer den bestimmten Dienst benutzt. Ein Nutzer ist hingegen intransparent für seinen Provider, wenn dieser die Personalisierung des Dienstes für seine Nutzer verlangt. Das ist insbesondere dann der Fall , wenn der Kunde Provider für weitere End-Kunden ist, und er den Dienst weiterverkauft [GaKe 01]. Subprovider können ebenfalls transparent oder intransparent für ihren Kunden sein. Allgemein können die Rollen, so wie es in [GHHK 01] definiert wird, auf der entgegengesetzten Seite sowohl sichtbar als auch unsichtbar sein. Das bedeutet, dass ein intransparenter Subdienst von der Customersseite des Hauptdienstes aus sichtbar ist.

Beispiel: Für die zwei Beispielszenarien werden DNS, Storage-, SSH- und Connectivity-Dienst als grundlegende Dienste angeboten, um überhaupt E-Mail- und Web-Hosting-Dienst realisieren zu können. Diese sind gegenüber dem Nutzer transparent (unsichtbar) wie in Abbildung 2.5 im Vergleich zu Abb. 2.2 dargestellt.

Ein Beispiel für einen intransparenten (sichtbaren) Subdienst gibt es im Fall des Web-Hosting-Dienstes, wenn bestimmte Institute den virtuellen Webserver benutzen. Denn dann brauchen sie auch den IP-Dienst, der vom LRZ zur Verfügung gestellt wird. Das heißt, das LRZ benutzt den IP-Dienst, dieser ist für die Kunden (Institute des MWNs) sichtbar.

2.3.2.4 Dienstfunktionalität-spezifische Abhängigkeiten

Diese Art von Abhängigkeiten bezieht sich auf einzelne, vom Dienst angebotene Funktionalitäten und nicht auf den Dienst insgesamt. Abhängigkeiten zu einzelnen Dienstfunktionalitäten, anstatt ganzen

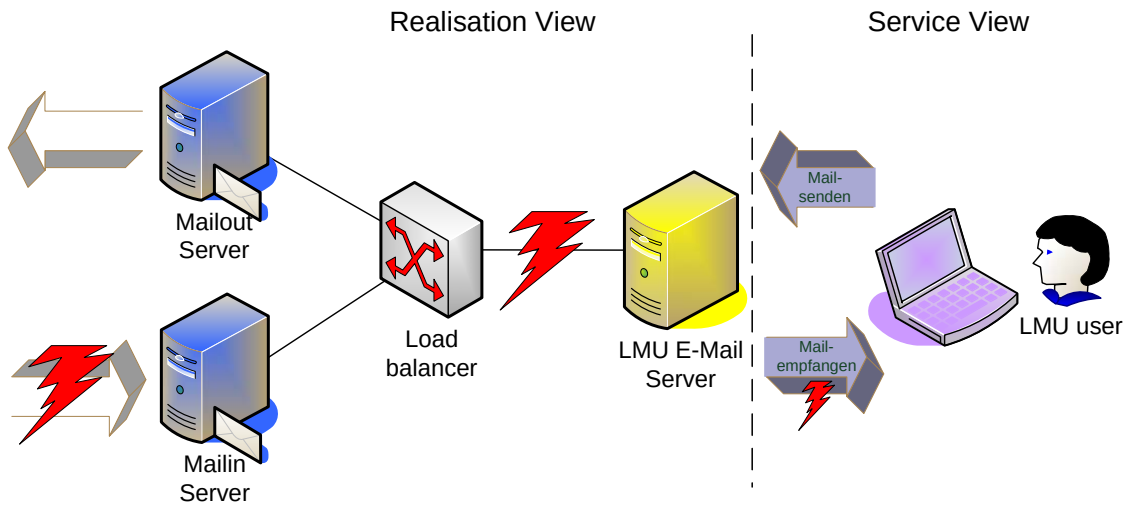

Abbildung 2.6: Die Dienstfunktionalitäten des E-Mail-Dienstes

Diensten, stellen eigentlich nichts Neues gegenüber den bisherigen Abhängigkeiten (zwischen Ressourcen und/oder Diensten) dar, sondern sie sind nur eine Verfeinerung davon. Das bedeutet, dass eine Störung in der Infrastruktur, nicht den Dienst als Ganzes beinträchtig, sondern nur bestimmte Funktionalitäten davon.

Beispiel: Neben den obigen spezifischen Beispielen wird noch als allgemeines Beispiel für Dienstfunktionalitäten folgendes geschildert:

- Der E-Mail-Dienst (Abb. 2.6) besitzt die zwei Nutzungsfunktionalitäten Mailsenden und Mailempfangen (als dunkle Pfeile repräsentiert), die die Kunden in Anspruch nehmen können. Es könnte daher sein, dass es bei einer Störung der Verbindung zwischen dem LMU E-Mail-Server und dem Load Balancer nicht zu einem Totalausfall des E-Mail-Dienstes kommt, sondern nur zu einer Störung der Mailsenden-Funktionalität. Die User der LMU können daher ihre E-Mails lesen, weil die Mail-Empfangen Funktionalität funktionsfähig ist, aber nicht versenden.
- Für den Web-Hosting-Dienst gibt es grundsätzlich auch mehrere Funktionalitäten: Erstellen, Ändern, Löschen und natürlich auch Abrufen von statischen und dynamischen Webseiten. Es kann daher die Situation auftreten, dass die Seiten geändert werden können, aber bei Aufruf einer dynamischen Seite kommt eine Fehlermeldung.

Es besteht die Möglichkeit diese grundlegenden Funktionalitäten weiter zu verfeinern. Das bedeutet, dass die obigen allgemeinen Fälle der Abhängigkeiten zwischen Ressourcen und Diensten (siehe Abschnitt 2.3.2.2) bzw. zwischen (Sub)Diensten und Diensten (siehe Abschnitt 2.3.2.3) jetzt verfeinert werden. Der Pfeil **A→B** bedeutet, dass B von A abhängig ist, oder A eine notwendige Voraussetzung für B ist.

Ressource → Dienstfunktionalität Die Ressource mailout-Server ist z.B. für die Dienstfunktionalität ,,Mailsenden“ nötig.

(Sub)Dienst → Dienstfunktionalität Die Webmailfunktionalität z.B. hängt vom E-Mail-Dienst ab; die mysql-Administration/Management Funktionalität hängt vom Webdienst ab.

(Sub)Dienstfunktionalität → Dienstfunktionalität Ein Beispiel ist die Abhängigkeit zwischen

der speziellen Apache-Spezialkonfiguration des Webdienstes, als Subdienstfunktionalität, und der phpmyadmin Funktionalität, als Funktionalität des mysql-Dienstes.

(Sub)Dienstfunktionalität → Dienst Die Abhängigkeit zwischen der phpmyadmin Funktionalität, als spezielle Managementfunktionalität des mysql-Dienstes und dem mysql Dienst oder die Abhängigkeit zwischen dem Webdienst und dem mysql-Dienst, benutzt nur für bestimmte Funktionalitäten wie z.B dynamische Webseiten.

2.3.3 Die Dynamik der Abhängigkeiten

Nach ihrem Änderungsgrad kann man zwei Arten von Abhängigkeiten grundlegend unterscheiden: statische und dynamische Abhängigkeiten. Die statischen Abhängigkeiten werden im Abschnitt 2.3.3.1 beschrieben gefolgt von der dynamischen Abhängigkeiten im Abschnitt 2.3.3.2.

2.3.3.1 Statische Abhängigkeiten

Die statischen Abhängigkeiten sind diejenigen Abhängigkeiten, die unter allen Umständen vorhanden sind und ohne deren Existenz der Dienst nicht funktioniert. Einige Dienste brauchen Subdienste, um die Nutzung und das Management des Dienstes bereitzustellen. Das heißt, Dienst A braucht immer die Unterdienste A_1- A_k.

Beispiel Der E-Mail- und Web-Hosting-Dienst benötigen immer den DNS-Dienst.

Die Mailübertragung basiert auf DNS. Im DNS gibt es verschiedene Arten von Einträgen – sogenannte Resource Records – bei denen einem Domänennamen bestimmte Informationen zugeordnet sind. Bei der Abfrage des DNS wird der Domänenname und der Typ des Eintrags angegeben und die zugehörige Information wird geliefert. Unter einem Domänennamen kann man verschiedene Dinge verstehen. Genaueres hierüber im Kapitel 5 (Modellierung des Szenarios).

Der Web-Hosting-Dienst benötigt auch notwendigerweise DNS, denn beim LRZ werden weltweit eindeutige Namen für die virtuellen Webserver, die den Instituten und Einrichtungen des MWN zur Verfügung gestellt werden, benötigt.

2.3.3.2 Dynamische Abhängigkeiten

Die dynamischen Abhängigkeiten unterscheiden sich von den statischen dadurch, dass sie die, sich ändernde, Zuweisung von Ressourcen und Subdiensten des Dienstes beschreiben. Im Falle einer Störung, besteht bei der Fehlersuche, der Vorteil darauß, anhand des Modells, nur die tatsächlich auftretenden Abhängigkeiten zu kennen, und nur diesen zu folgen.

Beispiel Beim LRZ ist der E-Mail-Server durch einen Load Balancer auf zwei Mailrelays verteilt. Es stellt sich die Frage, wie hoch die Auslastung dieser Relays ist? Wann und unter welchen Umständen wird der eine und wann der andere benutzt?

Der Load-Balancer (Level-4/7-Switches) überwacht die 12 Rechner, die für die Realisierung des Web-Hosting-Dienstes angeschlossen werden. Wenn sich die Webseite eines Kunden ändert, ist man nie sicher, auf welchem der sechs redundanten Servern man arbeitet.

2.3.4 Redundanzen in der Infrastruktur

Die Infrastruktur muss aus Fehlertoleranzgründen redundant sein. Also sollten alle „anfälligen“ Teile der Infrastruktur redundant gesichert werden, so dass, beim Ausfall einer der Komponenten, die redundante Komponente die Kontrolle, bis zur Wiederherstellung der Funktionalität der ersten Komponente, übernimmt.

Isolierte Abhängigkeiten sind Beziehungen zwischen zwei Komponenten oder Diensten die eigenständig sind, wie bespielsweise die Abhängigkiet zwischen dem mailout-Server und dem dns-Service. Es gibt aber auch der Fall wenn Abhängigkeiten zusammengefasst betrachtet werden müssen, wie z.B die drei Abhängigkieten zwischen dem mailout-Server und jedem der drei dns-Servern (dns1, dns2, dns3). Diese sind die Verbundabhängigkeiten. Die **Verbundabhängigkeiten** sind am meisten in redundanten Systemen oder Diensten present.

Beispiel: Offensichtlich sind bei der Realisierung des E-Mail-Dienstes als auch des Web-Hosting-Dienstes, wie in den Abbildungen 2.2 und 2.3 dargestellt wird, mehrere Server redundant miteinander verbunden. Das erfolgt aus Sicherheitsgründen und/oder für die Lastverteilung. Falls sie beim Ausfall des Webservers nicht redundant verbunden sind, ist die Erreichbarkeit der hier ausgelagerten Seiten nicht mehr möglich. Wenn sie hingegen redundant verbunden sind, wird beim Ausfall des Webservers diejenige redundante Komponente eingeschaltet, welche die identische Konfiguration besitzt, so dass der Nutzer nichts davon merkt, was „hinter der Bühne“ stattfindet. Das Ziel, dass die Nutzer mit so wenig Störungen wie möglich arbeiten, wird hiermit erreicht.

2.4 Dimensionen der Abhängigkeiten

Zusammenfassend werden in Abbildung 2.7 die Dimensionen dargestellt, die zur Betrachtung der Abhängigkeiten wichtig sind. Die Idee dieser Darstellung wurde aus [HAN 99] übernommen; die Dimensionen sind jedoch für das Abhängigkeitsmodell angepasst worden. Es werden folgende sieben Dimensionen dargestellt: Dynamik, Formalisierungsgrad, Abhängigkeitstyp, Dienstfunktionalitätsaufteilung, Redundanzen, Lebenszyklus und Ausprägung.

Die Dimension **Dienstfunktionalitätsaufteilung** bezieht sich auf die Tatsache dass im Falle jedes Dienstes Abhängigkeiten existieren, die den Dienst insgesamt betreffen, oder nur einzelne der verschiedenen Funktionalitäten des Dienstes (siehe auch Abschnitt 2.3.1).

Die Dimension **Abhängigkeitstyp** unterscheidet zwischen den Abhängigkeiten, die auf Ressourcenebene, auf der Dienstebene als auch zwischen den zwei Ebenen vorhanden sind (siehe Abschnitt 2.3.2).

Der **Formalisierungsgrad** zeigt wie stark das Abhängigkeitskonzept in einem Ansatz formalisiert ist. Man unterscheidet wie folgt drei Fälle: Erstens kann überhaupt keine Formalisierung existieren, zweitens sind Abhängigkeiten nur textuell dargestellt (auch Pseudocode ist möglich) und drittens als formales Modell beschrieben. Diese Arbeit konzentriert sich nur auf das formale Modell.

Die **Ausprägung** bezieht sich auf die Art der Vorkommnisse dieser Abhängigkeiten. Sie können eingeteilt werden in einerseits organisatorische (z.B. zwischen den verschiedenen Abteilungen) und andererseits funktionale (nur die tatsächliche Funktion des Dienstes und alles, was diese betrifft)

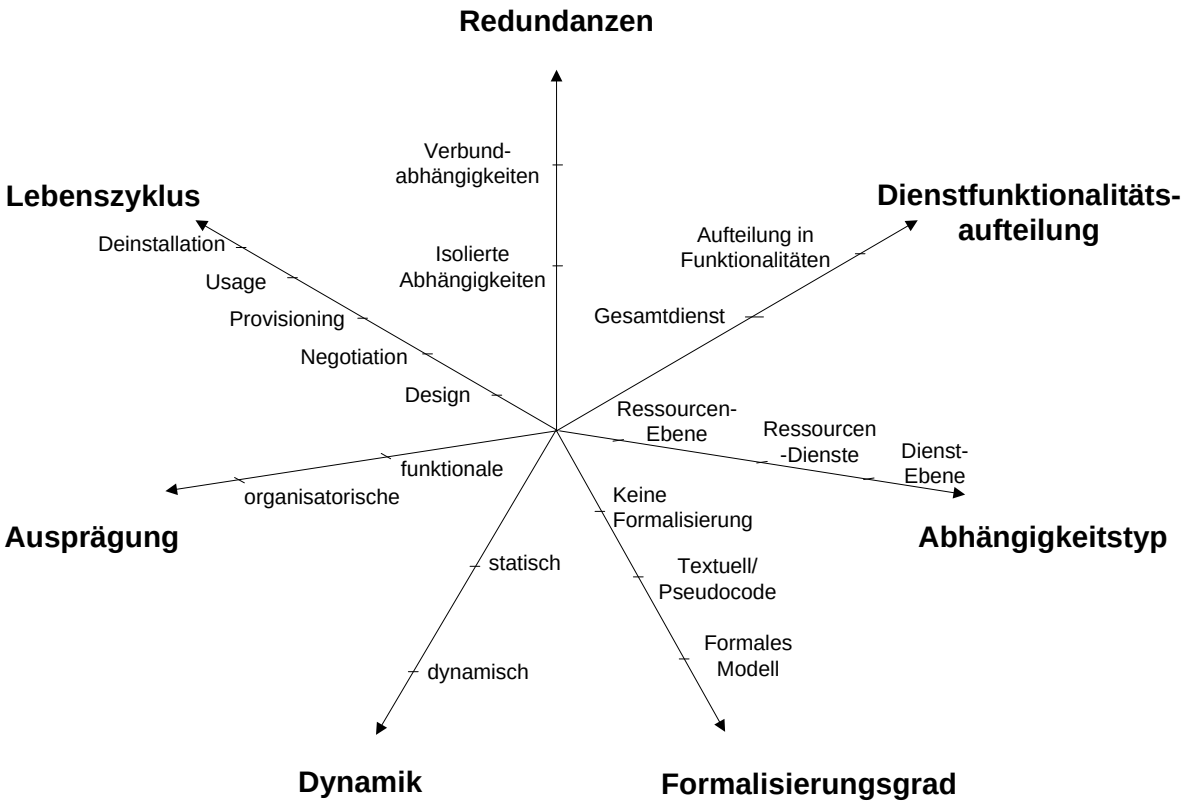

Abbildung 2.7: Dimensionen der Abhängigkeiten

Abhängigkeiten. In dieser Arbeit werden allerdings nur die funktionalen Abhängigkeiten zur Modellierung in Betracht gezogen.

Die Dimension **Dynamik** (siehe Abschnitt 2.3.3) bezieht sich auf den Änderungsgrad der Abhängigkeiten. Dabei unterscheidet man zwischen statischen und dynamischen Abhängigkeiten. Die statischen Abhängigkeiten bestehen unter allen Umständen. Ohne deren Existenz funktioniert der Dienst nicht. Die dynamischen Abhängigkeiten unterscheiden sich von den statischen dadurch, dass sie die sich ändernde Zuweisung von Ressourcen und Subdiensten des Dienstes beschreiben. Der Vorteil hierbei ist, dass die Fehlersuche, im Falle eines Modells das die dynamischen Abhängigkeiten miteinbezieht, viel schneller werden kann

Die Dimension **Lebenszyklus** wurde bereits im Abschnitt 2.3.1 detaillierter beschrieben. Sie zeigt die Wichtigkeit der unterschiedlichen Phasen des Dienstes für die Abhängigkeiten. Es können Abhängigkeiten nur während einer Phase entstehen. Es können allerdings auch Abhängigkeiten während des gesamten Lebenszyklus bestehen.

Aus dem Gesichtspunkt der **Redundanzen** (siehe Abschnitt 2.3.4) wird zwischen isolierten Abhängigkeiten und Verbundabhängigkeiten unterschieden. **Isolierte Abhängigkeiten** entstehen, wenn zwischen zwei Komponenten oder Diensten eine Abhängigkeitsbeziehung existiert und diese zusätzlich von anderen Abhängigkeiten isoliert ist. Das heißt, dass sich die Auswirkungen, bei einem Ausfall, nicht ausbreiten werden; wie wenn die bestehende Abhängigkeit mit anderen Abhängigkeiten in „Verbindung“ steht. Diese sind dann die **Verbundabhängigkeiten**. Das bedeutet, durch Redundanzen erhält man Verbundabhängigkeiten.

2.5 Aufstellung des Anforderungskatalogs

Bei der Erstellung des Anforderungskatalogs werden der größte Teil der benötigten Informationen aus Kapitel 2.3 zusammengefasst. Zusätzlich kommen noch andere für den Dienst wichtige Informationen, wie allgemeine Informationen über der Dienst, hinzu. Das wird in Form einer Liste dargestellt.

1. **Dienstbezogene Anforderungen**
   a) Technologieunabhängigkeit
   b) Allgemeinheit des Dienstbegriffes
   c) Lebenszyklus
   d) Dienstfunktionalitäten
   e) Dienstparameter
      - Dienstanbieter (Provider)
      - Kunde (Customer)
      - Nutzer (User)
   f) Erweiterbarkeit
   g) Anwendbarkeit
   h) Bereitstellung der Dienstgüte
2. **Dynamik der Abhängigkeiten**
   a) Statische Abhängigkeiten
   b) Dynamische Abhängigkeiten
3. **Arten von Abhängigkeiten**
   a) Transparenzen
   b) Ressource ➔ Ressource Abhängigkeiten
   c) Ressource ➔ Dienst Abhängigkeiten
   d) (Sub)Dienst ➔ Dienst Abhängigkeiten
   e) Abhängigkeiten übertragen auf Dienstfunktionalitäten
      Daraus ergeben sich mehrere Kategorien, die in Betracht gezogen werden müssen
      i. Ressource ➔ Dienstfunktionalität
      ii. (Sub)Dienst ➔ Dienstfunktionalität
      iii. (Sub)Dienstfunktionalität ➔ Dienstfunktionalität
      iv. (Sub)Dienstfunktionalität ➔ Dienst
4. **Redundanzen** Redundanzen der Infrastruktur oder Ressourcen im Allgemeinen, aber auch Dienstredundanzen.

2.6 Zusammenfassung

In diesem Kapitel wurden die Anforderungen an das Modell analysiert. Bei der Erstellung des Anforderungskatalogs wurden die Anforderungen nach dienstbezogenen und abhängigkeitsbezogenen Kriterien eingeteilt. Obwohl es sich hier um ein Abhängigkeitsmodell handelt, wurden diese Kriterien so ausgewählt dass auch die allgemeinen Anforderungen an ein Informationsmodell in Betracht gezogen werden können. Nichtsdestoweniger ist der wichtigste Teil der Anforderungsanalyse der Abhängigkeitsbezogene, denn dieser befassen sich mit den Abhängigkeiten zwischen den unterschiedlichen Komponenten, die zur Realisierung des Dienstes beitragen. Aufgrund dessen wurden die Dimensionen der Abhängigkeiten in einem Dimensionsstern dargestellt (siehe Abbildung 2.7). Am Ende des Kapitels wurden die Anforderungen zu einem Anforderungskatalog zusammengefasst. Dieser Katalog wird im Folgenden auf den State-of-the-Art (Kapitel 3) angewandt.

Kapitel 3

Vorhandene Ansätze

Inhaltsverzeichnis

Dieses Kapitel befasst sich mit den existierenden Arbeiten im Bereich des Fehlermanagements, der Dienstorientierung und der Ereigniskorrelation, die für die Abhängigkeitsmodellierung wichtig sind. Die Analyse dieser vorhandenen Ansätze ist für die Modellierung der Abhängigkeiten von großer Wichtigkeit, weil es gemäß des Anforderungskatalogs eine Bewertung durchgeführt wird. So können, nach dieser Bewertung, die für die Modellierung der Abhängigkeiten wichtigsten Konzepte, wiederbenutzt werden. Zuerst, im Abschnitt 3.1, werden ein paar Forschungsarbeiten auf dem Gebiet vorgestellt. In Abschnitt 3.2 werden einige Ansätze, die als bekannte Standards im Bereich des IT-Service Managements (ITIL, eTOM) bzw. Informationsmodelle (CIM, SID) gelten, dargestellt. Zuletzt werden, im Abschnitt 3.3, einige Tools, die für das Fehlermanagement in der Praxis benutzt werden, beschrieben.

3.1 Forschungsarbeiten

In diesem Abschnitt werden mehrere Forschungsarbeiten im Bereich des Fehlermanagements, der Ereigniskorrelation im Bezug auf Abhängigkeiten beschrieben. Als erstes wird im Abschnitt 3.1.1 das MNM-Dienstmodell allgemein und anhand der zwei Szenarien detailliert beschrieben. Abschnitt 3.1.2 befasst sich mit den Abhängigkeitshierarchien [KeKa 01], gefolgt von Abhängigkeitsgraphen [Grus 99] in Abschnitt 3.1.3, Abhängigkeitsgraden [BKH 01] in Abschnitt 3.1.4 und der Verfügbarkeit [Kais 99] in Abschnitt 3.1.4. Anschließend werden in Abschnitt 3.1.6 die Abhängigkeiten für die Internet-Dienst-Provider vorgestellt.

3.1.1 Das MNM-Dienstmodell

Wegen ihrer zunehmenden Komplexität werden große IT-Dienste gewöhnlich nicht durch einen einzelnen Dienstanbieter erbracht. Stattdessen bestehen sie aus den, voneinander abhängigen und überlagerten Dienstleistungen, die von den unterschiedlichen Anbietern betrieben werden. Diese überlagerten Dienstleistungen bauen sogenannte Dienstketten (service chains) oder -hierarchien (service hierarchies) auf. Um den Dienst richtig laufen zu lassen und beizubehalten muss das Managementsystem der Anbieter passend integriert sein. Deswegen ist ein allgemeines Verständnis des Dienstes auf den verschiedenen Niveaus erforderlich. Wie schon kurz in Abschnitt 2.1 erwähnt, liefert das MNM-Dienstmodell [GHKR 01] ein generisches Modell, das allgemein erforderliche dienstbezogene Bezeichnungen, Konzepte und Strukturierungsregeln in einer generellen und eindeutigen Weise definiert.

3.1.1.1 Das Basis-Modell

Wie in [GHKR 01] beschrieben, ist es schwierig, den Begriff **Dienst** so allgemein wie möglich, ohne Einschränkungen auf irgendwelche Szenarien zu definieren. In diesem ersten Abschnitt wird die Grundlage dieses Modelles bzw. die Vorgehensweise für die Realisierung des MNM-Dienstmodells beschrieben. Das Basis-Modell wurde hergeleitet durch :

1. Top Down-Analyse generischer Dienstumgebungen Eine wesentliche Charakteristik jedes Dienstes ist, dass er immer die zwei ,,Akteure“ Anbieter und Kunden impliziert. Die zwei Seiten, technisch auch **customer side** (Kundenseite) und **provider side** (Anbieterseite) genannt [GHHK 01], interagieren zur Realisierung des Dienstes. Daraus kann man Schlüsse über die Dienstfunktionalitäten ableiten, ohne dass die tatsächliche Dienstimplementierung betrachtet werden muß.

2. Betrachtung des gesamten Lebenszyklus des Dienstes

   Der Lebenszyklus des Dienstes ist ein zusätzlicher Aspekt, welches das MNM-Dienstmodell berücksichtigt. In der Abbildung 3.1 ist die Einteilung des Lebenszyklus eines Dienstes [Dreo 02] in fünf Phasen dargestellt. Im Folgenden ist eine kurze Darstellung dessen realisiert. Die *design*-Phase beinhaltet die Spezifikation der gewünschten Funktionalität und möglicher QoS-Parameter.

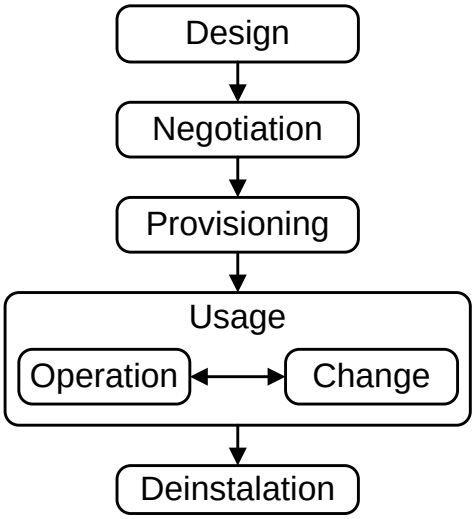

Abbildung 3.1: Service Life Cycle

Während der *negotiation*-Phase wird zwischen Kunde und Anbieter eine Vereinbarung bzgl. QoS-Parameter, Tarife, Strafen, Dienstmanagement getroffen. Die **Dienstvereinbarung** (Service Agreement) ist eine Beschreibung der zukunftigen Interaktionen, die zwischen der Kunden- und Anbieterseite stattfinden werden.

In der *provisioning*-Phase werden seitens des Anbieters die Implementierung, Konfiguration und Test des Dienstes und sein Management realisiert. Diese Phase wird mit einer **Annahmeerklärung** (Statement of Acceptance) seitens des Kunden abgeschlossen.

Die *usage*-Phase ist für den Kunden die wichtigste Phase, da in dieser die tatsächliche Nutzung des Dienstes stattfindet. Sie beinhaltet zwei Unterphasen **operation**, während dessen alle Maßnahmen um den Dienst operational zu halten realisiert werden und **change**, in der alle Änderungen des Dienstes oder seiner Implementierung, die erforderlich sind, durchgeführt werden.

3. Funktionale Klassifizierung von Interaktionen Im vorherigen Abschnitt wurden einige Interaktionen zwischen Kunde und Provider erwähnt. Es ist aber schwierig, über jede im System stattfindende Interaktion, Buch zu führen. Daher ist die Notwendigkeit einer Abstraktion entstanden. Infolge dessen existieren zwei Interaktionsklassen **Nutzung** (Usage) und **Management**. Diese werden dann im Modell in der Form der Funktionalitäten eines Dienstes wiederspiegelt.

4. Identifikation von Objekten und Rollen im Dienstumfeld Entsprechend der zwei oben beschriebenen Interaktionsklassen werden sowohl auf der Kundenseite

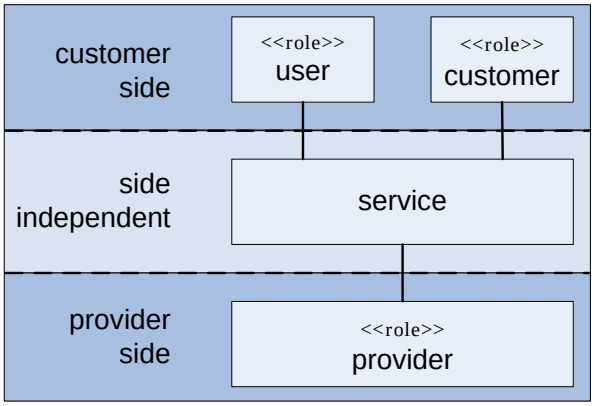

Abbildung 3.2: Basis MNM-Dienstmodell [GHHK 01]

(Nutzer und Kunde) als auch auf der Anbieterseite (Anbieter) Rollen (siehe Abbildung 3.2) zugeschrieben. Auf der Kundenseite (customer side) kann man zwischen zwei Hauptrollen differenzieren: die des *Nutzers* (user), der den Dienst eigentlich benutzt, und die des *Kunden* (customer), der für die Wartung des Dienstes zuständig ist, und daher alle Managementaktivitäten auf der Kundenseite ausführt. Auf der Anbieterseite (provider side) können die Nutzungs- und Managementaktivitäten zur Realisierung interner Prozesse nicht komplett getrennt werden. Daher ist auf der Anbieterseite nur die Rolle des *Anbieters* (provider) zu finden.

3.1.1.2 Anwendung des MNM-Basismodells auf das Szenario

Das oben beschriebene Dienstmodell wird in den folgenden zwei Abschnitten auf die bestehenden Szenarien Web Hosting- und E-Mail-Dienst angewandt.

Der E-Mail-Dienst Für den E-Mail-Dienst identifiziert man drei Rollen: E-Mail-Nutzer an der LMU (allgemeinen `users`), LMU als Kunde (allgemein `customer`), der den Dienst in Anspruch nimmt und das LRZ , das den Dienst anbietet (der `provider` im allgemeinen Modell). Diese Rollen sind wie in Abbildung 3.3 miteinander verbunden: LRZ als Dienstanbieter (auf der Anbieterseite) bietet seinem Kunden LMU für dessen Nutzer (auf der Kundenseite) den E-Mail-Dienst (seitenunabhängig). Als E-Mail-Nutzer werden im allgemeinen die Studenten, Dozenten, Mitarbeiter der Administration usw. betrachtet. Diese können Mails senden und empfangen. Der Kunde LMU vereinbart mit dem LRZ als Provider, wie der E-Mail-Dienst aussehen muß.

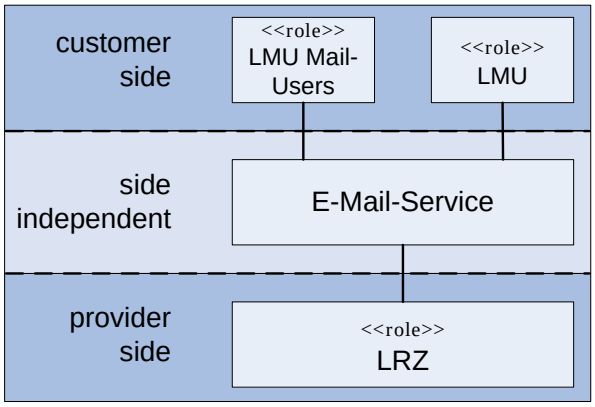

Abbildung 3.3: MNM-Basismodell für den E-Mail-Dienst

Der Web Hosting-Dienst Für den Web Hosting-Dienst werden ebenfalls drei Rollen identifiziert: Web Hosting-Nutzer an der TUM (in allgemeinen Fall `users`), TUM als Kunde (allgemein `customer`), der den Dienst in Anspruch nimmt, und das LRZ, das den Dienst anbietet (der `provider` im allgemeinen Modell). Diese Rollen werden in Abbildung 3.4 mit den entsprechenden Verbindungen repräsentiert. Das LRZ als Dienstanbieter (auf der Anbieterseite) bietet seinem Kunden TUM für dessen Nutzer (auf der Kundenseite) den Web Hosting-Dienst (seitenunabhängig) an. Die TUM-Nutzer sind die Studenten, Dozenten, usw., aber auch die Lehrstühle, Institute und Einrichtungen im allgemeinen. In Prinzip kann jeder sich die Webseiten ansehen und daher auch als User betrachtet werden. Durch die Einrichtung eines Passwort-geschützten Bereiches kann man die Nutzung auf Personen/Personengruppen einschränken. Wie in den nächsten Kapiteln vorgestellt, gibt es bei dem Web Hosting-Dienst zwei unterschiedliche Arten von Nutzern. Diejenigen, die einen virtuellen WWW-Server am LRZ besitzen können (Institut oder Einrichtung der Münchner Hochschuleinrichtung) und diejenigen, die dafür keine Berechtigung haben. Studenten können z.B. nur am entsprechenden Studentenserver eigene Webseiten ablegen.

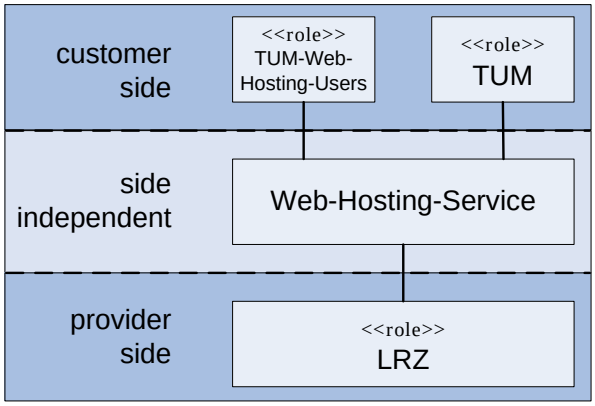

Abbildung 3.4: MNM-Basismodell für den Web Hosting-Dienst

3.1.1.3 Rekursive Anwendung des MNM-Dienstmodells

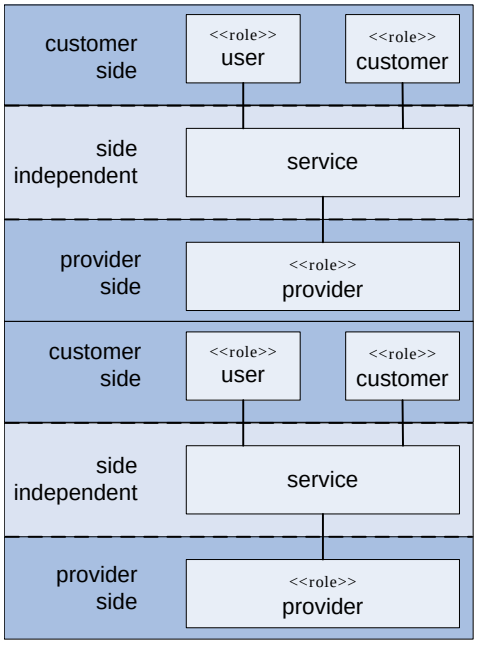

Abbildung 3.5: Verkettung der Dienste

Das MNM-Dienstmodell unterstützt Dienstketten oder -hierarchien [GHHK 01]. Eine Diensthierarchie (siehe Abbildung 3.5) entsteht dann, wenn ein Anbieter Dienste von einem anderen Anbieter in Anspruch nimmt, dann agiert der erste wie ein Kunde für den zweiten. Das heißt, dass der Bereich des Anbieters die Tätigkeiten der Rolle Nutzer/Kunde als auch die der Providerrolle umfasst. Man kann unter Umständen die bereits modellierten Assoziationen zwischen Kunde und Anbieter zur Modellierung der Assoziationen zwischen Providern und Unter-Providern (Sub-Provider) wiederbenutzen. Auf diese Art und Weise kann man Providerhierarchien oder -ketten (Anbieter, Unter-Anbieter, Unter-Unter-Anbieter usw.) im Modell zusammen repräsentieren. Der Anbieterbereich wird durch die Rollen des Kundenbereichs erweitert. Dadurch erhält man ein erweitertes Modell des Anbieters mit Rollen aus dem Kundenbereich, als auch aus dem Providerbereich, die sehr wichtig für die interne Realisierung des Dienstes sind. Für den Provider werden dann die drei Rollen in einer speziellen Sicht (Realisierungs-Sicht) des Modells repräsentiert. Dies wird im Abschnitt 3.1.1.4 ausführlicher beschrieben.

Anwendung der Rekursion auf das Szenario In diesem Abschnitt wird anhand des Szenarios die rekursive Anwendung des MNM-Dienstmodells erklärt.

Dieses bietet sich in diesen Fall an, da das LRZ auch Kunde für andere „Sub“-Provider ist. Die Abbildungen 3.3 und 3.4 werden durch Konstruktion einer Diensthierarchie oder Dienstkette (siehe Abbild.3.6 und 3.7) ergänzt.

Das LRZ bietet seine E-Mail- und Web Hosting-Dienste den Kunden LMU bzw. TUM und deren Nutzern an (in Abbildung 3.6 E-Mail-Nutzer der LMU und in der Abbildung 3.7 TUM-Nutzer des Web Hosting-Dienstes). Andererseits muß das LRZ von einem anderen Provider, z.B. M”Net, den IP-Connectivity-Dienst in Anspruch nehmen. Dadurch wirkt das LRZ gegenüber M”Net bzw. des T-Systems als Nutzer der IP-Connectivity als auch als Kunde, das für das Management zuständig ist. Es werden zwei ganz einfache Dienstketten betrachtet. Diese können beliebig erweitert werden je nachdem, wieviele Unterprovider und/oder wieviele Dienste angeboten werden usw.

3.1.1.4 Die Komponenten des Modells

Um die Definition zu vereinfachen, wurde der Dienst durch die Existenz der Rollen User, Customer, Provider als auch durch ihre Beziehung zueinander präzise definiert. Das MNM-Modell beinhaltet die Dienst-Sicht (**Service View**) und die Realisierungs-Sicht (**Realization View**). Die Abbildung 3.8 repräsentiert die Dienst-Sicht (Service View) des MNM-Dienstmodells, die eigentlich die Kunde-Anbieter Beziehung (Customer-Provider Relationship) beschreibt. Der Dienst muß auf Seiten des

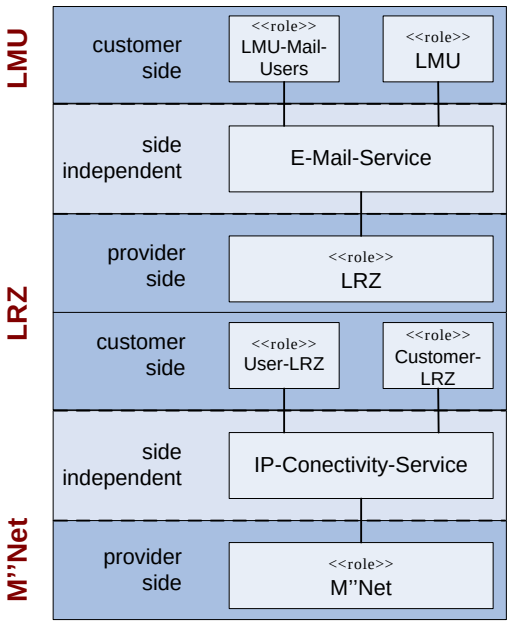

Abbildung 3.6: Die rekursive Anwendung des MNM-Dienstmodells für den E-Mail-Dienst

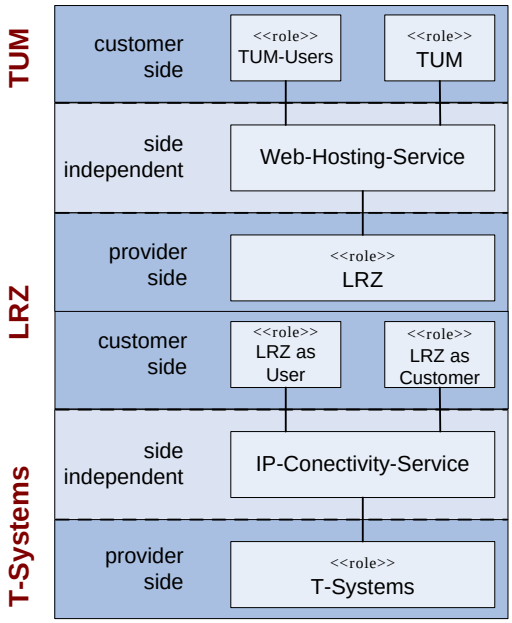

Abbildung 3.7: Die rekursive Anwendung des MNM-Dienstmodells für den Web Hosting-Dienst

Kunden und des Anbieters gleich verstanden werden. Daher werden in dieser Sicht die Dienstfunktionalitäten Nutzungsfunktionalität (Usage Functionality) und Managementfunktionalität, entsprechend der Interaktionsklassen (Nutzung und Management) beschrieben, als auch die QoS-Parameter, die von den beiden erfüllt werden müssen. Eine ausführlichere Beschreibung dieser Sicht wird in Abschnitt 3.1.1.4 realisiert.

Auf der anderen Seite steht die Realisierungs-Sicht, die die Anbieter-interne Realisierung des Dienstes repräsentiert. Die Realisierungs-Sicht (Abbildung 3.9) kann als Basis für die Implementierung eines Dienstes dienen, aber auch für die Modellierung der durch Verkettung entstandenen Diensthierarchien (siehe Kapitel 3.1.1.3). Man differenziert hier grob zwischen den zwei Klassen Dienstimplementierung (Service Implementation) und Dienstmanagement (Service Management) und den drei Rollen Kunde, User und Provider. Das Zusammenspiel dieser Komponeten wird im nächsten Abschnitt ausführlich beschrieben.

Die Dienst-Sicht (*Service View*) Wie oben schon beschrieben, muss der Dienst auf Seiten des Kunden und des Anbieters gleich verstanden werden. Das MNM-Modell richtet sich nach dem Dienstorientierungsansatz, der besagt, dass die Implementierung des Dienstes unabhängig von der Kundenseite ist. Weiterhin müssen in einer kundenorientierten Welt die seitenunabhängigen Informationen vom Gesichtspunkt des Kunden dargestellt werden. Das heißt, die Informationen, die vom Provider zur Verfügung gestellt werden, müssen vom Kunden „verstanden“ werden. Eine detailierte Darstellung der Dienst-Sicht wird im Abbildung 3.10 vorgestellt. Zur Beschreibung der Dienst-Sicht müssen die drei Seiten (Bereiche) [HLN 01], die für die Erbringung des Dienstes mitwirken, beschrieben werden.

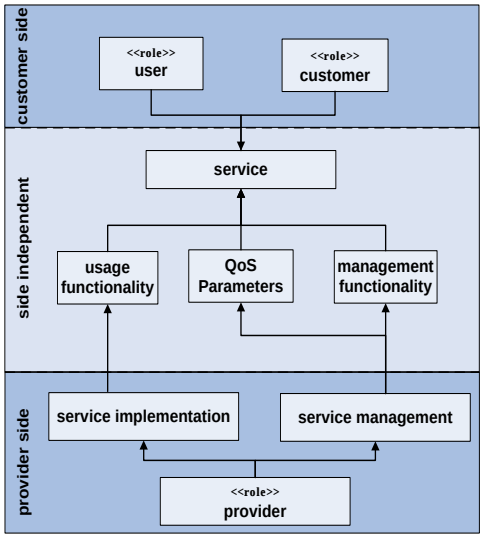

Abbildung 3.8: Der Service-View des MNM-Dienstmodells

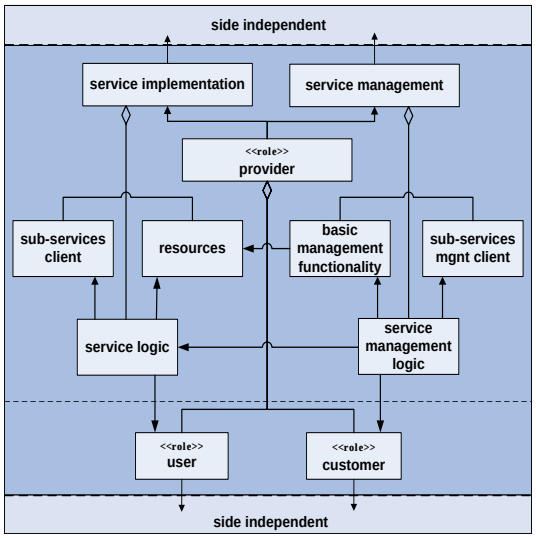

Abbildung 3.9: Der Realisation-View des MNM-Dienstmodells

Seiten-Unabhängig (*side independent*) Entsprechend der Hauptinteraktionsklassen beinhaltet der Dienst die Nutzungs- und Managementfunktionalität. Diese beiden Typen von Funktionalität müssen QoS-Parameter erfüllen. Die QoS-Parameter definieren ein Minimum an Dienstgüte, so dass der Dienst auf der Kundenseite nützlich wird. Sie definieren sowohl qualitative als auch quantitative Werte.

Die **Nutzungsfunktionalität** (*usage functionality*) umfasst die Interaktionen, die vom User genutzt werden. Diese Interaktionen repräsentieren den eigentlichen Zweck des Dienstes. Außerdem gibt es Interaktionen, die über den Zweck des Dienstes hinaus gehen und die benutzt werden zur Erfüllung der Kundenaufgaben, zur Anpassung des Dienstes an die Wünsche der Kunden/Nutzer und zur Kontrolle des Provisionings des Anbieters (Absch. 3.1.1.1). Diese Interaktionen sind von der **Managementfunktionalität** realisiert.

Die **Dienstvereinbarung** (*service agreement*) konkretisiert den Dienst, indem sie die Nutzungs- und Managementfunktionalität als auch die QoS-Parameter beschreibt.

Auf dieser Seite existieren auch Dienstschnittstellen (*service interface*) zwischen der Kundenseite und Anbieterseite, die deren Beziehungen zueinander definieren. Die Klasse der Dienstschnittstellen beinhalten Stammdienste (service primitives), Protokolle und evtl. physische Verbindungen. Die Definition der Schnittstellen ist ebenfalls in der Dienstvereinbarung beinhaltet, um der Kundenseite den Zugriff auf die Dienstfunktionalität zu gewähren. An der Schnittstelle endet die Zuständigkeit des Providers. Die Schnittstellen sind keine Teile des Dienstes. Durch Ändern und Hinzufügen von Schnittstellen ergibt sich kein neuer Dienst. Die Dienstvereinbarung spezifiziert den Dienst durch Definieren der Nutzungs- und Managementfunktionalitäten aus der Sicht des Kunden, der Dienstgüte und -logik und der technischen Aspekte der Nutzungs- und Managementschnittstellen zwischen Kunde und Anbieter.

Auf dieselbe Art und Weise, wie die Funktionalität in Nutzungs- und Managementfunktionalität gesplittet worden ist, wird auch die Schnittstelle aufgeteilt in **Nutzungsschnittstelle** (*usage interface*), auch **Service Access Point** (**SAP**) genannt, und **Managementschnittstelle** (*management interface*),

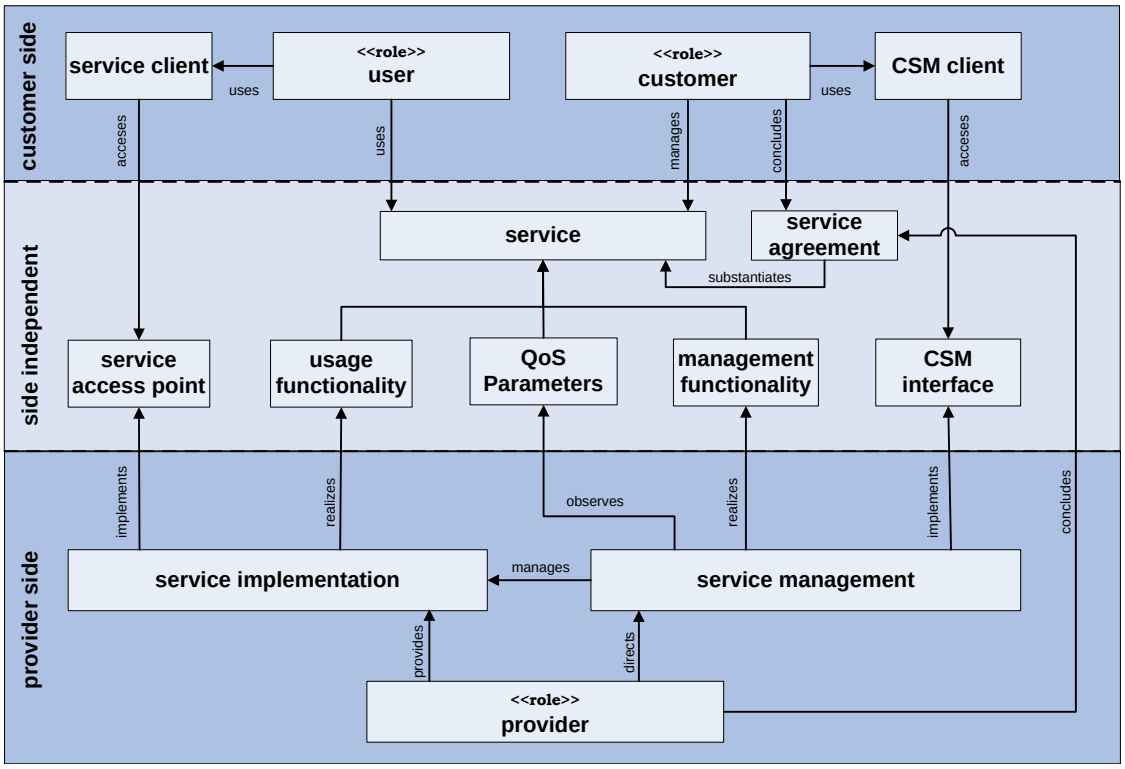

Abbildung 3.10: Die Dienst-Sicht (Service View) des MNM-Modells

auch **Customer Service Management Interface** (**CSM**)[LaNe 00] genannt. Wie in Abbildung 3.10 dargestellt, ist über die SAP die entsprechende Nutzungsfunktionalität und über die CSM-Schnittstelle die Managementfunktionalität erreichbar.

Kundenseite (*customer side*) Auf der Kundenseite ist eine Einrichtung notwendig, um die Dienstfunktionalität zu erreichen. Die **Clients** ermöglichen User und Customer, die Funktionalität bei der SAP und CSM-Schnittstelle zu erreichen. Beispiele für Clients sind: Telefone, Computer oder Applikationen. Die Zuständigkeit der Clients beschränkt sich nur auf die Customer Side.

Anbieterseite (*provider side*) Die wichtigste Aufgabe der Anbieterseite ist es, den Dienst verfügbar zu machen. Mit anderen Worten müssen die Nutzungs- und Managementfunktionalität auf der Kundenseite die QoS-Parameter erfüllen und die Schnittstellen müssen die Nutzung und das Management des Dienstes ermöglichen.

Weiterhin gibt es eine **Dienstimplementierung** (*service implementation*), die die Nutzungsfunktionalität realisiert. Um die Funktionalität für den Nutzer verfügbar zu machen, muss die Dienstfunktionalität auch die SAP implementieren. Also ist die Dienstimplementierung eine Zusammensetzung aus Wissen, Mitarbeitern, Software und Hardware, benötigt zur Realisierung der Nutzungsfunktionalität und der SAP.

Der Provider ist ebenfalls für das **Dienstmanagement** zuständig. Das Dienstmanagement (*service*

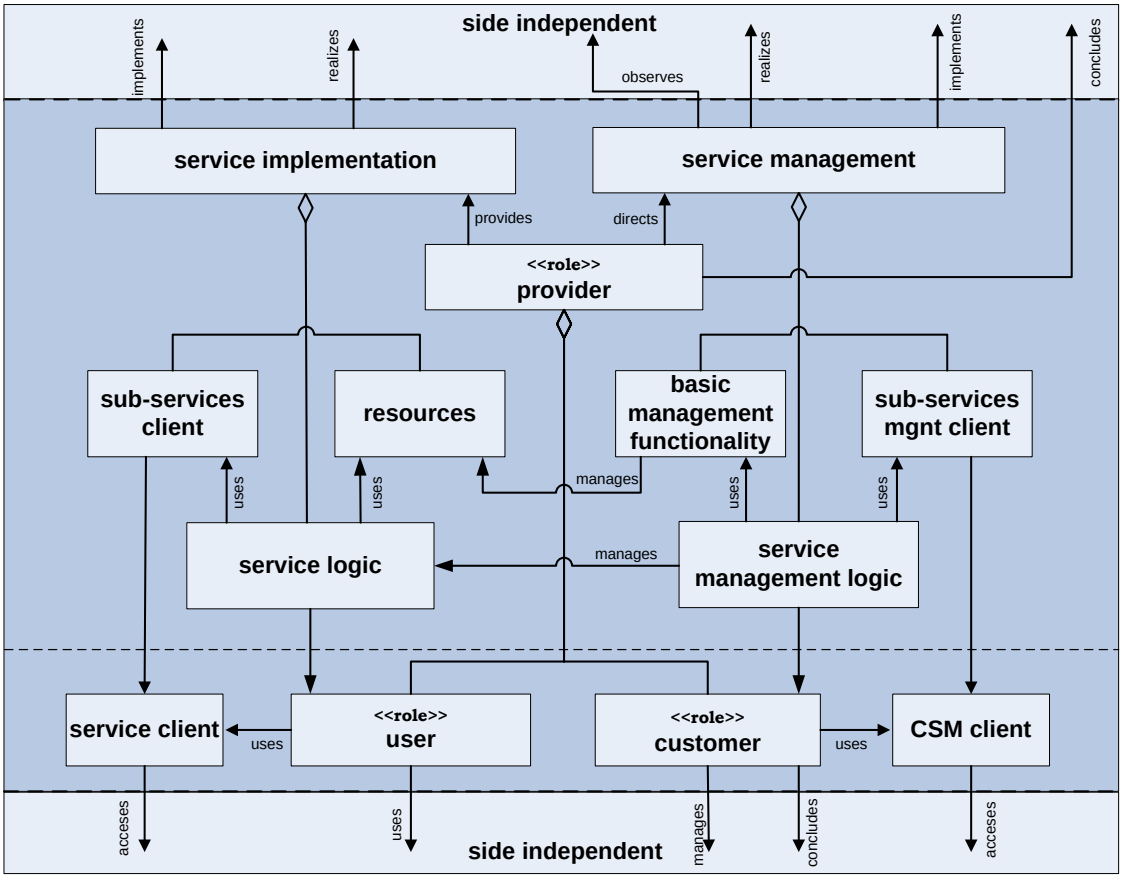

Abbildung 3.11: Die Realisierungs-Sicht (Realization View) des MNM-Modells

management) dient der Sicherstellung der Einhaltung der Dienstvereinbarung. Das heißt, die QoS-Parameter in den vereinbarten Rahmen durch Bereitstellung der Dienstimplementierung einzuhalten. Zusätzlich implementiert es die Managementschnittstelle für die Kundenseite und die Managementfunktionalität.

Die Realisierungs-Sicht (*Realization View*) Die Dienst-Sicht konzentriert sich auf die Kunden-Anbieter-Beziehung (*customer-provider-relationship*) aus dem Blickwinkel des Kunden. Ihr Zweck ist insbesondere zur Spezifikation der Dienstvereinbarungen wichtig. Die **Realisierungs-Sicht** wird zur Modellierung der providerinternen-Realisierung des Dienstes benutzt. Daher dient die Realisierungs-Sicht als Grundlage für die Implementierung eines bestimmten Dienstes. Diese Sicht muss auch die Tatsache reflektieren können, dass einige Dienste durch andere Unteranbieter erbracht werden. Das kann weiter fortgesetzt werden, wenn z.B der Unterprovider die Dienste von einem anderen Provider in Anspruch nimmt, was zu Dienstketten und -hierarchien führt (siehe auch Abschnitt 3.1.1.3). Also nutzt man die Realisierungs-Sicht auch zur Modellierung entstandener Diensthierarchien.

Der Anbieter, der einen Dienst von einem anderen Anbieter in Anspruch nimmt, wirkt für den zweiten als Kunde. Das heißt, der Anbieterbereich beinhaltet gleichzeitig die Nutzer/Kunde-Rolle und die

Anbieter-Rolle. Durch Erweiterung des Anbieterbereiches (*provider domain*) mit den Entitäten des Kundenbereiches (*customer domain*) erhält man die Klassen: Dienstimplementierung (*service implementation*) und Dienstmanagement (*service management*), Service-Clients und CSM-Clients als auch die Rollen: Anbieter, Nutzer und Kunde

In der Abbildung 3.11 wird die providerinterne Realisierung des Dienstes und sein Management repräsentiert. Die Clients, die für den Zugriff auf Unterdienste zuständig sind, werden auch in der Realisierungs-Sicht betrachtet, um die Interaktion mit dem Unteranbieter zu ermöglichen.

Die **Dienstimplementierung** beinhaltet die verfügbaren *Ressourcen* und (Sub)Dienste, die über den *Sub-Service-Clients* erreichbar sind. Folglich hat die *Dienstlogik* (*service logic*) die Rolle des Koordinators der beiden: die Nutzung der (Sub)Dienste als auch die Nutzung der Ressourcen des Anbieters.

Das **Dienstmanagement** besteht aus der **Basis-Managementfunktionalität** (**BMF**) (Funktionalität des klassischen Netzes, das System- und Anwendungsmanagement) als auch aus der von Unterdiensten angebotenen Managementfunktionalität. Auch auf dieser Seite existiert eine *Managementlogik* (*management logic*), die die BMF und die *Sub-Service-Management-Clients* für untergeordnetes Dienstmanagement kontrolliert.

Entsprechend der Assoziation zwischen Dienstmanagement und Dienstimplementierung, koordiniert die Managementlogik die Dienstlogik. Das heißt, dass eine Assoziation zwischen den Managementlogik- und Servicelogik-Klassen besteht.

3.1.1.5 Instantiierungsmethodik

In [GHHK 02] wird eine Methodik zur Instantiierung dargestellt. Diese beinhaltet mehrere Schritte. Schritt eins ist die Darstellung des Arbeitsablaufs (Workflow) für das Basismodell. Danach folgt Schritt zwei was seinerseits zwei Varianten haben kann. In der ersten Variante wird der *top down-Arbeitsablauf* des Service Views gefolgt von den *bottom up-Arbeitsablauf* des Realization Views realisiert. In der zweiten wird der *bottom up-Arbeitsablauf* des Realization Views gefolgt von den *top down-Arbeitsablauf* des Service Views dargestellt.

Der Arbeitsablauf für das Basismodell beinhaltet die zwei Schritte Rollenidentifikation und Dienstbenennung (Service Naming). Die Rollen, die bei der der Erbringung des Dienstes beteiligt sind (Kunde-Anbieter-Beziehung), müssen definiert werden aber gleichzeitig auch die Transparenzen zwischen Kunde und Dienst, Dienst und Subdienst usw. Bei der Dienstbenennung, wie der Name auch andeutet, muss man einen Namen für den Dienst finden. Für alle anderen intransparenten (sichtbaren) Sub-Dienste muß ebenfalls ein Name gefunden werden. Danach folgt entweder der Top Down- oder der Bottom Up- Modellierungsprozess.

Der Top Down-Modellierungsprozess definiert mit Hilfe der Analyse der Kundenanforderungen den Service View. Zur Fertigstellung des Service Views muß man erst die Nutzungs- bzw. Managementfunktionalitäten definieren, gefolgt von der Spezifikation der QoS-Parameter für jede davon. Dafür werden für jede der Funktionalitäten *UML Use Case Diagramme* erstellt, die im nächsten Schritt dann zu *Aktivitätsdiagrammen* verfeinert werden. Aufgrund der schon realisierten Diagramme wird der Realization-View hergeleitet. Die providerinterne Prozesse und Aktivitäten für jede der oben realisierten Diagramme werden definiert. Für jede der Aktivitäten werden *UML Kollaborationsdiagramme* erstellt.

Der Bottom Up-Modellierungsprozess beginnt mit der Erstellung des Realization Views, aufgrund der Anforderungen seitens der Anbieter. Als erstes wird der Bottom Up-Ablauf zum Entwurf des Realization Views durchgeführt: UML Use Case Diagramme für die Providerinterne Prozesse realisieren, Clients, Ressourcen und BMF(Basic Management Funktionalität) identifizieren, Service und der Service Management Logic mit Hilfe der Kollaborationsdiagrammen spezifizieren. Danach folgt der Service View der oberen Diagramme durch Abstrahieren und Destillieren der Funktionalitäten entsteht. Es werden dann Aktivitätsdiagramme für jede der herausgefundene Funktionalität realisiert.

3.1.1.6 Bewertung nach Anforderungen

Das MNM-Dienstmodell repräsentiert ein allgemeines Modell, das auf jeden Dienst anwendbar und technologieunabhängig ist. Es werden Rollen zugewiesen und die Funktionalitäten werden in Nutzungs- und Managementfunktionalität geteilt. Es wird die Implementierung des Dienstes von der tatsächlichen Erbringung des Dienstes getrennt. Der Lebenszyklus des Dienstes, als auch die QoS-Parameter für die Erbringung, wurden bei der Entwicklung dieses Modells miteinbezogen.

Innerhalb der providerinternen Realisierung des Dienstes wird dann u.a. auf Ressourcen, Unterdienste und Managementfunktionalitäten der Unterdienste zugegriffen. Die verschiedenen Abhängigkeiten wurden schon erwähnt, aber noch nicht definiert. Im Realization View und in der Service Logic findet man einen Teil der Abhängigkeiten; aber auch nur teiweise beschrieben, und zwar die zwischen Dienst und Subdiensten und die zwischen Dienst und Ressourcen. Statische Abhängigkeiten können auch einigermaßen herausgefunden und modelliert werden, andere Arten aber schwieriger. Der Vorteil ist, dass das MNM-Dienstmodell sehr gut die dienstbezogenen Anforderungen erfüllt. Das Modell ist leicht auf jeden Dienst anwendbar und gleichzeitig, z.B. für Abhängigkeitsmodellierung, erweiterbar.

3.1.2 Abhängigkeitshierarchien

Die Untersuchung der Abhängigkeiten in verteilten Systemen wurde von Keller und Kar in [KeKa 01] beschrieben. Desweiteren wurde präsentiert, wie ein bestehendes Netz und das dazugehörige Management zu einem Management der Anwendungsdienste erweitert werden kann. Das wird durch sogenannte **Network Service Providers** (**NSP**) realisiert. Ein NSP definiert ein **Virtual Private Network** (**VPN**) über die Infrastruktur des **Wide Area Network** (**WAN**) für einen bestimmten Kunden, so dass unterschiedliche Standorte miteinander kommunizieren können. Das bringt dem Kunden wesentlich niedrigere Kosten für Konnektivität und höhere Flexibilität im Hinblick auf die Bandbreite.

Diese Arbeit basiert auf einer Reihe von Forschungen auf dem Gebiet der Dienstabhängigkeit. In [EnKe 01] und [EnKe 01a] werden von A. Keller und C. Ensel folgenden Themen beschrieben: die Abhängigkeitshierarchien, die Anforderungen an ein Abhängigkeitsmodell, eine Architektur für Dienstabhängigkeiten und wie diese Daten dann in XML/RDF umzusetzen sind.

In der Arbeit wird davon ausgegangen, dass in Abwesenheit eines einheitliches Formats der Austausch zwischen unterschiedlichen Abhängigkeitsmodellen über mehreren Plattformen sehr schwierig ist. Dienste sind von anderen Diensten abhängig und daher unterscheidet man in [KeKa 01] zwischen **Dependents** und **Antecedents**. **Dependents** sind Dienste, die von anderen Diensten abhängig sind.

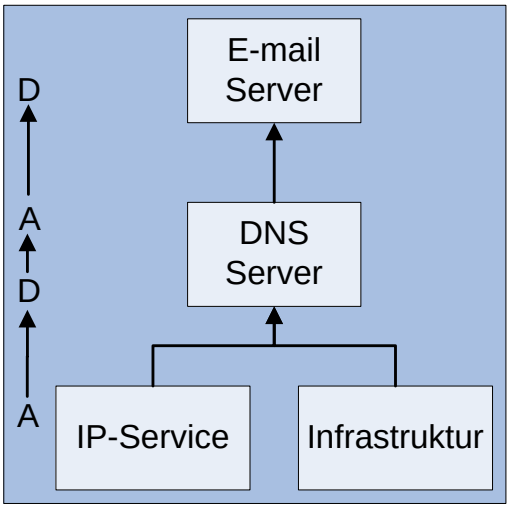

Abbildung 3.12: Beispiel einer Abhängigkeitshierarchie

Antecedents sind diejenigen, von denen andere Dienste abhängig sind. Zu bemerken ist, dass ein Dienst beide Rollen einnehmen kann. Das heißt, ein Dienst ist für andere Dienste vital, aber seine Funktionalität kann nicht existieren ohne andere Dienste, von denen er abhängig ist. Diese Eigenschaft steht daher als Basis für die **Abhängigkeitshierarchien** (*dependency hierarchy*), die mit einem gerichteten Graphen modelliert werden. Ein vereinfachtes Beispiel davon (angewandt auf das E-Mail-Dienst Szenario) ist in der Abbildung 3.12 dargestellt.

Der E-Mail-Dienst ist Dependent für den DNS und der DNS ist Antecedent für den E-Mail-Dienst. Der DNS ist aber auch Dependent für den IP-Dienst und für die Infrastruktur. Auf dieser Weise kann man beliebig viele Ebenen von Diensten in die Hierarchie mit einbeziehen.

Bewertung nach Anforderungen Abhängigkeitshierarchie ist ein wichtiger Begriff im IT-Service Managemet und ist für die Abhängigkeitsmodellierung relevant sein. Der technologieunabhängige Ansatz erfüllt die Voraussetzungen bezüglich der Allgemeinheit des Dienstbegriffes. Das Modell ist erweiterbar und anwendbar auf jeden Dienst. Diese Punkte sind aber nicht in dem Maße erfüllt, wie beim MNM-Dienstmodell. Lebenszyklus, Dienstgüte und Dienstfunktionlitäten werden nicht miteinbezogen. Im Bereich der abhängigkeitsbezogenen Anforderungen werden bis auf die dynamischen Abhängigkeiten fast alle Punkte erfüllt. Die XML/RDF Implementierung des Modells ist aber eine gute Voraussetzung, um eine ähnliche Implementierung für das MNM-Dienstmodell zu realisieren.

3.1.3 Abhängigkeitsgraphen

Gruschke beschreibt in [Grus 98b] und [Grus 99] die Abhängigkeitsgraphen, die für die Entwicklung eines Ereigniskorrelators benutzt worden sind. Eine Methodik zum Aufbau der Abhängigkeitsgraphen wird hingegen in [Kais 99] beschrieben. Dessen Nützlichkeit hat sich beim Management von Events in verteilten Systemen erwiesen. Ein Abhängigkeitsgraph ist die einfachste Datenstruktur, mit der man die Relationen zwischen den **Managementobjekten (engl. Managed Object)** (**MO**) darstellen kann. Er ist ein gerichteter Graph, dessen Knoten MO und dessen Kanten Abhängigkeitsbeziehungen zwischen ihnen repräsentieren.

Arten von Abhängigkeiten In [Grus 99] werden drei Arten von Abhängigkeiten definiert: QoS-Abhängigkeiten, Security-Abhängigkeiten und funktionale Abhängigkeiten. Unter **QoS-Abhängigkeiten** versteht man, dass die Dienstgüte eines höheren Dienstes aus Kennzahlen tieferliegender Dienste ermittelt wird, also von der Dienstgüte tieferliegender Dienste abhängt. Die Knoten des Graphen müssen unter Umständen mit einzelnen Meßverfahren und die Abhängigkeitsbeziehungen mit Aggregationsvorschriften attributiert werden. Gruschke verfolgt aber in seiner Arbeit diesen Ansatz der QoS-Abhängigkeiten nicht weiter.

Der nächste Ansatz der **Security-Abhängigkeiten** kommt aus dem Bereich des Sicherheitsmanagements. Die Bewahrung der Integrität der Managementobjekte ist eine seiner Aufgaben. Der Begriff der Security-Abhängigkeit wird dadurch definiert, dass man annimmt, dass die Integrität eines bestimmten *MO* nur dann gewährleistet werden kann, wenn auch die Integrität anderer bestimmter *MO* aufrechterhalten wird. In den verteilten Systemen steigt die Anzahl der Security-Abhängigkeiten selbst bei einer kleinen Anzahl an *MO*s sehr stark und das System wird daher undurchschaubar. Diese Arten werden auch nicht mehr weiter betrachtet.

Die Kategorie der **funktionalen Abhängigkeiten** ist diejenige, die beim Aufbau des Eventkorrelators benutzt wird. Eine funktionale Abhängigkeit von einem *MO* A zu einem *MO* B besagt, dass der von B erbrachte Dienst für die Diensterbringung durch A notwendig ist. Das heißt, A ist funktional abhängig von B, falls das *MO* A das *MO* B benötigt, um funktionsfähig zu sein. Eine Störung in B kann so zu einer Störung in A führen.

Während *MO*s als Knoten repräsentiert werden, gibt es für Managementobjektklassen (MOC) keine Entsprechung in Abhängigkeitsgraphen. Die Notation, die benutzt wird, ist die UML-Notation: die Knoten werden mit abgerundeten Ecken repräsentiert und die Pfeile dazwischen werden als „ist funktional abhängig davon“ interpretiert. Im Gegensatz zu objektorientierten Modellen hat diese keine Attribute, Methoden und auch keine andere Beziehungen außer den Abhängigkeiten. Das System, das gemanagt wird, ist dynamisch, und dadurch muß auch der Abhängigkeitsgraph dynamisch sein [Grus 98b]. Seine Knoten und Kanten können eingefügt und gelöscht werden, so dass ein reales System approximiert werden kann.

Bewertung nach Anforderungen In diesem Ansatz wird die Abhängigkeit wie eine Beziehung zwischen unterschiedlichen Entitäten definiert. Er wurde so entworfen, dass er für die Korrelation auf unterschiedlichen Abstraktionsstufen benutzt werden kann. Der Begriff der „Abhängigkeit“ wurde aber nicht tiefer definiert oder beschrieben. Die Abhängigkeitsgraphen erfüllen weder die dienstbezogenen noch die abhängigkeitsbezogenen Anforderungen (siehe zusammenfassende Tabelle 3.1 auf Seite 48).

3.1.4 Abhängigkeitsgrade

In [BKH 01] stellen Bagchi et al fest, dass eine verteilte Umgebung als aufeinander aufbauende Schichten von Ressourcen logisch modelliert werden kann. Unter Ressourcen versteht man nach diesem Ansatz: Dienste, Applikationen und andere Software- und Hardwarekomponenten, die zur Realisierung des Dienstes mitwirken. Dienste und/oder Komponenten einer Schicht sind von Funktionen der darunter liegenden Schichten abhängig. Daher beeinflussen Störungen in einer Schicht die Funktionalität der Komponenten in der abhängigen Schicht.

Abhängigkeitserkennung und Abhängigkeitsanalyse Das System wird als gerichteter Graph (engl. **Directed Acyclic Graph** (**DAG**)) beschrieben. Die Knoten stehen für Systemkomponeten (Dienste, Applikationen, Software, Hardware, Netze) und die **gewichteten** Kanten repräsentieren die Abhängigkeiten zwischen den Knoten. Eine Abhängigkeitskante zwischen zwei Knoten zeigt, dass ein Ausfall oder eine Störung an dem Knoten an der Spitze der Kante die Aktivität des Knoten am Ende der Kante beeinträchtigen kann. Das Kantengewicht repräsentiert die Größe der Auswirkung beim Ausfall des Knoten an der Spitze der Kante auf den Knoten am Ende der Kante.

Die Identifizierung der Hauptursache eines Problems wird dann durch Navigieren im Graph herausgefunden. Theoretisch läuft das hervorragend. In der Praxis jedoch gibt es das Problem, dass Abhängigkeiten nicht so explizit definiert sind. Das, was in [BKH 01] angestrebt wird, ist ein dynamisches Modell, in dem Abhängigkeitsketten reflektiert werden. Zur Untersuchung der Abhängigkeiten werden zwei Methoden vorgeschlagen:

- **Static Dependency Analysis** (*statische Abhängigkeitsanalyse*) - benutzt Informationen aus den Konfigurationsdateien, die das System beschreiben. Dieser Ansatz ist aber fehlerhaft, weil das Bild der Abhängigkeiten nicht komplett ist. Das heißt, es werden insbesondere Abhängigkeiten die einzelnen Knoten oder Abhängigkeiten die innerhalb eines beschränkten Bereiches zugewiesen sind, dargestellt. Es wird aber nicht das Gesamtbild der Abhängigkeiten im System dargestellt.
- **Active Dependency Discovery** (**ADD**) (*aktive Abhängigkeitserkennung*) - repräsentiert die aktive Manipulierung (Störung) der Systemkomponeten, während man die Reaktion des Systems überwacht. Die Methode ist gut, hängt aber sehr stark von den Ereignissen ab, die induziert werden.

Es werden dann die Abhängigkeitsgrade definiert. Man kann daraufhin zwischen **keinen (*absent*)**, **schwachen (*weak*)**, **mittleren (*medium*)** und **starken (*strong*)** Abhängigkeisbeziehungen unterscheiden.

Bewertung nach Anforderungen Der Vorteil dieses Ansatzes ist, die Definition der Stärke der Abhängigkeit zwischen zwei Komponenten. Die Abhängigkeitsgrade muß man aber intuitiv auswählen, da keine Methodik vorgeschlagen wird, um diese Abhängigkeitsgrade zu definieren. Der Ansatz erfüllt die Allgemeinheit des Dienstbegriffes nur teilweise und ist nur teilweise erweiterbar. Die restlichen Punkte der dienstbezogenen Anforderungen sind nicht erfüllt. Es werden Beziehungen zwischen Diensten und Subdiensten und statische Abhängigkeiten definiert, aber die anderen abhängigkeitsbezogenen Anforderungen sind gar nicht oder nur teilweise erfüllt.

3.1.5 Verfügbarkeit

Die Verfügbarkeit der verteilten Applikationen und gleichzeitig die Verfügbarkeit der Dienste wurde in [DrKa 97] und [Kais 99] zusammengefasst.

Die Verfügbarkeit (*availability*) ist eines der wichtigsten Kriterien zur Beurteilung der Dienstgüte. Sie gibt also teilweise die Qualität der Diensterbringung wieder. Kaiser [Kais 99] stellt Kriterien zur Feststellung der korrekten Funktionsweise des Dienstes vor:

- **Definition anhand von Fehlermeldungen**
  Der Provider nimmt an, dass der Dienst korrekt läuft, solange vom Kunden keine Fehlermeldung vorliegt. Ein Dienst gilt als ausgefallen von dem Zeitpunkt an, wenn die Meldung vom Kunden bzgl. des Ausfalls eingetroffen ist, bis zur Fehlerbeseitigung.
- **Auftragsbezogene Definition**
  Es wird eine korrekte Funktionsweise eines Dienstes anhand der Erledigung der Dienstanfragen festgestellt.

- **Ausfallbezogene Definition**
  Die korrekte Funktionsweise des Dienstes wird durch Überwachung des Dienstes und dessen Komponenten festgestellt.

Da die Verfügbarkeit ein wichtiger Qualitätsparameter des Dienstes ist, basiert sie auf der Kenntnis der Abhängigkeiten im System, die zur Realisierung des Dienstes beitragen. Als graphische Darstellung werden die Verfügbarkeitsgraphen benutzt, in dem die Knoten die parametrisierten Dienstdeskriptoren (z.B DNS, IP-Connectivity) und die Kanten die funktionalen Abhängigkeiten (nur zum Zweck der Kalkulation) repräsentieren. Es werden die **AND** und **OR** Abhängigkeiten definiert. Ein Beispiel dafür wäre: der E-Mail-Dienst ist verfügbar, wenn der DNS **AND** die IP-Connetivity verfügbar sind. Hingegen für **OR**, ist die Redundanz zwischen zwei DNS-Servern ein passendes Beispiel: LMU-DNS **OR** LRZ-DNS (also im Falle eines Fehlers des LMU-DNS-Servers greift man automatisch auf den DNS-Server des LRZ zu).

Bewertung nach Anforderungen Die Anforderungen sind in Bezug auf diesen Ansatz, bis auf Technologieunabhängigkeit, Miteinbeziehen des Dienst-Lebenszyklus und statische Abhängigkeit, nur teilweise oder gar nicht erfüllt. Die Unterscheidung zwischen **AND** und **OR** ist einer der Punkte, die für die Abhängigkeitsmodellierung wichtig sind. Diese zwei Arten von Abhängigkeiten sind aber nicht tiefer beschrieben. Die Tatsache dass diese viel zu einfach beschrieben sind und die QoS-Parameter nicht einmal in Betracht gezogen werden, ist der Grund dafür, dass dieser Ansatz in dieser Arbeit nicht verwendet werden kann.

3.1.6 Abhängigkeiten für Internet Dienst Provider

Caswell und Ramanathan [CaRa 99] beschreiben Abhängigkeiten insbesondere für Internet Service Provider. Sie differenzieren daher zwischen den folgenden Kategorien:

- **Execution dependency** (*Ausführungsabhängigkeiten*) Die Leistung eines Applikationsserverprozesses, der auf einer Hostmaschine läuft, ist von dem Status des Hostes abhängig.
- **Link dependency** (*Verbindungsabhängigkeiten*) Die Leistung eines Dienstes, der über eine Netzverbindung angeboten wird, ist von dem Status der Verbindung abhängig.
- **Component dependency** (*Abhängigkeiten zwischen den Komponenten*) Im Fall eines Webdienstes, der auf unterschiedlichen Front-End-Servern (die von einem Round-Robin DNS selektiert werden) läuft, ist die Leistung vom selektierten Server abhängig.
- **Inter-service dependency** (*Abhängigkeiten zwischen Diensten*) wie z.B Abhängigkeit zwischen E-Mail-Dienst und DNS.
- **Organizational dependency** (*Organisatorische Abhängigkeiten*) Dienste und/oder Server können unterschiedlichen Verantwortungsbereichen zugeordnet sein.

Bewertung nach Anforderungen Diese Arten von Abhängigkeiten sind sehr ähnlich zu denen, die durch diese Arbeit modelliert werden sollen. Die inter-service dependencies z.B. entsprechen der im Abschnitt 2.3.2.3 dargestellten Anforderungsanalyse den Abhängigkeiten zwischen Diensten und Subdiensten. Die link und component dependencies sind das Analogon für Abhängigkeiten zwischen

Ressourcen (siehe 2.3.2.1). Die organisatorischen Abhängigkeiten werden in dieser Arbeit nicht betrachtet. Diese Einteilung ist jedoch nur auf ISP-Szenarien eingeschränkt.

3.2 Standardisierungsansätze

Im vorherigen Abschnitt wurden unterschiedlichen Ansätze aus dem Forschungsbereich vorgestellt, jedoch keiner davon ist eine Standard. Dieser Abschnitt befasst sich mit einigen wichtigen Standards, die im Bereich des IT-Service Managements bzw. Informationsmodelle existieren. Zunächst wird im Abbschnitt 3.2.1 die Modellierung der Abhängigkeiten im Common Information Model (CIM) beschrieben. Abschnitt 3.2.2 befasst sich mit der Beschreibung von Abhängigkeiten aus der Sicht vom IT Infrastructure Library - der De-facto-Standard im Service Management Bereich. Im Abschnitt 3.2.3 wird Enhanced Telecom Operations Map (eTOM), ein Standard in Telekommunikationsbereich, und die Art wie die Abhängigkeiten dargestellt werden, beschrieben. Zusätzlich wird das, zur eTOM gehörende, Shared Information and Data Model (SID) im Bezug auf Abhängigkeiten analysiert.

3.2.1 Das Common Information Model (CIM)

Das **Common Information Model (CIM))** in [HAN 99a] auch ,,Renaissance der komplexen Informationsmodelle“ genannt, ist ein objektorientiertes Informationsmodell, das auf der **Unified Modelling Language** (**UML**) basiert. Dieses Modell wurde von der **Distributed Management Task Force** (**DMTF**), einer Organisation, die sich mit der Entwicklung von Managementstandards und Integrationstechnologien für Unternehmens- und Internetumgebungen befaßt, herausgegeben. Es ist ein konzeptionelles Informationsmodell zur Beschreibung von Management, das keine bestimmte Implementierung besitzt.

CIM-Schema Das Metaschema der CIM-Spezifikation [cim 03] definiert die Bezeichnungen, die verwendet werden, um das Modell und seine Verwendung und seine Semantik auszudrücken. Die Elemente des Metaschemas sind Klassen (*classes*), Eigenschaften (*properties*) und Methoden (*methods*) (siehe Anhang A Abbildung A.1).

Das CIM-Schema selbst wird in drei eindeutigen Schichten strukturiert:

- **Das Kernschema** (*core schema*) ist ein Informationsmodell, das Begriffe beinhaltet, die auf alle Bereiche des Managements anwendbar sind (siehe Abbildung 3.13) Diese UML-Repräsentation des Kernschemas wird mit Hilfe anderer UML-Diagramme erweitert (siehe Anhang A): Abbildung A.2 zeigt detailliert die hierarchische Darstellung des ManagedSystemElement, Abbildung A.3 stellt die CIM Service/SAP Klassen dar und in Abbildung A.4 werden die CIM Konfigurations- und Einstellungsklassen präsentiert.
- **Die gemeinsamen Schemata** (*common schemas*) sind Informationsmodelle, die Begriffe für bestimmte Managementbereiche, die sich nicht auf eine bestimmte Technologie oder Implementierung beziehen, beinhalten. Die gemeinsamen Bereiche sind Systeme, Vorrichtungen, Netze, Anwendungen, Metriken, Datenbanken, die Umgebung, Ereignisdefinition und -behandlung, Management einer CIM-Infrastruktur, Benutzer und Sicherheit, Policy- und Trouble-Ticket Austauschinformationen.

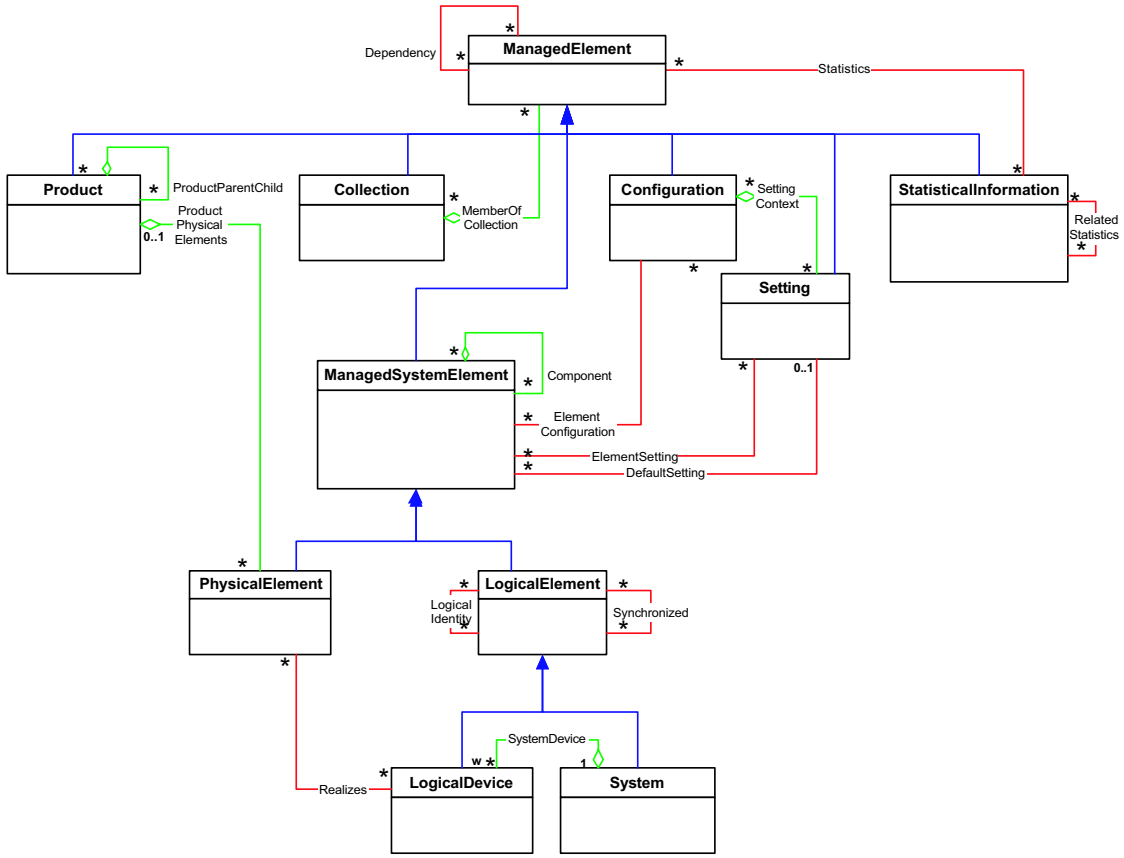

Abbildung 3.13: Das Kernschema - die CIM Objekthierarchie [cim00]

- **Die Erweiterungsschemata** (*extension schemas*) verfeinern die gemeinsamen Schemata für spezielle Technologien.

Die formale Definition des CIM-Schemas ist in mehreren **Managed Object Files** (**MOF**) ausgedrückt. Ein MOF ist eine ASCII- oder UNICODE-Datei (**American Standard Code for Information Interchange** (**ASCII**)), die als Eingabe für eine Applikation in einen MOF-Editor, -Parser oder -Compiler benutzt werden kann.

Bewertung nach Anforderungen Als gemeinsames Informationsmodell bietet CIM einige Klassen, Attribute und Beziehungen, so dass dieser Ansatz die Möglichkeit gibt, die Abhängigkeiten zwischen Ressourcen zu definieren. Schwieriger wird es jedoch bei der Repräsentation der Abhängigkeiten zwischen Diensten, weil die Objekte inbesondere als Ressourcen definiert werden, ebenso die Dienstfunktionalitäten. Man kann ein zusätzliches Schema für Dienste und Abhängigkeiten konstruieren und als Teil der Erweiterungsschemata definieren. Die Abhängigkeiten zwischen CIM-Elementen (z.B *ServiceAffectsElement*) sind als Assoziationsklassen modelliert, die von einer einzigen Klasse *CIM_Dependency* abstammen. Daher ist das CIM Abhängigkeitsmodell „flach“ und nicht hierarchisch strukturiert. CIM ist ein technologieunabhängiger Ansatz. Er erfüllt nicht oder nur teilweise die dienstbezogenen Anforderungen (wie in der Tabelle 3.1 dargestellt). Einer der größten Nachteile bei CIM ist es, das neue Erweiterungen nur durch eine totale Veränderung der Klassen, Operationen und Einstellungen stattfinden kann und nicht durch einfaches Hinzufügen neuer Klassen.

3.2.2 Abhängigkeiten aus der Sicht vom ITIL (IT Infrastructure Library)

IT Infrastructure Library (**ITIL**) ist heute der weltweite De-facto-Standard im Service Management Bereich und beinhaltet eine umfassende und öffentlich verfügbare fachliche Dokumentation zur Planung, Erbringung und Unterstützung von IT-Serviceleistungen. Die ITIL Bücher stellen aber nur einen Best Practice Leitfaden für Service Management dar, in dem das ,,WAS“ beschrieben wird, und nicht das ,,WIE“[ITSS 00].

Für das Thema der Abhängigkeitsmodellierung sind für diese Arbeit allerdings das Configuration- und das Change-Management die bedeutensten. Das **Change Management** steuert die Veränderungen im IT-Betrieb unter Berücksichtigung von wirtschaftlichen Gesichtspunkten und minimiert negative Auswirkungen auf die Service-Qualität. Das **Configuration Management** stellt hierfür ein durchgängiges logisches Modell der Infrastrukturkomponenten zur Verfügung und kontrolliert deren Konfigurationen.

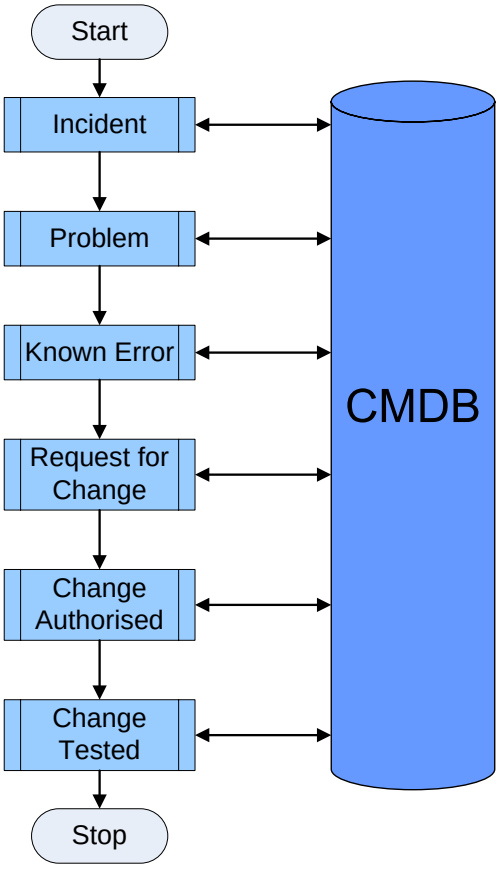

Abbildung 3.14: Die CMDB [ITSS 00]

Zentrale Grundlage ist die **Configuration Management Database** (**CMDB**), die sich bei konsequenter Umsetzung zu einem Unternehmensrepository entwickelt, in dem neben den eigentlichen **Configuration Items** (**CI**) (Hardware, Software und andere Elemente der IT-Infrastruktur) auch Services, Rollen und Prozesse etc. abgebildet werden können. Abhängigkeiten der Komponenten werden auch als CIs in der CMDB gespeichert.

Einige der Anforderungen an die CMDB orientieren sich auf das Herausfinden von CIs, die kritischen Dienste und dessen Komponenten beschreiben.

Das Configuration Management liefert die wesentlichen Grundlagen für die Integration der Dienstmanagement-Prozesse. Hier wird nicht nur sichergestellt, daß die IT-Konfigurationen kontrolliert und dokumentiert werden. Ganz wichtig ist der Zusammenhang zwischen Infrastrukturkomponenten, Anwendern, Diensten, Vereinbarungen, Prozessen und Vorgängen wie Störungen, Problemen oder Änderungen. Wie in der Abbildung 3.14 sichtbar ist, sind alle wichtigen Prozesse des Service Supports von der CMDB abhängig. Das liegt daran, dass die CMDB, wie oben erwähnt, die Konfigurationsitems beinhaltet. Wenn z.B. aufgrund einer Störung (Incident) im System ein Change eingeführt wird, erfolgt der Ablauf entspechend der Abbildung 3.14.

In der Configuration Management Policy und im Konfigurationsplan müssen die richtigen Entscheidungen darüber getroffen werden. Entscheidend sind hierfür die Anforderungen aus den Lifecycle-Prozessen der Komponenten, aus dem Servicemanagement und nicht zuletzt auch die Besitzverhältnisse für Komponenten.

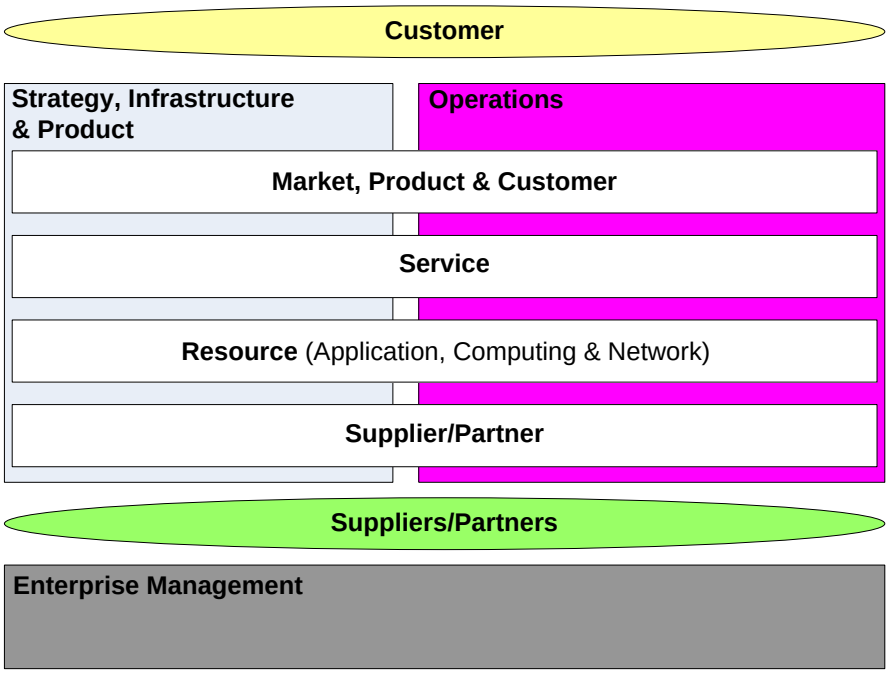

Abbildung 3.15: eTOM Level-0-Sicht [eTOM 04]

Bewertung nach Anforderungen ITIL als Best-Practice Dokumentation, liefert einen Rahmen für die Abhängigkeitsmodellierung durch die CMDB, die eigentlich alle Konfigurationsdaten beinhaltet, also implizit auch alle schon beschriebenen Abhängigkeiten. Die Anforderungen an die CMDB werden in [ITSS 00] beschrieben. Das Problem ist, dass es keine Methodik und Tools zur Erkennung und Beschreibung der Abhängigkeiten gibt. In ITIL wird sehr viel aus einem unternehmerischen Gesichtspunkt beschrieben. Die Abhängigkeiten werden aber direkt nicht einmal erwähnt. Es ist bekannt dass Abhängigkeiten wichtig sind, dass diese in der CMDB in der Form eines CI eingetragen werden müssen. Es gibt aber keinerlei Angaben über die Art und Weise, wie man das realisiert.

3.2.3 Abhängigkeiten aus der Sicht der eTOM und SID

Die **Enhanced Telecom Operations Map** (**eTOM**) [eTOM 04] ist ein Entwurf des **TeleManagement Forums** (**TMF**). In diesem Forum haben sich zahlreiche Telekommunikationsunternehmen zusammengeschlossen, um die für diese Art der Unternehmen typischen Prozesse zu analysieren und aufzustellen. Wie ITIL ist auch eTOM ein Standard, aber nicht in IT-Bereich, sondern in Telekommunikationsbereich. Die eTOM wird aber im IT-Bereich auch eine sehr große Rolle spielen, da die Entwicklungen der IT und der Telekommunikation Hand in Hand gehen. eTOM ist die Bereitstellung eines Industrieeigenen Modells von Geschäftsprozessen [Bren 04a], definiert und führt neue Terminologie für das Dienstmanagement ein. Die eTOM definiert verschiedene Sichten auf die Prozesse, die dann in mehrere Schichten eingeteilt werden. Je größer die Zahl des Schichten, umso verfeinerter ist die Sicht der Prozesse. In der Abbildung 3.15 wird der Level-0 der eTOM-Prozesse dargestellt (Process grouping). eTOM ist sehr stark kundenorientiert. Es beinhaltet zwei zentrale (vertikale) Prozessbereiche: **Operations** und **Strategy, Infrastructure & Produkt** (**SIP**). Die Operationsprozesse repräsentieren den eigentlichen Kern von eTOM und sind die einzigen, die ausreichend beschrieben

werden. Sie befassen sich mit der Bereitstellung und dem Betrieb von Dienstinstanzen. Die Strategy, Infrastructure & Product Säule beinhaltet Prozesse, die nicht direkt zur Wertschöpfung beitragen, sondern sich mit langfristigen, dienstbezogenen Fragestellungen befassen [Bren 04a]. Quer über diese zwei Bereiche liegen vier **funktionale Bereiche (*Functional Areas***) die nach den in ihnen gemanagten Objekten benannt sind: **Markt, Produkt and Kunde**, **Service**, **Ressourcen** und **Supplier/Partner**.

3.2.3.1 Shared Information and Data Model (SID)

Eine Initiative des TMF ist **New Generation Operations Systems and Software** (**NGOSS**). Sie hat als Ziel die Entwicklung einer herstellerunabhängigen Architektur und Terminologie für komplette Managementlösungen. Durch NGOSS verspricht man sich, die Effizienz und Effektivität in der Interaktion Kunde-Provider zu steigern. Das NGOSS braucht aber ein eigenes Informationsmodell, um seine Komponeten definieren zu können. Das **Shared Information and Data Model** (**SID**) [SID 04] ergänzt den eTOM Prozessrahmen, indem es Sachverhalte in Bereich der Geschäftsprozesse, Mitarbeiter, Vermögen, Produkte und Dienste definiert.

Das SID ist ein Datenbankmodell, eine Implementierung von Softwareklassen, eine **Application Programming Interface** (**API**), das die Rahmen definiert, wie man eine Software schreibt. Es ist aber keine Definition der NGOSS-Klassen, der Plattformen, Protokolle, Softwaresprachen oder Softwareprodukte benutzt zur Entwicklung von NGOSS-Komponenten. Die Basiskonzepte, die in SID definiert sind Identitäten (*identities*) und Werte (*values*), klassifizierbare Dinge, Kontexte und Rollen, Zustände und Lifecycle und Beziehungen. Im analytischen Modell SID benutzt man die UML Notation mit Entitäten, Assoziationen, Aggregationen und Vererbung/Instanziierung.

SID und eTOM modellieren dieselben Inhalte, aber mit zwei unterschiedlichen Gesichtspunkten. eTOM fokussiert sich auf Geschäftsprozesse (also die Dynamik des Geschäfts). Das analytische Modell SID fokussiert sich auf statische Aspekte des Geschäfts (die Dinge, nach denen sich die Geschäftsprozesse richten und ihre Eigenschaften und Beziehungen). Der SID-Domain (obere Level von Gruppierung) entspricht den eTOM Level-0 Definitionen: `Market/Sales, Product, Customer, Service, Resource, Supplier/Partner`.

3.2.3.2 Bewertung nach Anforderungen

Der eTOM ist der bekannteste Ansatz im Bereich Telekommunikation. Wie bei ITIL ist es gut ihn zu kennen, aber die Beschreibung ist für die Zwecke dieser Arbeit viel zu abstrakt definiert. Der für diese Arbeit wichtiger Teil ist der *Operations*-Bereich, der sehr umfangreich und genau beschrieben wird, aber immer noch viel zu allgemein und ohne konkrete Vorstellungen. Das SID ist auch ein sehr gutes und umfangreiches Informationsmodell, aber es basiert auf eTOM und daher sind die für uns wichtigen Entitäten im Bereich SIP noch nicht definiert. SID verwendet ein objektorientiertes Modell. Er stellt ein Framework zur Bestimmung von Dienstmanagement Informationen bereit. Diese Informationen werden aber aus dem Blickwinkel des Geschäfts betrachtet. Als neuen Ansatz im Bereich der Modellierung, SID hat sich bis jetzt auf der Realisierung der wichtigsten Bausteine des Modells fokusiert. Das beinhaltet aber nicht eine systematische, hierarchische Modellierung der Abhängigkeiten.

3.3 Kommerzielle Anwendungen

Die vorherigen Abschnitte haben sich mit Ansätzen aus der Forschung und Standardisierung befasst, die aber nicht immer reibungslos in der Praxis anwendbar sind. In diesen Abschnitt werden zwei Tools beschrieben die im Fehlermanagement benutzt werden. Abschnitt 3.3.1 vermittelt ein Bild vom HP OpenView Service Navigator, ein sehr oft benutztes Tool im Managen von großen Netzen. In Abschnitt 3.3.2 wird IBM Tivoli Enterprise Console, ein anderes bekanntes Werkzeug beschrieben.

3.3.1 HP OpenView Service Navigator

Der Service Navigator ([OWSN 05] und [OWSN 05a]) ist ein Tool, das es ermöglicht, Abhängigkeiten zwischen Diensten zu definieren. Es wird dadurch eine Diensthierarchie hergestellt und unterschiedliche Verantwortlichkeiten werden verschiedenen Nutzern zugewiesen. Der Nutzer kann dann den aktuellen Dienststatus in einer Java GUI anschauen oder er kann in der gegebenen Hierarchie nach unten navigieren und die Informationen über die abhängigen Dienste ablesen. In so einer Diensthierarchie wird der Status eines Dienstes vom „Gewicht“ (im Sinne der Wichtigkeit) seiner Informationen und vom Status jedes Subdienstes berechnet.

Aufbau des Tools Die Haupteigenschaften und Vorteile, die der Service Navigator für die Nutzer bringen, sind das Managen von Applikationen und Diensten, rollenbasierte Dienstsichten, die Analyse von Auswirkungen eines Ausfalls (Service Impact) und die Analyse der grundlegende Ursachen (root cause) für das proaktive Management und eine schnellere Durchschnittszeit zur Recovery (engl. *mean time to recovery(MTTR)* und die Aufzeichnung aller Änderungen im Dienststatus.

Die Vater-Kind Beziehung in der Diensthierarchie wird von den folgenden Faktoren beeinflußt werden:

- Statusausbreitung repräsentiert, wie der Status eines Subdienstes zu dem Vaterdienst übertragen wird.
- Statusberechnung bezieht sich auf die Kalkulation des Status eines Dienstes. Er wird von der Glaubwürdigkeit seiner Informationen und vom Status jedes Subdienstes berechnet.
- Gewichtung der Kindobjekte trägt dazu bei, dass einige Subdienste mehr Wichtigkeit als anderen gegeben wird.

Der Service Navigator ist eigentlich ein Konglomerat von Tools, die bei dem Management eines großen Systems zusammenwirken. Die Funktionalitäten, die dieses komplexe Tool bieten, sind:

Dienstsichten (engl. *Service Views*) Alle Kompomenten der IT-Umgebung (Netzkomponenten, Computersysteme, Datenbanken, Applikationen) und ihre Abhängigkeiten werden in Beziehung mit den Geschäftsprozessen im allgemeinen gebracht. Mit diesen Dienstsichten werden als erstes die kritischen Dienste betrachtet.

Rollenbasierte Service-Views Abhängig von Rolle oder Kompetenz des Nutzers, können mehrere Sichten innerhalb der gemanagten Umgebung benutzt werden. Diese ermöglichen dem Nutzer, sich auf seinen Kompetenzbereich zu fokusieren.

Dienstbeeinflussung Analyse (engl. *service impact analysis*) Das Interpretieren der Daten von unteren Schichten ist wichtig für die Dienste auf oberen Schichten bei der Vorbeugung gegen Probleme, bevor sie die Geschäftsprozesse beeinflussen.

Navigation zu der grundlegende Fehlerursache (engl. *root cause*) Der HP OpenView Service Navigator erlaubt den IT-Fachleuten, die Ursache einer Störung durch Navigieren in dem Baum herauszufinden.

Dienstaktionen (engl. *service actions*) Nutzer können selbst dienstspezifische Aktionen definieren, die ihnen die Kontrolle über die Dienstapplikationen gewährt.

Out-of-the-box-Modelle Vordefinierte Dienstmodelle sind vorgesehen für die selektierten Dienste und Applikationen. HP will seine out-of-the-box-Modelle im HP OpenView Smart Plug-in erweitern.

Dienstberichte (engl. *service reports*) Zu einer besseren Bestimmung und Analyse der Störungen und Ereignisse ist die Software mit einer Datenbank vorgesehen, wo jeder Dienststatus, der jemals vorgekommen ist aufgezeichnet wird. Dieser HP OpenView Reporter benutzt die Daten zur Erzeugung von out-of-the-box Dienstaufzeichnungen (service reports). Diese können dann entweder wiederbenutzt oder als Basis für neue Aufzeichnungen verwendet werden.

Bewertung Durch dieses Tool wird das Service Management sehr leicht gemacht. Es ist auf fast jeder Plattform einsetzbar (mit den spezifischen Varianten). Die Abhängigkeiten zwischen Ressourcen werden sehr gut und einfach repräsentiert. Die Dienstabhängigkeiten, Redundanzen, Transparenzen des Systems sind allerdings nicht repräsentierbar. Dieser Ansatz basiert auf einem kommerziellen Produkt, das gekauft werden muß. Außerdem sind die Hintergrundinformationen bzgl. des Informationsmodells nicht bekannt und der Algorithmus ist verborgen, was wissenschaftlich nicht von großem Interesse ist. Das Produkt ist auch nur teilweise mit anderen Produkten (außer von HP) kompatibel. Daraufhin ist es schwierig ein solches Modell zu erweitern. Die Begriffe Abhängigkeit und Dienst werden ebenfalls nicht definiert. Man unterscheidet die Dienstfunktionalitäten nicht und ist bzgl. dem Lebenszyklus nicht leicht anwendbar.

3.3.2 IBM Tivoli Enterprise Console

Ein zweites in der Industrie verwendetes Tool ist IBM Tivoli Enterprise Console [ITEC]. Sie bringt folgende Vorteile: eine anspruchsvolle und automatisierte Problemdiagnose, Verbesserung der Systemleistung und Verringerung der Supportkosten. Die neueste Weiterentwicklung konzentriert sich auf die Benutzung von Best-Practice vordefinierten Modelle zur Vereinfachung und Beschleunigung der Entwicklung. Die Auto-Discovery Funktion ermöglicht das Verstehen der Umgebung und der zusammenhängenden Prozesse. Das Wertvolle für das Ereignismanagement ist, dass man Ereignisse sehr einfach filtern kann und die Analyse der grundlegenden Fehlerusache als auch ihre Auflösung ermöglicht wird. Eine andere Funktion der IBM Tivoli Enterprise Console ist die automatische Korrelation der Systemprobleme zu der Infrastruktur, so dass die grundlegende Fehlerusache ermittelt werden kann. Es kann auf mehreren Plattformen benutzt werden (HP-UX, Linux, Solaris, Windows NT, Windows 2000, Red Hat, SuSE).

Bewertung Die IBM Tivoli Enterprise Console ist ein sehr komplexes Tool. Sie ist kostenpflichtig, beinhaltet Abhängigkeiten auf der Ebene der Infrastruktur, aber auch Korrelationen mit den obenliegenden Diensten. Es werden aber das Konzept der Abhängigkeit und des Dienstes nicht definiert. Es ist mehr ein praktisches Tool als ein wissenschaftlicher Ansatz. Wie HP OV Service Navigator ist dem Nutzer der Algorithmus der als Basis für die grafische Darstellung der Abhängigkeiten und Impact- und Root Cause-Analyse liegt, verborgen.

3.4 Zusammenfassung

In diesem Kapitel wurden einige vorhandene Ansätze präsentiert, die wichtig für die Abhängigkeitsmodellierung sind und die in Zusammenhang mit dieser Thematik stehen. Wie man sehen konnte sind nicht alle, die man sich bei dieser Aufgabe vorgenommen hat, für die Modellierung geeignet.

In der Tabelle 3.1 wird ein Überblick über die Erfüllbarkeit der Anforderungen, bezüglich jedes Ansatzes, gezeigt. Die Tabelle ist, nach den Arten von Anforderungen, in zwei Bereiche aufgeteilt. Zu bemerken ist, dass bei den meisten Ansätzen eine teilweise positive Einschätzung im unteren Bereich (abhängigkeitbezogene Anforderungen) existiert, allerdings im oberen Bereich (dienstbezogene Anforderungen) weniger. Wenn die dienstbezogenen Anforderungen nicht erfüllt sind, ist es schwierig weiter auf diesem Ansatz aufzubauen. Bei den Ansätzen aus der Industrie ist es sogar so dass der dienstbezogene Bereich sehr schwach ausgeprägt ist; der abhängigkeitsbezogene aber größtenteils erfüllt ist. Diese Tatsache trägt aber trotzdem nicht viel zu dieser Arbeit bei; weil in Abwesenheit eines erweiterbaren Modells, eines allgemein definierten Dienstbegriffes oder wenn der Lebenszyklus und die QoS-Parameter des Dienstes nicht in Anspruch genommen werden, kein allgemeines Konzept der Abhängigkeiten definiert werden kann.

Auf dieser Grundlage wurde das MNM-Dienstmodell für eine weitere Verfeinerung ausgewählt. Das MNM-Dienstmodell ist gut erweiterbar. Es erfüllt die dienstbezogenen Anforderungen und man kann, auf diesen Fakten basierend, sehr gut im Bereich der Abhängigkeiten darauf aufbauen. Die Idee ist aber auch andere wichtige Informationen von den anderen Ansätze zu benutzen, um eines allgemeines Modell zu entwerfen.

vorhandene Ansätze / Anforderungen	MNM-Dienstmodell	Abhängigkeitshierarchien	CIM	Abhängigkeitsgraphen	Abhängigkeitsgrade	Verfügbarkeit	Internet Management	ITIL	eTOM	HP OV Service Navigator	IBM Tivoli EC
Dienstbezogen	**++**	**0**	**-**	**-**	**- -**	**0**	**0**	**+**	**0**	**-**	**0**
Technologieunabhängigkeit	++	++	++	++	++	++	+	++	+	-	-
Allgemeinheit des Dienstbegriffes	++	+	- -	-	0	-	0	+	+	-	-
Lebenszyklus	++	-	- -	- -	- -	+	- -	+	-	-	-
Dienstfunktionalitäten	++	-	-	- -	- -	0	- -	0	+	-	0
Dienstparameter	0	0	+	0	- -	+	+	0	+	+	+
Erweiterbarkeit	++	+	0	0	0	0	-	0	+	-	0
Anwendbarkeit	+	+	+	-	+	0	+	-	-	+	+
Dienstgüte	++	-	-	0	- -	0	-	+	+	0	0
Abhängigkeitbezogen	**-**	**++**	**-**	**-**	0	**-**	**-**	**0**	**-**	**+**	**+**
Abhängigkeiten zwischen Ressourcen	- -	++	++	+	0	-	+	0	0	+	+
Abhängigkeiten zwischen Ressourcen und Dienste	0	++	+	-	0	-	0	+	+	+	+
Abhängigkeiten zwischen Dienste und Subdienste	0	++	-	- -	+	0	+	0	0	+	+
Dienstfunktionalität spez. Abhängigkeiten	- -	0	- -	- -	- -	- -	- -	0	0	0	+
Statische Abhängigkeiten	0	++	-	+	+	+	-	0	+	+	+
Dynamische Abhängigkeiten	-	-	-	-	-	- -	- -	- -	- -	-	0
Redundanzen	- -	+	-	+	0	0	- -	-	-	0	0

Legende:

Anforderung ist:

++	völlig erfüllt
+	erfüllt
0	teilweise erfüllt
-	limitiert erfüllt
- -	nicht oder unzureichend erfüllt

Tabelle 3.1: Überblick der vorhandenen Ansätze bzgl. den Anforderungen

Kapitel 4

Modellierung der Abhängigkeiten

Inhaltsverzeichnis

Das MNM-Dienstmodell wurde in Abschnitt 3.1.1 sehr ausführlich beschrieben und die Schlussfolgerung davon war, dass dieses Modell sich als Grundlage zur Erweiterung eignet. Dieses Kapitel befasst sich mit der Verfeinerung des MNM-Dienstmodells und seiner Instantiierungsmethodik. Das Ziel ist es, ein allgemeines Modell für Abhängigkeiten daraus zu entwickeln. Es gelten dabei die Definitionen aus dem MNM-Dienstmodell [GHHK 01]. In Abschnitt 4.1 wird zunächst gezeigt, wie durch die Verfeinerung mit der MNM Dienstmodellierungsmethodik Abhängigkeiten gefunden werden. In Abschnitt 4.2 wird aufgrund der bestimmten Abhängigkeiten ein Abhängigkeitsmodell vorgeschlagen.

4.1 Verfeinerung des MNM-Dienstmodells

Die Instantiierungsmethodik aus [GHHK 02], die im Abschnitt 3.1.1.5 beschrieben worden ist, benennt zwei Varianten von Anwendungen. In der ersten Variante wird der *Top Down-Arbeitsablauf* des Service Views (Use Case- und Aktivitätsdiagrammen für jede Funktionalität) und gefolgt vom *Bottom Up-Arbeitsablauf* des Realization Views realisiert (Erweiterung der bestehenden Aktivitätsdiagramme und Erstellung von Kollaborationsdiagrammen). In der zweiten wird der *Bottom Up-Arbeitsfluss* des Realization Views(Anwendungsfälle für die Realisierung herausfinden und für jede davon Kollaborationsdiagrammen erstellen) gefolgt vom *Top Down-Arbeitsablauf* des Service Views (aus der bestehenden Kollaborationsdiagrammen Aktivitätsdiagrammen erstellen) dargestellt.

Für diese Arbeit wird der erste Ansatz benutzt. Demnach muss zuerst der Service View und anschließend der Realisation View anhand der Aktivitätsdiagramme bzw. Kollaborationsdiagramme verfeinert werden. Zunächst wird in Abschnitt 4.1.1 der Service View anhand der Anwendungsfall- bzw. Aktivitätsdiagramme verfeinert. In Abschnitt 4.1.2 wird der Realization View verfeinert, indem die Aktivitätsdiagramme erweitert und Kollaborationsdiagramme für jede der Funktionalitäten erstellt werden.

4.1.1 Verfeinerung des Service Views

Dieser Abschnitt behandelt als ersten Schritt des *Top Down*-Ansatzes die Verfeinerung des Service Views. Es werden daraufhin, für jede der Funktionalitäten, die der Dienst besitzt, Use-Case Diagramme erstellt. Die in diesen Diagrammen dargestellten Anwendungsfälle werden danach in Aktivitätsdiagrammen noch genauer beschrieben. In Abschnitt 4.1.1.1 wird die Verfeinerung der Funktionalitäten aufgezeigt. In Abschnitt 4.1.1.2 folgt ihre Einschätzung bezüglich der Modellierung von Abhängigkeiten.

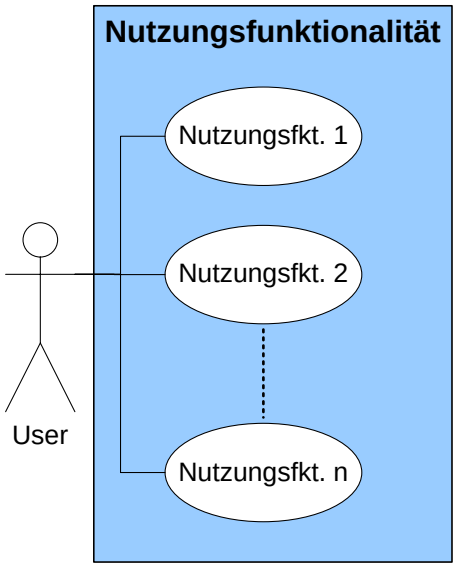

Abbildung 4.1: Die Nutzungsfunktionalität

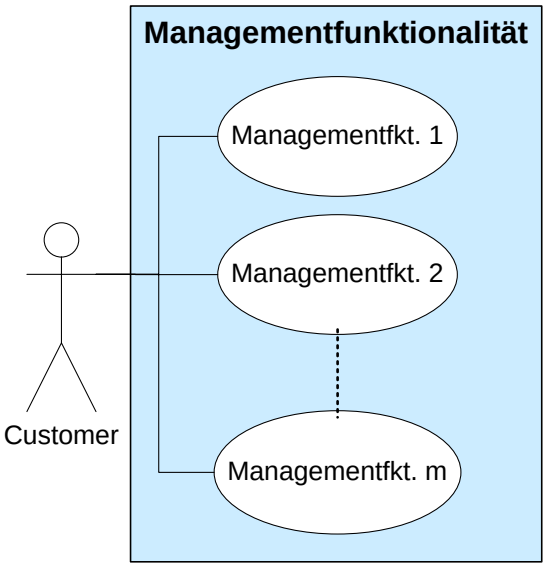

Abbildung 4.2: Die Managementfunktionalität

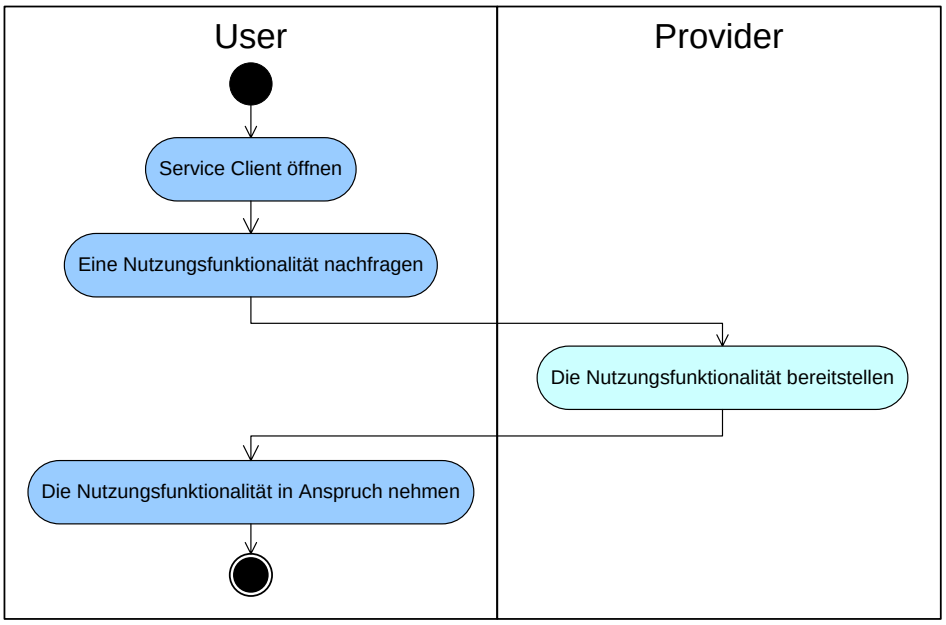

Abbildung 4.3: Eine Nutzungsfunktionalität als Aktivitätsdiagramm

4.1.1.1 Verfeinerung der Funktionalitäten

Die Funktionalitäten eines Dienstes werden allgemein in zwei Klassen eingeteilt (wie in 3 beschrieben). Es gibt also die zwei großen Interaktionsklassen Nutzung und Management und dadurch auch die zwei Kategorien, Nutzungsfunktionalität und Managementfunktionalität, von Funktionalitäten. Danach werden sie in Anwendungsfalldiagrammen dargestellt (wie in den Abbildungen 4.1 und 4.2). Man erhält dadurch einen klaren Überblick über die möglichen Funktionalitäten. Des Weiteren definiert man die Rollen, die diese Funktionalitäten in Anspruch nehmen. Im Falle der Nutzungsfunktionalität die Rolle des Users und im Falle der Managementfunktionalität die Rolle des Customers oder eventuelle Verfeinerungen davon wie z.B. Student, LRZ, Mitarbeiter, Institut usw.

Für jede der Nutzungsfunktionalitäten **Nutzungsfkt. 1** bis **Nutzungsfkt. n** und der Managementfunktionalitäten **Managementfkt. 1** bis **Managementfkt. m** werden nach der Methodik in [GHHK 02] in diesem Abschnitt Aktivitätsdiagramme erstellt.

Es wird weiterhin zwischen den zwei grundsätzlichen Arten von Funktionalitäten unterschieden. Die Nutzungsfunktionalitäten, die in der Abbildung 4.1 dargestellt sind, werden von der Rolle Provider (Anbieter) bereitgestellt und von der Rolle User (Nutzer) in Anspruch genommen. Auf der anderen Seite werden die Managementfunktionalitäten (siehe Abbildung 4.2) vom Provider (Anbieter) bereitgestellt und vom Customer (Kunde) in Anspruch genommen. Anhand dieser Rollen werden dann in den Aktivitätsdiagrammen jeder Funktionalität auch die Verantwortlichkeitsbereiche abgegrenzt.

Abbildug 4.3, zeigt wie ein User eine bestimmte Nutzungsfunktionalität des Dienstes X in Anspruch nehmen möchte. Als erstes wird der Service Client geöffnet, da laut des MNM-Dienstmodells der Nutzer eine Funktionalität über einen Service Client in Anspruch nimmt. Die zweite Aktion ist demnach das Anfragen nach einer bestimmten Nutzungsfunktionalität. Der Provider stellt die nachgefragte Nutzungsfunktionalität bereit, aber wie er das tut, ist für den Nutzer nicht relevant. Der Nutzer bekommt dann die Rückmeldung vom Provider und kann die gewünschte Funktionalität in Anspruch nehmen.

Analog der Abbildung 4.3 möchte der Customer (Kunde) eine Managementfunktionalität benutzen. Er öffnet den CSM Client, weil im MNM-Dienstmodells der Customer eine Managementfunktionalität über einen CSM Client in Anspruch nimmt. Über diesen fragt er nach der gewünschten Managementfunktionalität beim Provider. Der Provider stellt diese Managementfunktionalität bereit und danach wird die Anfrage erwidert. Dabei ist es nicht wichtig wie der Provider die Funktionalität bereitstellt sondern nur das er sie bereitstellt.

4.1.1.2 Modellierung der Service View-spezifischen Abhängigkeiten

Bei den obigen Darstellungen kann man hier sehr schwer Abhängigkeiten erkennen. Man kann „vermuten“, dass sich Abhängigkeiten im Hintergrund verbergen, aber welche es sind, kann man nicht genau sagen. Die im Abschnitt 2.5 erstellte Anforderungsliste sollte jedoch für jede der Funktionalitäten wie in der Tabelle 4.1 durchgeführt werden.

Anforderungen	Bewertung
Abhängigkeiten zwischen Ressourcen	nicht erfüllt
Abhängigkeiten zwischen Ressourcen und Diensten	nicht erfüllt
Abhängigkeiten zwischen Diensten und Subdiensten	nicht erfüllt
Dienstfunktionalität-spez. Abhängigkeiten	nicht erfüllt
Statische Abhängigkeiten	nicht erfüllt
Dynamische Abhängigkeiten	nicht erfüllt
Redundanzen	nicht erfüllt

Tabelle 4.1: Erfüllung der Anforderungen für die Modellierung durch den Service View

Abhängigkeiten zwischen Ressourcen untereinander, zwischen Ressourcen und Diensten und sogar zwischen Diensten untereinander können aus den Aktivitätsdiagrammen nicht bestimmt werden, denn der Nutzer fragt nach einer bestimmten Funktionalität und nimmt sie danach in Anspruch. Was sich aber dahinter verbirgt, ist nicht ersichtlich. Die Dienstfunktionalität-spezifischen Abhängigkeiten könnten in einzelnen Fällen sichtbar sein. Im Allgemeinen aber kann diese ausschließlich mithilfe der Aktivitätsdiagramme für den Service View nicht gefunden werden. Statische und dynamische Abhängigkeiten als auch Redundanzen sind aus den bestehenden Diagrammen unmöglich herzuleiten, weil sie anbieterspezifisch sind.

Aus diesem Grund können Abhängigkeiten nicht alleine durch das Service View bestimmt werden. Von Fall zu Fall kann die eine oder andere Anforderung limitiert erfüllt sein, aber im Allgemeinen ist die obige Darstellung unzureichend für die Abhängigkeitsmodellierung. Das wird im Abschnitt 4.1.2 dargestellt. Die Aktivitätsdiagramme müssen erweitert werden, indem die interne Realisierung des Dienstes berücksichtigt wird.

4.1.2 Verfeinerung des Realization-Views

In der Realisierungssicht sind Dienstimplementierung und -management zu verfeinern. Für die providerinterne Realisierung des Dienstes ist wichtig, von welchen Ressourcen und Sub-Diensten die Realisierung jeder beschriebenen Nutzungsfunktionalität abhängig ist. Auf der Managementseite sind

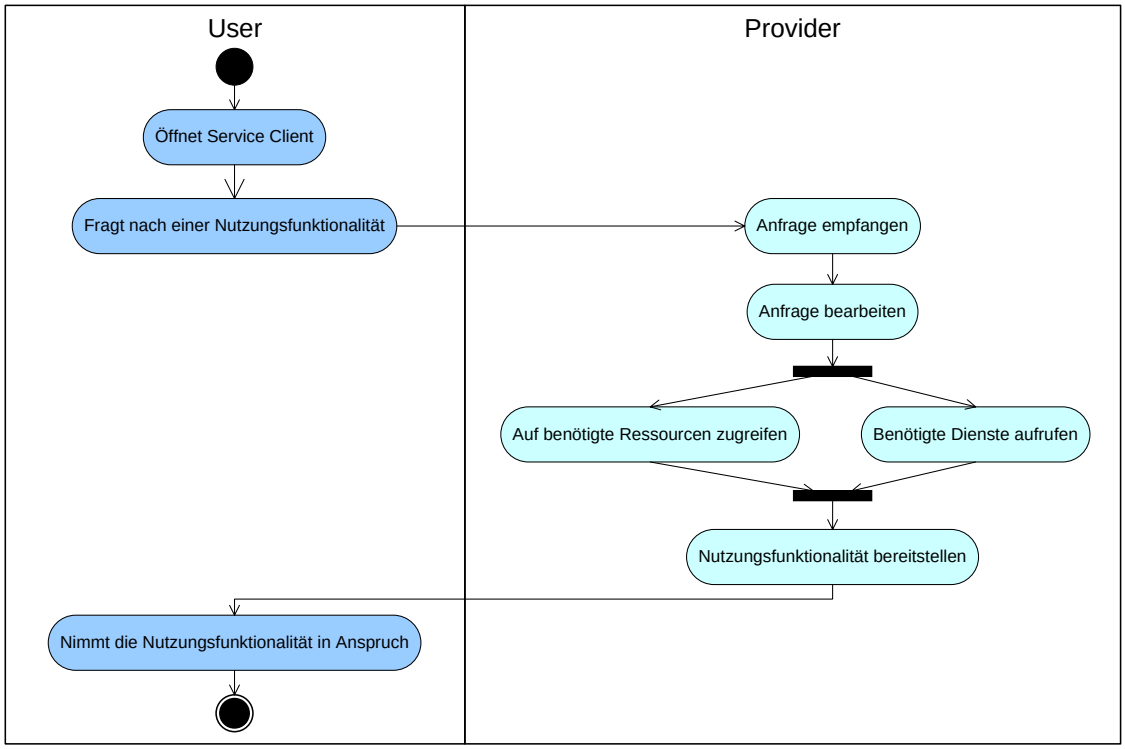

Abbildung 4.4: Aktivitätsdiagramm einer allgemeinen Nutzungsfunktionalität (Service Implementation)

die Abhängigkeiten zwischen BMF bzw. Management von Sub-Diensten und die Managementfunktionalitäten von Interesse. Dies wird jeweils durch Service Logic bzw. Service Management Logic implementiert.

Auf Basis der schon entwickelten Aktivitätsdiagramme, für die Funktionalitäten aus Customer-Sicht, wird die interne Realisierung des Dienstes dargestellt. Die Methodik in [GHHK 02] nennt als nächsten Schritt, in der Analyse des Dienstes, die Erweiterung jedes im Service View spezifizierten Aktivitätsdiagrammes. Danach werden für alle Fälle jeweils Kollaborationsdiagramme erstellt.

4.1.2.1 Verfeinerung der Service Implementation und des Service Managements

Die Abbildung 4.3 wird als Basis für die Darstellung der internen Realisierung der Nutzungsfunktionalität, im MNM-Dienstmodell auch Service Implementation genannt, benutzt. Die Verantwortlichkeitsbereiche, die in Betracht gezogen werden müssen, sind identisch mit den Rollen, die in den bestimmten Funktionalitäten impliziert sind: User und Provider.

Der User möchte eine der Nutzungsfunktionalitäten nutzen (Abbildung 4.4). Daraufhin stellt er eine Anfrage nach dieser Nutzungsfunktionalität. Diese Anfrage wird vom Provider empfangen und daraufhin bearbeitet. Es wird auf die benötigten Ressourcen zugegriffen und gleichzeitig werden benötigte (Sub)Dienste aufgerufen. Das Zusammenspiel zwischen den Ressourcen und (Sub)Diensten trägt dazu bei, dass diese bestimmte Nutzungsfunktionalität realisiert wird. Diese Nutzungsfunktionalität kann jetzt von dem Nutzer in Anspruch genommen werden.

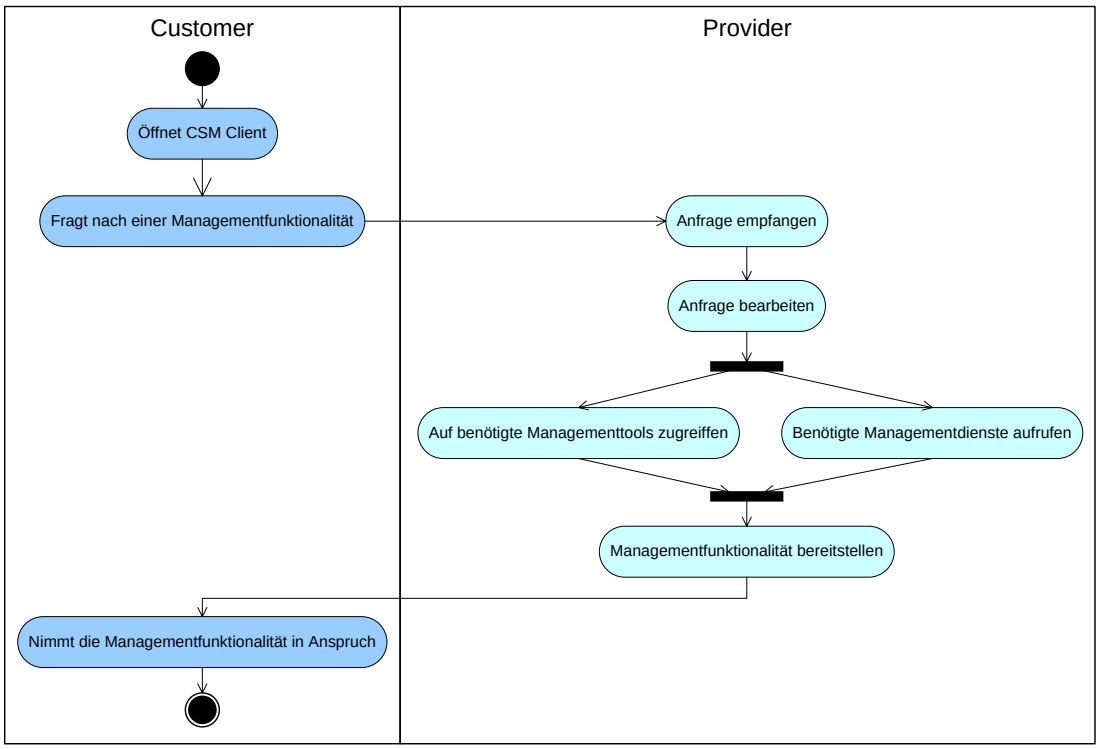

Abbildung 4.5: Aktivitätsdiagramm einer allgemeinen Managementfunktionalität (Service Management)

Die Realisierung der Managementfunktionalität wird in der Abbildung 4.5 als Aktivitätsdiagramm dargestellt. Die Rollen, die bei der Managementfunktionalität beteiligt sind, werden zu Verantwortlichkeitsbereichen Customer und Provider in dem Aktivitätsdiagramm.

Die interne Realisierung der Managementfunktionalität wird auf der Providerseite realisiert und wird vom Customer auf der Kundenseite in Anspruch genommen. Der Customer, der auf der Kundenseite für das Management des Dienstes zuständig ist, möchte eine Managementfunktionalität benutzen. Er fragt beim Provider an, diese Funktionalität jetzt in Anspruch nehmen zu können. Die Anfrage wird vom Provider empfangen und danach bearbeitet. Zu diesem Zeitpunkt werden dann die providereigenen Managementtools (Basic Management Funktionalität) in Anspruch genommen. Gleichzeitig werden eventuell benötigte (Sub)Managementdienste aufgerufen. Dadurch wird die Managementfunktionalität realisiert, so dass der Kunde seine Managementaktion ausführen kann.

4.1.2.2 Modellierung der Realization View-spezifischen Abhängigkeiten anhand von Aktivitätsdiagrammen

Dadurch, dass die Aktivitätsdiagramme für die interne Realisierung des Dienstes viel umfangreicher sind als die der Funktionalitäten im Service View sind, kann man einige Abhängigkeiten ablesen. Man kann hier schon feststellen, dass bestimmte Ressourcen und bestimmte (Sub)Dienste bei der Realisierung einer Funktionalität mitwirken. Nichtsdestotrotz ist dies immer noch eine zu oberflächliche Darstellung, weil die speziellen Beziehungen zwischen unterschiedlichen Diensten und Ressourcen, als auch die Redundanzen nicht eindeutig eingeordnet werden können.

Anforderungen	Bewertung
Abhängigkeiten zwischen Ressourcen	nicht erfüllt
Abhängigkeiten zwischen Ressourcen und Diensten	unzureichend erfüllt
Abhängigkeiten zwischen Diensten und Subdiensten	limitiert erfüllt
Dienstfunktionalität-spez. Abhängigkeiten	limitiert erfüllt
Statische Abhängigkeiten	limititert erfüllt
Dynamische Abhängigkeiten	nicht erfüllt
Redundanzen	nicht erfüllt

Tabelle 4.2: Erfüllung der Anforderungen für die Modellierung der Abhängigkeiten durch Aktivitätsdiagramme

In der Tabelle 4.2 wird eine Übersicht darüber geschaffen, wie die in den Aktivitätsdiagrammen dargestellten Abhängigkeiten die Anforderungen aus Abschnitt 2.5 erfüllen. Die dienstspezifischen Abhängigkeiten sind limititert erfüllt, da das Diagramm den dienstfunktionalitätspezifischen Ablauf wiedergibt, aber nicht dessen detaillierte Beschreibung bzw. Darstellung. Die Abhängigkeiten zwischen den Ressourcen und Diensten kann man zumindest erahnen, sie sind jedoch nur angedeutet, aber nicht richtig beschrieben. Die Abhängigkeiten zwischen Diensten und Subdiensten und insbesondere die statischen Abhängigkeiten sind sichtbar, aber auch nicht weiter darstellbar. Für die restlichen Abhängigkeiten und für die Redundanzen gibt es allerdings keine Modellierungsmöglichkeit.

Wie aus der Tabelle 4.2 ersichtlich, sind die Aktivitätsdiagramme für den Realization View viel umfangreicher und detaillierter bzgl. der Abhängigkeiten, aber noch nicht ausreichend genug. Einige der Anforderungen sind unzureichend oder limitiert erfüllt, aber die Abhängigkeiten zwischen Ressourcen, die dynamischen Abhängigkeiten, und die Redundanzen können mit dieser Art von Diagrammen nicht detailliert dargestellt werden. Aus diesem Grund wird eine Erweiterung bzw. Verfeinerung der obenbeschriebenen Aktivitätsdiagramme benötigt, nämlich Kollaborationsdiagramme.

4.1.2.3 Verfeinerung der Service Logic und der Service Management Logic

Die Service Logic bzw. Service Management Logic sind diejenigen Komponeten des MNM-Dienstmodells, die für die interne Realisierung des Dienstes und darauf implizit auch für die Realisierung der Nutzungs- und Managementfunktionalität zuständig sind. Die Darstellung dessen, was bei der Realisierung eines Dienstes tatsächlich passiert, ist der Gegenstand der Servicelogik bzw. der Servicemanagementlogik und wird in Kollaborationsdiagrammen wiedergespiegelt.

Ein Kollaborationsdiagramm beinhaltet Dienste und Ressourcen als Objekte in UML-Repräsentation sowie die bestimmten Kommunikationen, die zwischen diesen bei der Realisierung einer Funktionalität bestehen. Zusätzlich zu diesen werden der Client und der Access Point repräsentiert, da der Ablauf der Dienstfunktionalität vom Client initiiert, vom Access Point angenommen und weiter zu den entsprechenden Diensten oder Ressourcen weitergeleitet wird. Ein Beispiel für ein Kollaborationsdiagramm ist in der Abbildung 4.6 dargestellt. In diesem Diagramm werden als Objekte n Dienste und m Ressourcen als auch deren Beziehungen repräsentiert. Der Client schickt eine Nachricht zum AccessPoint, um eine bestimmte Dienstfunktionalität in Anspruch nehmen zu können. Der Access Point leitet sie zu der ersten Komponente, die für die Realisierung der Funktionalität zuständig ist, weiter. Der **Dienst 1** schickt eine Nachricht zur **Ressource 1** und wartet dann auf die Antwort von

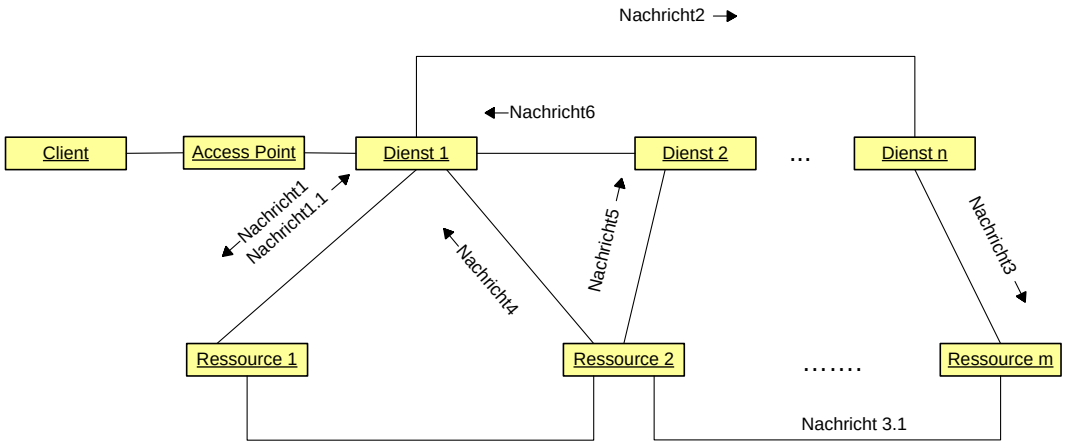

Abbildung 4.6: Kollaborationsdiagramm für die interne Realisierung des Dienstes

dieser. Wenn die Antwort (durch die **Nachricht 1.1**) zurückgekommen ist, kann er weiter eine **Nachricht 2** zu **Dienst n** schicken, der darauf mit einer **Nachricht 3** an **Ressource m** antwortet. Abhängig von **Nachricht 3** reagiert dann die sofort folgende **Nachricht 3.1** an **Ressource 2** usw.

Durch diese Art von Diagrammen hat man einen sehr umfangreichen Überblick über das, was bei der Realisierung des Dienstes bzw. der Dienstfunktionalität stattfindet. Für den Fall, dass das Bild unübersichtlich werden sollte, empfiehlt es sich, einige Abläufe auszugliedern.

4.1.2.4 Modellierung der Realization View-spezifischen Abhängigkeiten anhand von Kollaborationsdiagrammen

Die Kollaborationsdiagramme sind ein sehr wertvolles Mittel zur Repräsentierung der Abhängigkeiten (wie auch in Tabelle 4.3 dargestellt). Abhängigkeiten zwischen den Ressourcen, Abhängigkeiten zwischen Diensten und Ressourcen und zwischen Diensten und Subdiensten besonderes die statische Abhängigkeiten können mit Hilfe der Kollaborationsdiagramme entsprechend den Anforderungen im Abschnitt 2.5 hervorragend modelliert werden. Aus diesen Kollaborationsdiagrammen kann man alle Abläufe und die damit zusammenhängenden Komponenten aufs genaueste ablesen. Die dienstfunktionalitätspezifische Abhängigkeiten sind ebenfalls sehr gut damit repräsentierbar, denn es entsteht für jede Funktionalität ein Kollaborationsdiagramm. Man kann also für jede Funktionalität die entsprechenden spezifischen Abhängigkeiten herausfinden.

Es bleibt aber immer noch ein Teil der Anforderungen, die nur teilweise (dynamische Abhängigkeiten) oder gar nicht (Redundanzen) erfüllt werden. Für die dynamischen Abhängigkeiten könnte man sehr viele Kollaborationsdiagramme erstellen, oder in dem Ablauf unterschiedliche Testoperationen einfügen (Verzweigungspunkte), um jeden einzelnen auftretenden Fall abzudecken, aber aus Modellierungssicht ist das zu umständlich.

Obwohl diese Darstellung eine sehr gute Grundlage für Abhängigkeitsmodellierung bildet, werden weiter einige Aspekte zur Behebung der obengenannten Defizite untersucht. Zum Beispiel können die dynamischen Abhängigkeiten nicht vollständig in Abwesenheit der Modellierung der Redundanzen modelliert werden. Die Dienstfunktionalität-spezifischen Abhängigkeiten werden erst dann komplett

Anforderungen	Bewertung
Abhängigkeiten zwischen Ressourcen	vollständig erfüllt
Abhängigkeiten zwischen Ressourcen und Diensten	vollständig erfüllt
Abhängigkeiten zwischen Diensten und Subdiensten	vollständig erfüllt
Dienstfunktionalität-spezif. Abhängigkeiten	vollständig erfüllt
Statische Abhängigkeiten	vollständig erfüllt
Dynamische Abhängigkeiten	teilweise erfüllt
Redundanzen	nicht erfüllt

Tabelle 4.3: Erfüllung der Anforderungen für die Modellierung der Abhängigkeiten durch Kollaborationsdiagramme

definiert, wenn alle Dienstfunktionalitäten in Kollaborationsdiagrammen dargestellt und die Beziehungen zwischen ihnen ersichtlich sind.

4.1.2.5 Darstellung von Redundanzen

Für die Darstellung der Redundanzen wurden ebenfalls Kollaborationsdiagramme ausgewählt, weil diese die Zwecke der Abhängigkeitsmodellierung am besten erfüllen. Die existierenden Redundanzen in der Infrastruktur bzw. Dienstredundanzen werden in einem Kollaborationsdiagramm dargestellt.

In der Abbildung 4.7 wird ein allgemeines Beispiel dargestellt von **m** redundanten Ressourcen, die zur Realisierung einer Dienstfunktionalität beitragen. Die Kommunikation zwischen jeder der Ressourcen und die Dienstfunktionalität sind identisch. Angenommen, zwischen Kunde und Anbieter wurde folgendes SLA festgelegt: eine Verfügbarkeit von 24/7, d.h. 24 Stunden am Tag, 7 Tage der Woche. Wie wirkt sich das auf der interne Realisierung des Dienstes aus. Obwohl diese Ressourcen identisch konfiguriert sind, ist es trotzdem in Bezug auf die SLAs nicht dasselbe, ob alle **m** Komponenten einwandfrei laufen, oder ob nur 50% davon funktionsfähig sind. Die Auslastung der Komponenten ändert sich dadurch. Auch wenn der Dienst nicht ausfällt, hat trotzdem der Anbieter intern überlastete Komponente, die sehr schnell zu einer möglichen Störung führen könnten.

Dieses Problem steht eng in Zusammenhang mit der Formulierung des SLAs. In dem oberen SLA wurde eine Verfügbarkeit von 24/7 festgelegt. Das ist ernster zu nehmen, wie eine 97% Verfügbarkeit. Wenn als Folge der Überlastung der Dienst ausfällt, ist das erste SLA verletzt aber das zweite noch nicht. Das heißt in dem zweiten Fall, wenn der Dienst schnell genug wiederherstellt wird, wird das SLA nicht verletzt. In Zusammenhang mit dieser Problematik stellen sich folgende Fragen: Was für eine Verletzung der SLAs kann eine überlastete Komponente verursachen? Wieviele Komponenten müssen ausfallen, um eine Verletzung der SLAs hervorzurufen? Das alles wird in der Provisioning- bzw. Negotiation-Phase festgelegt.

Wenn z.B. 50% der Ressourcen von einer Störung betroffen sind, muss sich die Last, die zuvor in Normallfall auf 100% der Komponenten verteilt war, auf die noch verbleibenden 50% verteilen. Das heißt, dass die Auslastung der Komponenten sich verdoppelt hat.

Die Servicelogik bestimmt, wie mehrere redundante Komponenten für die Erbringung des Dienstes eingesetzt werden müssen. Hierfür führt man in dieser Arbeit eine Zahl **c** ein (c von Control), die für jedes System von redundanten Komponenten bestimmt wird. Bei Überschreitung dieser Zahl kann

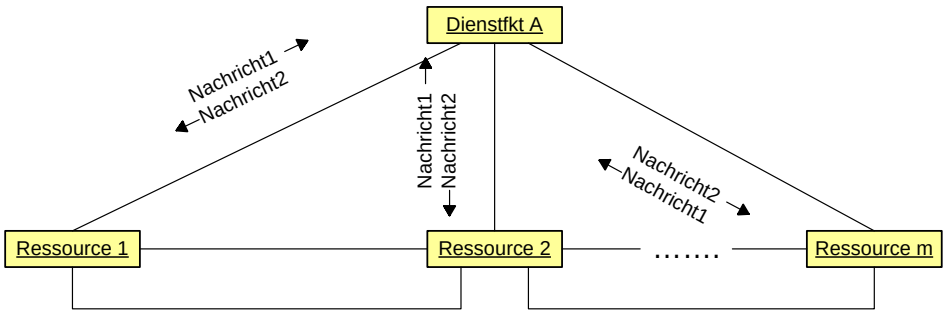

Abbildung 4.7: Kollaborationsdiagramm für die Ressourcenredundanz

einen Alarm auslösen, obwohl noch keine Fehlermeldung in der Infrastruktur aufgetreten ist, die zeigt, dass die redundanten Komponenten nicht mehr mit höchster Leistung arbeiten. Diese Zahl **c** ist nicht nur ein Indikator für die Verfügbarkeit der Komponenten, sondern auch ein Kennzeichen der Auslastung, die bei den Komponenten möglich ist, ohne dass Funktionsprobleme bzw. Performanceprobleme auftreten.

Im Grunde genommen existieren in einem redundanten System mit **m** Komponenten (**1** bis **m**) **m** Abhängigkeiten. Man weiß jedoch nie genau, welche der Komponente gerade genutzt wird. Wenn die Komponente **Ressource p** ausfällt, springt die Komponente **Ressource q** automatisch ein oder wird manuell eingeschaltet. Wenn mehrere Komponente ausfallen, dann übernehmen diejenigen, die noch funktionsfähig sind, die Funktionalität der ausgefallenen. Hier wird die Notwendigkeit der Zahl **c** bewiesen: die restlichen Komponeten übernehmen solange die Funktion der ausgefallenen Komponenten, wie der Auslastwert die Zahl **c** nicht überschreitet. Dieser Ansatz ist wichtig nicht nur für die Bestimmung der Redundanzen sondern auch der dynamischen Abhängigkeiten.

Mit diesem Zusatz über die Redundanzen wurde das in der Tabelle 4.3 gezeigte Defizit gemildert. Offen bleibt die Anforderung „dynamische Abhängigkeiten“ aber auch die Notwendigkeit eines allgemein anwendbaren Abhängigkeitsmodell, welches einerseits alle Abhängigkeiten gleich betrachten kann (Allgemeinheit), aber andererseits jede einzelne Abhängigkeit separat (Spezifizität).

4.2 Das Abhängigkeitsmodell

In den vorherigen Abschnitten wurde der Schwerpunkt auf das Bestimmen der Abhängigkeiten gelegt. Zur Bestimmung der Abhängigkeiten wurde die Methodik des MNM-Dienstes verwendet. Jedes andere Tool oder jede andere Methodik, zur Bestimmung der Abhängigkeiten, kann hier ebenfalls verwendet werden. Die gefundenen Abhängigkeiten werden mit dem im Folgenden vorgestellten Modell einheitlich modelliert. Im Abschnitt 4.2.1 wird das Composite Pattern mit den entsprechenden Eigenschaften, die zur Modellierung in Betracht gezogen werden müssen, vorgestellt. Es folgt in Abschnitt 4.2.2 eine Beschreibung der Hilfsklassen *ServiceBuildingBlock* und *Functionality* und anschließend, im Abschnitt 4.2.3, eine umfangreiche Beschreibung des entwickelten *Dependency*-Modells.

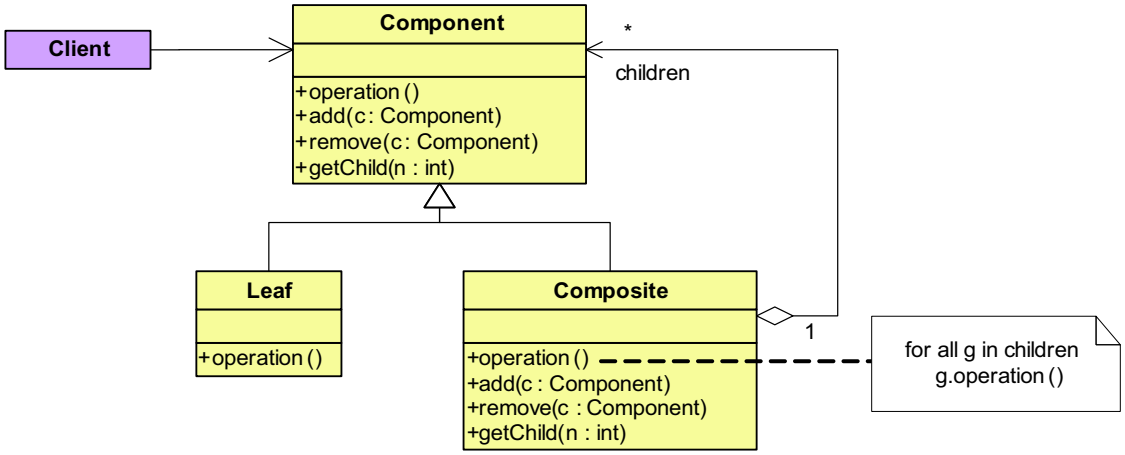

Abbildung 4.8: Das Composite Pattern [Gamma 95]

4.2.1 Das Composite Pattern - Allgemeinbeschreibung

Die Kollaborationsdiagramme sind ein sehr hilfreiches Mittel, um Abhängigkeiten zu bestimmen. Allerdings kann ein Kollaborationsdiagramm wegen den sehr vielen beteiligten Komponenten unübersichtlich werden. Hinzu kommt noch, dass die Abhängigkeiten miteinander in Beziehung stehen. Daher entsteht die Notwendigkeit, eine einfache Struktur zur Darstellung der Abhängigkeiten zu entwickeln.

Design Patterns beschreiben einfache und elegante Lösungen für manche Probleme im Softwaredesign. Diese Eigenschaften des Design Pattern wurden in Betracht gezogen für die Realisierung des Abhängigkeitsmodells, das einfach und leicht anwendbar sein muss. Als Basis für die Modellierung wurde der Strukturelle Design Pattern Composite [Gamma 95] gewählt. Im Folgenden werden die begriffe Komponente (Component) und Client im Sinne des Composite Patterns verwendet und nicht als Begriffe aus dem Netzbereich bzw. Dienstbereich. Der Zweck des Strukturellen Pattern Composite ist die Anordnung von einzelnen Objekten in Baumstrukturen, um „Teil-Ganzes“-Hierarchien darzustellen. Das Composite-Pattern ermöglicht es dem Client, sowohl einzelne als auch zusammengesetzte Objekte einheitlich zu behandeln.

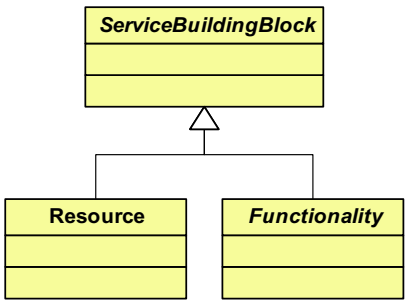

Abbildung 4.9: Die Klassenhierarchie für ServiceBuildBlock

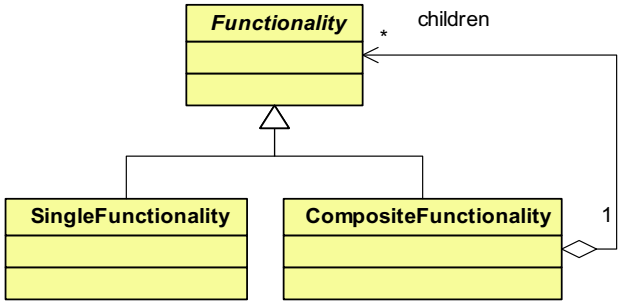

Abbildung 4.10: Das Composite Pattern für die Klasse *Functionality*

Das Composite Pattern (siehe Abbildung 4.8) besteht aus vier Komponenten:

- ***Component*** deklariert die Schnittstelle für die Objekte in der Composite-Klasse (zusammengesetzte Objekte); implementiert ein Standard-Verhalten für die Schnittstelle, das allen Klassen gemeinsam ist (wie benötigt); deklariert eine Schnittstelle für den Zugriff, verwaltet seine Kindkomponenten.
- ***Leaf (Blattkomponente***) repräsentiert Blatt-Objekte in den zusammengesetzten Komponenten und definiert das Verhalten für die Grundobjekte der Komposition.
- ***Composite (zusammengesetzte Komponente***) definiert das Verhalten für die Komponenten mit Kindern, speichert Kindkomponenten und implementiert die kindbezogenen Informationen in der *Component*-Schnittstelle.
- ***Client*** manipuliert Objekte in der Composition durch die Component Schnittstelle. Wird benutzt um verschiedenen Aktionen auf den Component-Objekte zu initiieren (z.B. Erstellung neuer Objekte, Änderung bestehender Objekte etc.).

Clients benutzen die Klasse Component, um mit den Objekten in der zusammengesetzten Struktur interagieren zu können. Wenn der Empfänger der Kommunikation ein *Leaf* ist, dann wird die Anfrage direkt bearbeitet. Wenn der Empfänger ein *Composite* ist, dann wird die Anfrage zu den Kindkomponenten weitergeleitet (möglicherweise können auch andere Operationen durchgeführt werden, bevor und/oder nachdem die Weiterleitung stattgefunden hat). Im Fall der Abhängigkeitsmodellierung in einem Netz kann der Client ein Management Tool sein mit dem die verschiedene Abhängigkeiten u.a. visualisiert, gespeichert, hinzugefügt werden können.

4.2.2 Die Klassen *ServiceBuildingBlock* und *Functionality*

Die Beobachtung ist, dass bei der Realisierung eines Dienstes die Interaktionen zwischen Komponenten betrachtet werden müssen. Diese Komponenten können Dienste, Subdienste, Funktionalitäten und Teilfunktionalitäten der Dienste aber auch Ressourcen sein. Die Funktionalitäten werden in zwei große Interaktionsklassen eingeteilt: Nutzung und Management. Da oben in dem Kollaborationsdiagrammen die Feststellung gemacht worden ist, dass diese zu unübersichtlich sind, ist die Idee entstanden, diese Komponenten in einfachen Strukturen zu organisieren. Der Vorteil dabei ist, dass die Komponenten, die zur Realisierung des Dienstes beitragen, durch eine gemeinsame Oberklasse, aber auch spezifisch durch Unterklassen repräsentiert werden.

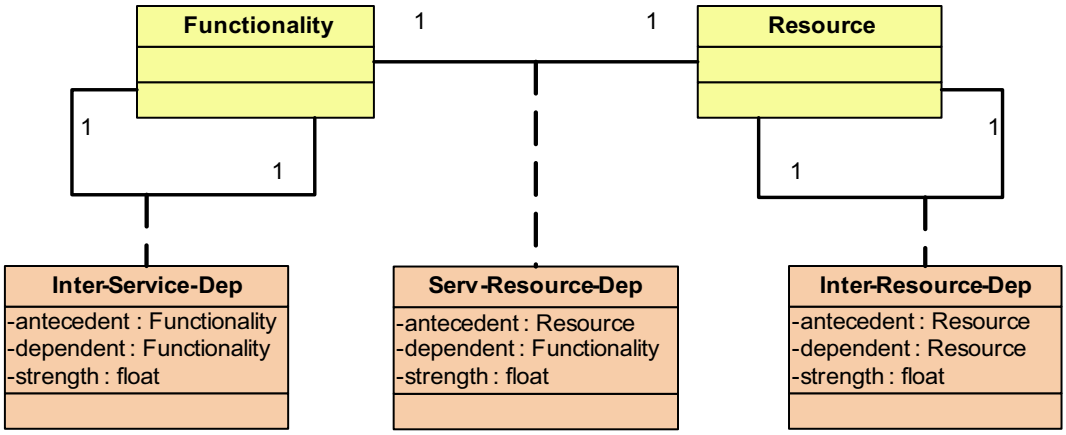

Abbildung 4.11: Mögliche Abhängigkeiten zwischen Ressourcen und Funktionalitäten

Im Folgenden werden die Begriffe *SercciceBuildingBlock*, was eine für die Dienstrealisierung benötigte Komponente, d.h. Dienst, Funktionalität oder Ressource repräsentiert, und *Functionality* als Verallgemeinerung für Dienst und Dienstfunktionalitäten beschrieben.

Der Begriff *ServiceBuildingBlock* repräsentiert im Allgemeinen eine Teilkomponente, die zur Realisierung des Dienstes beiträgt. Darunter fallen Dienste, Subdienste, Funktionalitäten, aber auch Ressourcen. Die *ServiceBuildingBlock*-Klasse ist eine abstrakte Oberklasse in der Hierarchie (siehe Abbildung 4.9) und repräsentiert alle Komponenten, die Teil einer Dienstrealisierung sind. Diese Oberklasse hat zwei Unterklassen, die von der *ServiceBuildingBlock*-Klasse die Eigenschaft erben, Komponenten zur Realisierung eines Dienstes zu sein. Diese zwei Unterklassen sind:

- ***Resource*** ist die Klasse aller Ressourcen (Hardware, Software, Mitarbeiter, Expertenwissen usw.)
- ***Functionality*** ist die Klasse der Funktionalitäten, deren Gesamtheit den Dienst ausmachen. Der Begriff *Functionality* repräsentiert eine Verallgemeinerung für Dienst, Dienstfunktionalität, Subdienst, Managementfunktionalität, Nutzungsfunktionalität. Sie ist aus der Notwendigkeit der vereinfachten Darstellung von Beziehungen zwischen diesen Komponenten entstanden. Das heißt, dass der Dienst als eine komplexe Funktionalität aus mehreren zusammengesetzten und/oder einfachen Funktionalitäten angesehen wird.

  Um die Klasse *Functionality* (siehe Abbildung 4.10) angemessen repräsentieren zu können wurde der Composite Pattern benutzt. Die Teile des Patterns sind die Blätter *SingleFunctionality* und das zusammengesetzte Objekt *CompositeFunctionality*, welches mehrere einzelne und/oder zusammengesetzte Funktionalitäten beinhalten kann. Das heißt, dass ein Dienst als eine (rekursiv) zusammengesetzte Funktionalität betrachten werden kann. Wie im Allgemeinfall des Composite Patterns beschrieben, können zusätzliche Funktionalitäten den zusammengesetzten Funktionalitäten hinzugefügt werden, und es können Teilfunktionalitäten (Single- und/oder CompositeFunctionality Objekte) vom Container entfernt werden.

Ein sehr wichtige Methode, die in dieser Klassen implementiert werden sollte, ist *getStatus(time)* was den Status des *ServiceBuildingBlock*-Objekts zu einen bestimmten Zeitpunkt zurückgibt. Dieser kann auch in der Vergangenheit liegen, wenn man die Funktionsfähigkeit der entsprechende Komponente zu einem früheren Zeitpunkt testen will. Diese Methode wird zusammen mit den Methoden des

Dependency-Patterns insbesondere für das Fehlermanagement (siehe Abschnitt 4.2.3.4) benutzt.

In der Abbildung 4.11 wird eine vereinfachte Darstellung der unterschiedlicher Arten von Abhängigkeiten zwischen Ressourcen untereinander, zwischen Ressourcen und Diensten bzw. Dienstfunktionalitäten und zwischen Diensten bzw. Dienstfunktionalitäten untereinander dargestellt. Die Klassen *Functionality* bzw. *Resource* sind wie oben definiert worden. Diese können voneinander abhängig sein, wie z. B. *Functionality* von *Functionality*, *Functionality* von *Resource* oder *Resource* von *Resource*.

Die oben beschriebenen Klassen stellen genau die drei Arten von Abhängigkeiten dar. Es werden dafür die Assoziationsklassen **Inter-Service-Dep** für die Abhängigkeiten zwischen Funktionalitäten im Allgemeinen, **Serv-Resource-Dep** für die Abhängigkeiten zwischen den Ressourcen und Funktionalitäten und **Inter-Resource-Dep** für die Abhängigkeiten zwischen Ressourcen dargestellt. Die Attribute und Methoden dieser Klassen werden im Abschnitt 4.2.3 ausführlich beschrieben.

Obwohl diese Darstellung schon viel übersichtlicher ist als zuvor, fehlt trotzdem noch ein wesentlicher Teil, und zwar die Zusammensetzung von mehreren vorkommenden Abhängigkeiten, und deren Beziehungen zueinander. Wie die Modellierung zu realisieren ist, wird im nächsten Abschnitt beschrieben.

4.2.3 Das *Dependency* - Pattern

Die Abbildung 4.11 stellt die möglichen Abhängigkeiten zwischen den unterschiedlichen *ServiceBuildingBlock*-Klassen dar. Zu beobachten ist aber, dass alle drei Assoziationsklassen, obwohl sie unterschiedlich sind, denselben Typ haben (Typ Abhängigkeit im Allgemeinen) und ähnliche Eigenschaften (gleichnamige Attribute) besitzen. Diese Tatsache sowie die Beziehungen zwischen den unterschiedlichen Abhängigkeiten muss auch darstellbar sein. Dafür wurde der Composite Pattern benutzt, genau um diese ,,Teil-Ganze“-Hierarchien von Abhängigkeiten darzustellen.

Abbildung 4.12 stellt das Abhängigkeitsmodell dar. Das Objekt *Component* des Composite-Patterns ist in diesem Modell die abstrakte Klasse ***Dependency*** und deklariert die Schnittstelle für alle Objekte in der **CompositeDependency**-Klasse (das zusammengesetze Element des Composite-Patterns). Die Blätter sind genau die oben genannten Basistypen von Abhängigkeiten: **Inter-Service-Dep**, **Serv-Resource-Dep** und **Inter-Resource-Dep**, deren Komposition die komplexen Typen von Abhängigkeiten ergeben. Ein *CompositeDependency*-Objekt kann mehrere Blattobjekte beinhalten und/oder ein oder mehrere andere *CompositeDependency*-Objekte. Eine ausführliche Beschreibung jeder dieser Komponente als auch der Attribute und Methoden dieser Klassen wird in den folgenden Abschnitten gegeben.

4.2.3.1 Attribute

Die Attribute, die die Klasse *Dependency* besitzt, sind als Eigenschaften der Beziehung zwischen zwei Komponenten, die zur Realisierung des Dienstes beitragen, zu verstehen. Diese Attribute werden speziell den bestimmten Basisklassen entsprechend belegt. Die Attribute, die zur Beschreibung der Klasse **Dependency** genutzt werden, sind:

- ***antecedent*** ist ein Attribut vom Typ *ServiceBuildingBlock* d.h. *Resource* oder *Functionality*; das heißt, es kann entweder eine Ressource oder eine Dienstfunktionalität sein (siehe Einteilung in

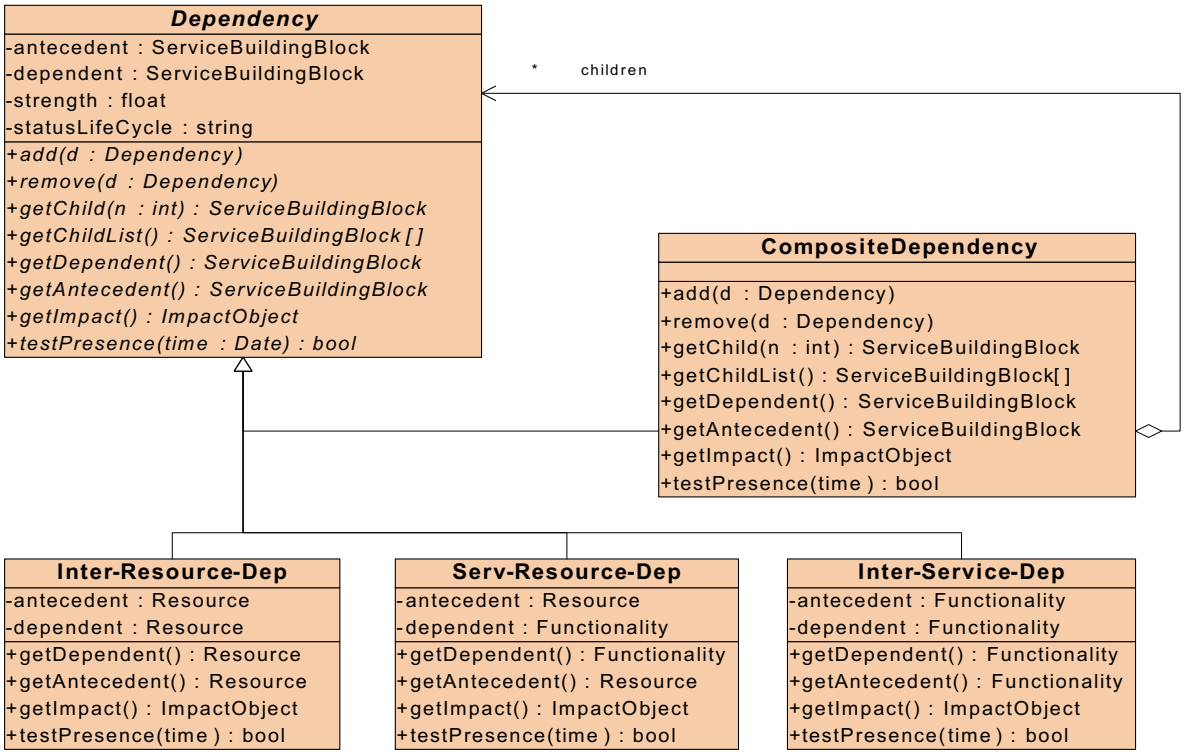

Abbildung 4.12: Das Abhängigkeitsmodell

der Abbildung 4.9). Ein Antecedent ist ein *ServiceBuildingBlock*-Objekt, von dem andere *ServiceBuldingBlock*-Objekte abhängig sind. Das *Antecedent*-Attribut ist wichtig, weil man daraus erkennen kann, von wo die Abhängigkeit ausgeht. Es ist ein wichtiger Teil der Abhängigkeitshierarchie, da es die Komponenten auf einer niedrigeren Ebene mit denen auf einer höheren verbindet.

- ***dependent*** ist ein Attribut vom Typ *ServiceBuildingBlock* (siehe Abbildung 4.9), das heißt entweder *Resource* oder *Functionality*. Ein Dependent ist ein *ServiceBuildingBlock*-Objekt, das von anderen *ServiceBuildingBlock*-Objekten abhängig ist. Es ist eine Komponente höher in der Abhängigkeitshierarchie und von den Komponenten auf eine niedrigeren Ebene abhängig. Es ist ebenfalls eine wichtige Komponente bei der Analyse und Navigation in der Abhängigkeitshierarchie.

- ***strength*** repräsentiert die Wichtigkeit einer Abhängigkeit und ist vom Typ Float (im Bereich $[0..1]$). Somit ist es eines der wichtigsten Attribute für die Beschreibung von Redundanzen. Der Wert 1 bedeutet, dass der Antecedent absolut notwendig ist, weil ohne ihn die höheren Komponenten (Dependent) nicht funktionsfähig sein können. Wenn der Wert kleiner wird, kann man davon ausgehen, dass der Ausfall der entsprechende Komponente in einem gewissen Ausmaß die höheren Komponenten beeinflusst, ein totaler Ausfall Komponente kann jedoch nicht allgemein und generell verursacht werden. Implizit zeigt es, was für eine Auswirkung und welches Ausmaß der Ausfall des Antecedents hat. Die entsprechenden Werte dieses Attributs werden bei der Systemkonfiguration festgelegt. Es wird eine spezielle Heuristik dafür benutzt, um die Wichtigkeit der Abhängigkeit einschätzen zu können. Wenn eine Komponente einer Abhängig-

keit, mit einem *strength*-Attribut vom Wert „1“ z.B. wegen des Ausfalls der Antecedents nicht mehr funktionsfähig ist, ist es von größerer Wichtigkeit, als wenn die entsprechende Abhängigkeit den Wert „0,40“ hat.

- ***statusLifeCycle*** ist ein Attribut, das den Zustand einer Abhängigkeit bezüglich des Lebenszyklus eines Dienstes zeigt. Mögliche Werte dafür sind: *planned*, wenn es sich um eine Abhängigkeit handelt, die in Entwurfphase des Dienstes festgelegt wird. Diese existieren in der *design* bzw *negotiation* Phase des Service Life Cycle. Der Wert *active* ist eigentlich der wichtigste, denn er bezieht sich auf die Abhängigkeiten in den *provisioning*- und *usage*-Phasen, das heißt während der Dienst implementiert, konfiguriert und getestet bzw. benutzt wird. In der *usage*-Phase wird auch der Wert *inactive* zugewiesen, und zwar dann, wenn diese Abhängigkeit tatsächlich besteht aber die Antecedents nicht verwendet werden oder in der *change*-Unterphase, wenn die Komponente oder Implementierung durch neue ersetzt werden. Für die *deinstallation*-Phase wird das Attribut den Wert *withdrawn* annehmen, der zeigt, dass der Dienst nun nicht mehr benutzt wird, aber die Struktur dahinter noch besteht.

4.2.3.2 Kindbezogene Methoden

Die abstrakte Klasse *Dependency* hat 8 abstrakte Methoden, die in kinderbezogene Methoden und Methoden für das Fehlermanagement unterteilt sind. Diese Einteilung liegt im Aufbau des Composite Pattern selbst. Die kinderbezogenen Methoden sind diejenigen, die für die Verwaltung und Speicherung von Kinderelementen zuständig sind. Diese Methoden werden nur in der Klasse *CompositeDependency* implementiert, weil nur diese Art von Objekten die Eigenschaft, Kinderelemente zu besitzen, haben. Die Methoden für das Fehlermanagement werden für alle Typen von Kinderelementen repräsentiert. Die letzten werden im Abschnitt 4.2.3.4 ausführlich beschrieben. Die kinderbezogenen Methoden der Klasse *Dependency*, die dann in der Klasse *CompositeDependency* implementiert werden, sind:

- ***add(d:Dependency)*** bewirkt, dass eine zusätzliche Abhängigkeit **d** dem Container hinzugefügt wird. Man betrachtet ein *CompositeDependency*-Objekt als einen Behälter (Container), der mehrere einfache Komponenten beinhaltet. Wenn z.B. ein *CompositeDependency*-Objekt zwei Blattkomponenten hat, wird eine neue Abhängigkeit *d* durch Anwendung der Operation *add(d)* dem zusammengesetzten Objekt hinzugefügt (das *CompositeDependency*-Objekt hat nun drei Blattkomponenten).

- ***remove(d:Dependency)*** ist die Methode, die eine Abhängigkeit **d** aus dem Container entfernt. Diese Methode wird dann benutzt, wenn eine Abhängigkeit (Blattkomponente) aus einen *CompositeDependency*-Objekt entfernt werden muss. Ein *CompositeDependency*-Objekt hat z.B. drei Blattkomponenten; durch Anwendung der Methode *remove(d)* wird die Abhängigkeit *d* gelöscht und daraufhin hat das zusammengesetzte Objekt nur noch zwei Blattkomponenten.

- ***getChild(n:int)*** gibt die n-te Kindkomponente der Abhängigkeit zurück. Es wird eine Reihenfolge jeweils für die nächsttiefere Stufe von Kinderobjekte vorgegeben, entsprechend der Reihenfolge der Einfügeregeln.

- ***getChildList()*** gibt eine Liste aller Kindkomponenten einer Abhängigkeit zurück. Diese Methode bezieht sich auch nur auf die nächsttiefere Stufe von Kindkomponenten.

4.2.3.3 Klassen des Composite Patterns

Das Composite Pattern *Dependency* beinhaltet eine zusammengesetzte Klasse *CompositeDependency* und drei Arten von Blattklassen. In diesem Abschnitt werden all diese Klassen ausführlich beschrieben und es werden die Ergänzungen bzw. Erweiterungen zu der abstrakten Klasse *Dependency* aufgezeigt.

Die drei Blattkomponenten des Composite-Patterns in dem Abhängigkeitsmodell sind:

- ***Inter-Resource-Dep*** repräsentiert die Klasse der Abhängigkeiten zwischen Ressourcen. Die Attribute: *antecedent, dependent, strength, statusLifeCycle* sind von der *Dependency*-Klasse geerbt. Der Typ der Attribute *dependent* und *antecedent* ist für diesem Fall Ressource (Objekt der Klasse *Resource*). Die Methoden *getImpact(), getDependents(), getAntecedents()* und *testPresence(time)* implementieren die abstrakten Methoden der Klasse Dependency. Die Methoden *getDependents(), getAntecedents()* haben für diese Klasse die Rückgabewerte vom Typ *Resource*.
- ***Service-Resource-Dep*** ist die Klasse der Abhängigkeiten zwischen den Ressourcen (Objekte vom Typ *Resource*) und Funktionalitäten im Allgemeinen (Objekte der Klasse *Functionality*). Sie erbt von der Klasse Dependency die Attribute *antecedent, dependent, strength, statusLifeCycle* und implementiert die Methoden *getImpact(), getDependents(), getAntecedents()* und *testPresence(time)*. Es handelt sich hier um die Klasse aller Abhängigkeiten zwischen Ressourcen und Diensten, Dienstfunktionalitäten, Subdiensten oder Teilfunktionalitäten. Das Attribut *dependent* ist vom Typ *Resource* und *antecedent* ist von Typ *Functionality*, wie am Anfang des Kapitels beschrieben, heißt das Dienst oder Dienstfunktionalität).
- ***Inter-Service-Dep*** stellt die Klasse der Abhängigkeiten zwischen *Functionality*-Objekten dar. Das heißt, es umfasst alle Abhängigkeiten zwischen Diensten und/oder Subdiensten, Funktionalitäten und/oder Teilfunktionalitäten. Die Klasse besitzt dieselben Attribute wie die abstrakte Klasse *Dependency*. Die Attribute *dependent* und *antecedent* werden überschrieben und haben den Typ *Functionality*. Die Methoden *getDependents(), getAntecedents()* haben für diese Klasse die Rückgabewerte von Typ *Functionality*. Die Methoden *getImpact()* und *testPresence(time)* behalten die Rückgabewerte, die in der *Dependency*-Klasse angegeben sind.

Die Klasse ***CompositeDependency*** ist die zusammengesetzte Klasse dieses Patterns. Es beschreibt das Verhalten eines Objektes mit Kindern, speichert diese Kindkomponenten, Blatt- und zusammengesetzte Komponenten und implementiert die Methoden, die sich auf die Kindkomponenten beziehen (*add, remove, getchild, getChildList*) aus der *Dependency* Schnittstelle. Die restlichen Methoden werden rekursiv auf alle Kinder angewendet. Die Methode *getImpact()* muss z.B. so implementiert werden, dass die Auswirkung der Antecedents auf die Abhängigkeit als auch auf die Dependents berechnet werden.

4.2.3.4 Methoden für Fehlermanagenement

Zusätzlich zu den in 4.2.3.2 beschriebenen Methoden existieren noch die Methoden, die hauptsächlich für das Fehlermanagement benutzt werden. Die kinderbezogenen Methoden sind für die Verwaltung und Speicherung von Kinderelementen zuständig und deshalb in der Klasse *CompositeDependency* implementiert. Auf der anderen Seite gibt es die übrigen Methoden zur Bestimmung der Ursachen von Fehlern in einem System, wenn eine Funktionalität nicht mehr in Anspruch genommen werden kann,

oder den Auswirkungen, die ein Ausfall einer tieferen Komponente auf verschiedene Funktionalitäten bzw. Nutzer hat. Diese unterscheiden sich von den ersteren, weil sie in beiden Arten von Komponenten implementiert werden. In den Blattkomponenten wird in Betracht gezogen, dass sie Basiskomponenten sind. In der zusammengesetzten Komponente müssen die Methoden so implementiert werden, dass sie rekursiv alle Rückgaben der Methoden auf die niedrigeren Ebenen miteinbeziehen.

Die Methoden für das Fehlermanagement sind:

- ***getAntecedent()*** gibt alle *ServiceBuildingBlock*-Objekte zurück, von denen die Dependents der aktuellen Abhängigkeit abhängig sind. Sie besitzt auch unterschiedliche Rückgabewerte, angepasst an die Klasse, in der sie implementiert wird. Auf der Blattebene gibt die Methode den Antecedent, d.h. diejenige Komponente, von der der Dependent abhängig ist, zurück. In der *CompositeDependency*-Klasse wird sie rekursiv implementiert und gibt, von einer bestimmten Komponente ausgehend, die Antecedents aller Kindkomponenten, die davon abhängig sind, zurück.

  Diese verwendete Rekursion trägt dazu bei, dass die Methode *getAntecedent()* in einem zusammengesetzten Objekt für die **top down**-Navigation, d.h zur Analyse, wie man im Falle einer Störung die Ursache schnell und effizient finden kann (*engl.* ***root cause analysis***) benutzt wird.

- ***testPresence(time)*** berechnet und gibt den Status der Abhängigkeit zu einem gewissen Zeitpunkt (der Zeitpunkt wird als Parameter benutzt) zurück. Diese Abhängigkeit ist auf jeden Fall modelliert, aber aufgrund der Dienstlogik wird sie gerade benutzt (Rückgabewert *true*) oder nicht (Rückgabewert *false*).

  Diese Methode spielt eine sehr große Rolle für die dynamischen Abhängigkeiten. Wenn ein bestimmtes Ereignis auftritt, dann kann z.B nur eine der möglichen Abhängigkeiten ,,aktiv“ sein und alle anderen ,,inaktiv“. Das heißt, je nachdem welche der einfachen Abhängigkeiten aus dem ,,BÜndel“ von dynamischen Abhängigkeiten gerade ausgewählt wird, wird nur für diese Abhängigkeit der Wert *true* zurückgegeben und für anderen Abhängigkeiten der Wert *false*.

- ***getDependent()*** gibt alle *ServiceBuildingBlock*-Objekte, die von den Antecedents der aktuellen Abhängigkeit abhängig sind, zurück. Je nachdem, in welcher Klasse sie implementiert wird, werden unterschiedliche Rückgabewerte betrachtet. Sie gibt aber immer den Dependent zurück. In der Blattklasse muss sie den entsprechenden Dependent, d.h die Komponente, die vom Antecedent abhängig ist, zurückgegeben. In der *CompositeDependency*-Klasse wird sie rekursiv implementiert. Sie gibt die Dependents (*ServiceBuildingBlock*-Objekte) aller Kindkomponenten zurück.

  Wegen dieser Eigenschaft der Methode ist sie hauptsächlich für die **bottom up**-Navigation gedacht, d.h zur bestimmung, welche Auswirkungen und in welchem Ausmaß hat der Ausfall einer Komponente auf die von ihr abhängigen Komponenten (*engl.* ***impact analysis***).

- ***getImpact()*** berechnet die Auswirkung des Ausfalls eines ihrer Antecedents auf die Dependents, d.h. auf alles, was in der Hierarchie untergeordnet und mit der aktuellen Abhängigkeit verknüpft ist). Der Rückgabewert ist vom Typ ImpactObject. Der ImpactObject Typ wird in dieser Arbeit nicht definiert, es wird nur angenommen, dass es sich um einen komplexen Typ handelt, der das genaue Ausmass an Auswirkung definieren kann.

  Diese Methode wird unterschiedlich für Blatt- und für zusammengesetzte Komponenten im-

plementiert. In einer Blattklasse berechnet die Methode die Auswirkung, die der Ausfall von *antecedent* auf den *dependent* verursacht. Für die Klasse *CompositeDependency* wird sie auf eine andere Weise implementiert, und zwar muss sie rekursiv die Methoden *getImpact()* der Kindkomponenten aufrufen und daraus die Auswirkung des Ausfalles aufgrund dieser komplexen Abhängigkeit berechnen.

Diese Methoden sind, wie der Titel auch impliziert für das Fehlemanagement entworfen. Wie schon bei der Beschreibung dieser Methoden und in der Einleitung dieser Arbeit (siehe Kapitel 1) erwähnt, gibt es zwei Ansätze, denen gefolgt wird, wenn es um das Fehlermanagement geht: der Top Down-(root cause analysis) und der Bottom Up-(impact analysis) Ansatz.

Eine kurze Darstellung der beiden Ansätze gegenüber dem hier entwickelten Modell wird eine der Anwendungen dieses Modells zeigen.

Top-Down-Analyse Wie schon im Kapitel 1 Abbildung 1.1 gezeigt, besteht eine der größten Problematiken des Fehlemanagements darin, wie schnell und ohne große Verzögerungen die Ursache eines Fehlers herausgefunden und aufgehoben wird.

Wie wird das anhand des obenbeschriebenen Modelles realisiert? Angenommen, Nutzer X meldet den Ausfall der Dienstfunktionalität D1, aber völlig normale Funktion der anderen Dienstfunktionalitäten. Das erste, was man tut, ist die Auflistung aller Abhängigkeiten, die irgendwie mit dieser Funktionalität in Verbindung stehen und nicht mit den funktionierenden Funktionalitäten.

Durch Anwendung der Operation *getAntecedent()* werden die möglichen Kandidaten, von denen die Funktionalität D1 abhängig ist, zurückgegeben. Der Test über die tatsächliche Verwendung dieser Abhängigkeiten wird mit *testPresence(time)* durchgeführt. Wenn ein „true“ Wert zurückgegeben wird, dann heißt, es dass diese Abhängigkeit zum Zeitpunkt *time* im Ansatz (benutzt) ist bzw. gewünscht ist. Wenn aber ein *false* zurückgegeben wird, kann man daraus schließen, dass die Abhängigkeit zum Zeitpunkt *time* nicht aktuell war, was gleichbedeutend mit „nicht wichtig“ für den normalen Ablauf der Dienstfunktionalität D1 ist.

Wenn der Wert *true* zurückgegeben wird, wird der Test auf dem Antecedent durchgeführt. Das wird mit der Methode *getStatus(time)* (siehe Seite 61) der Klasse *ServiceBuildingBlock* angewendet. Wenn der Rückgabewert *false* ist, dann ist die Komponente, die zur Realisierung des Dienstes beiträgt, zum getesteten Zeitpunkt nicht voll funktionsfähig gewesen. Dass heißt, die Ursache des Ausfalls ist gefunden worden. Wenn aber der Rückgabewert der Methode *getStatus(time) true* ist, dann muß die obere Prozedur rekursiv angewendet werden, immer eine Ebene tiefer, bis die Ursache der Störung gefunden wird.

Bottom Up-Analyse In der Abbildung 1.2 wurde eine kurze Darstellung dieses Ansatzes realisiert. Die zugrundeliegende Problematik ist, welche Auswirkungen der Ausfall einer Komponente (z.B. einen Switch) verursacht.

Angenommen, eine Komponente (Resource R1) fällt aus. Das heißt, *R1.getStatus(time)* liefert den Wert *false* zurück. Durch die Navigation in der Abhängigkeitshierarchie muss herausgefunden werden, welche Auswirkungen der Ausfall auf das Gesamtsystem hat. Wie wird das mit dem Modell realisiert? Durch der Ausfall der Ressource R1 werden höchstwahrscheinlich eine oder mehrere Dependents beeinträchtigt. Daraufhin wird die Methode *testPresence()* auf die Abhängigkeiten zu diesen möglichen Kandidaten angewendet. Die mit Rückgabewerte *true* sind für die weitere Analyse wichtig, da die Abhängigkeiten zum Zeitpunkt *time* tatsächlich

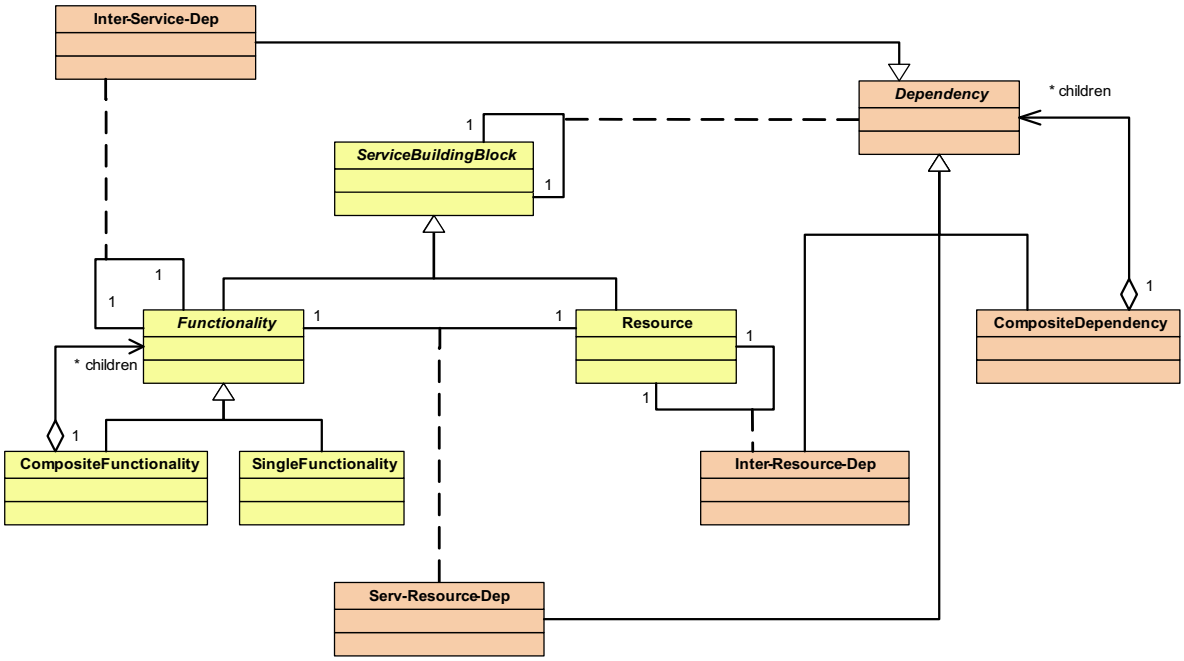

Abbildung 4.13: Klassendiagramm des Abhängigkeitsmodells

verwendet worden sind bzw. müssen. Die mit Rückgabewerte *false* sind unwichtig, da die Abhängigkeiten zum genannten Zeitpunkt keine Verwendung hatten. Es geht hier aber nicht um Zeiträume, wobei diese sehr wichtig sind. Wenn z. B. eine Komponente zum Zeitpunkt **t** ausfällt aber der Dienst **X** braucht sie erst um Zeitpunkt **t+4** dann muß man *testPresence(t+4)* und nicht *testPresence(t)*. Dieses Problem der Zeiträume wurde hier nicht betrachtet.

Die Methode *getDependent()* wird dann auf allen Abhängigkeiten die auf die Anwendung von *testPresence(time)* den Rückgabewert *true* hatten, angewendet. Sie liefert dann alle direkten Dependents, die vom Ausfall beeinträchtigt werden. Wenn der Dependent das obere Element in der Abhängigkeitshierarchie ist, dann wird er zurückgegeben. Sonst wird die oben beschriebene Prozedur nach oben wieder und wieder angewendet bis alle Komponenten, die vom Ausfall beeinträchtigt sind, gefunden sind.

Das Ausmaß der Auswirkungen kann man auch quantifizieren, indem man die Methode *getImpact()* rekursiv nach oben anwendet. Der Rückgabewert dieser Funktion ist vom Typ ImpactObject und muss genau beschreiben können, wie sich der Ausfall einer Komponente auf die von ihr abhängigen Komponenten auswirkt. Es muss ebenfalls mit Bezug zu *strength* die Wichtigkeit der Abhängigkeit berechnet werden. Denn es ist nicht dasselbe, ob eine Abhängigkeit mit `strength=0.4` ausfällt oder eine mit `strength=1.0`.

Eine Übersicht über alle in diesem Abschnitt dargestellten Klassen und deren Beziehungen zueinander kann man sich in der Abbildung 4.13 verschaffen. Man kann daraus die Komplexität des Modelles erkennen, allerdings auch die Tatsache, dass es auf einfachere Teile, wie in diesem Abschnitt dargestellt, zurückführbar ist.

4.3 Zusammenfassung und Bewertung

In diesem Kapitel wurden auf der einen Seite die Methodik zur Verfeinerung des MNM-Dienstmodells, im Hinblick auf die Abhängigkeiten, beschrieben. Nach jedem Verfeinerungsschritt wurde analysiert, inwiefern die Abhängigkeiten modelliert werden können. Die Schlussfolgerung daraus ist, dass die MNM-Dienstmodell-Methodik eigentlich für das Bestimmen der Abhängigkeiten besser geeignet ist als für deren Modellierung (siehe Anfang von Abschnitt 4.2.3). Sehr viele Informationen, die in den Aktivitätsdiagrammen bzw. Kollaborationsdiagrammen auftreten, sind sehr hilfreich für die Modellierung der Abhängigkeiten, die dann im zweiten Teil des Kapitels folgte.

Anforderungen	Bewertung
Dienstbezogen	
Technologieunabhängigkeit	vollständig erfüllt
Allgemeinheit des Dienstbegriffes	vollständig erfüllt
Lebenszyklus	vollständig erfüllt
Dienstfunktionalitäten	vollständig erfüllt
Dienstparameter	teilweise erfüllt
Erweiterbarkeit	vollständig erfüllt
Anwendbarkeit	erfüllt
Dienstgüte	vollständig erfüllt
Abhängigkeitbezogen	
Abhängigkeiten zwischen Ressourcen	vollständig erfüllt
Abhängigkeiten zwischen Ressourcen und Dienste	vollständig erfüllt
Abhängigkeiten zwischen Dienste und Subdienste	vollständig erfüllt
Dienstfunktionalitätspezifischen Abhängigkeiten	vollständig erfüllt
Statische Abhängigkeiten	vollständig erfüllt
Dynamische Abhängigkeiten	vollständig erfüllt
Redundanzen	vollständig erfüllt

Tabelle 4.4: Erfüllung der Anforderungen durch das Abhängigkeitsmodell

Auf der anderen Seite wurde ein Abhängigkeitsmodell dargestellt, das auf mehreren Teilkomponenten basiert. Es wurden erst die Begriffe *ServiceBuildingBlock, Resource, Functionality, SingleFunctionality, CompositeFunctionality* beschrieben. Auf deren Basis baut dann das Abhängigkeitsmodell auf. Dieses Modell ist aus den Assoziationsklassen aller Kombinationen der obigen Klassen entstanden. Diese wurden zu einem Composite Pattern *Dependency*, mit Blattelementen als Basisabhängigkeiten und einem zusammengesetzten Element als Kombinationen der oberen modelliert. Für die Klassen des Modells wurden Attribute und Methoden definiert, die zur Speicherung und Verwaltung der Abhängigkeiten auf der einen Seite und für Fehlermanagement auf der anderen Seite benutzt werden können.

In der Tabelle 4.4 wird das Abhängigkeitsmodell bewertet. Die dienstbezogenen Anforderungen beziehen sich hauptsächlich auf das MNM-Dienstmodell, welches den Rahmen für die Entwicklung des Abhängigkeitsmodells festlegt. Trotz des Abhängigkeitsmodells kann man, im Bereich der dienstbezogenen Anforderungen, die Dienstparameter nicht vollständig in Betracht ziehen. Das beruht darauf, dass das MNM-Dienstmodell für jeden Parameter separat (z.B. für jeden Kunden separat) instantiiert werden muß. Der Parameter (z.B. Kunde) wird am Anfang festgelegt und nicht mehr bis zum

Ende der Modellierung geändert. Dass heißt, wenn mehreren Kunden denselben Dienst nutzen, kann man mit diesem Modell nicht eine einzige Darstellung realisieren, die dann durch Parameter gesteuert wird; sondern es muss für jeden Kunden separat ein Abhängigkeitsmodell instantiiert werden. Die Anwendung des Modells ist nicht trivial sondern setzt gewisse Kenntnisse über das MNM-Dienstmodell voraus.

Die abhängigkeitsbezogenen Anforderungen werden alle vom Abhängigkeitsmodell vollständig erfüllt. Die drei Arten von Abhängigkieten zwischen Ressourcen untereinander, zwischen Ressourcen und Diensten und zwischen Diensten untereinander werden durch den Klassen *Inter-Resource-Dep*, *Service-Resource-Dep* bzw. *Inter-Service-Dep* dargestellt. Die dienstfunktionalitätspezifischen Abhängigkeiten werden durch die Darstellung von Aktivitäts- bzw. Kollaborationsdiagrammen, für jede Funktionalität separat festgelegt und erst danach in dem Abhängigkeitsmodell zusammengefasst. Die statischen Abhängigkeiten werden erst durch das MNM-Dienstmodell bestimmt und durch das Abhängigkeitsmodell, mithilfe der Assoziationsklassen und deren Beziehungen, zueinander modelliert. Die dynamischen Abhängigkeiten werden in dem Abhängigkeitsmodell instantiiert und deren tatsächliche Verwendung wird durch die Methode *testPresence()* festgelegt. Redundanzen sind durch Kollaborationsdiagramme darstellbar, jedoch deren genaue Detaillierung wird durch die Methoden des Abhängigkeitsmodells realisiert.

Im nächsten Kapitel werden Anwendungen der hier entstandenen Methodik und des Abhängigkeitsmodells auf Beispielszenarien gezeigt.

Kapitel 5

Modellierung des Szenarios

Inhaltsverzeichnis

In diesem Kapitel wird die Anwendung des im Kapitel 4 entwickelten Modells auf den Web Hosting- bzw. E-Mail-Dienst beschrieben. Dabei wird die dort beschriebenen Methodik angewendet. Im ersten Teil des Kapitels wird der vorgestellten Ansatz auf das Web Hosting-Dienst-Szenario angewendet. Es werden dabei sehr ausführlich die Aspekte der Dienstsicht als auch der Realisierungssicht betrachtet. In dem zweiten Teil wird eine kurze Anwendung der Methodik auf das E-Mail-Dienst-Szenario beschrieben.

5.1 Modellierung des Web Hosting Dienstes

Der Web Hosting-Dienst ist einer der wichtigsten Dienste, die das LRZ den am Münchner Wissenschaftsnetz angebundenen Instituten anbietet. Dabei muss eine sehr schnelle Reaktion, auf die möglichen Probleme, die auftreten können, gewährleistet werden. Die Modellierung der Abhängigkeiten

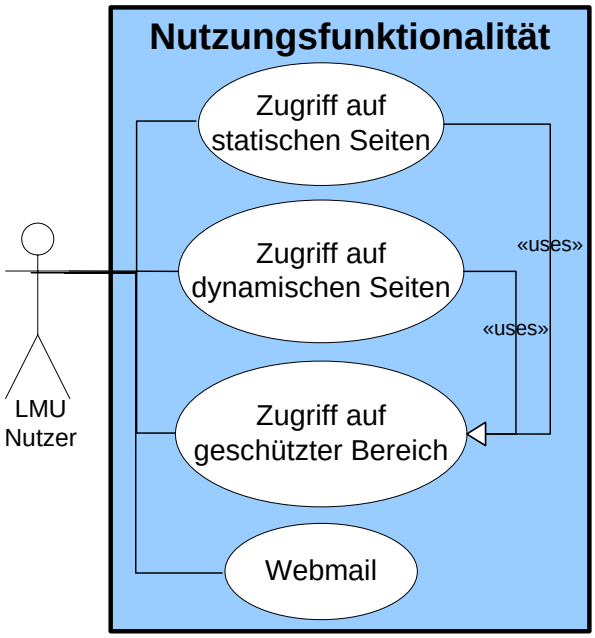

Abbildung 5.1: Anwendungsfalldiagramm für die Nutzungsfunktionalität

für den Web Hosting-Dienst (als Anwendung der oben beschriebenen Methodik) bezweckt eine Beschleunigung der Fehlersuche im Falle einer Störung. Dieses Unterkapitel ist folgendermaßen aufgebaut. Im Abschnitt 5.1.1 wird eine sehr detaillierte Beschreibung des Service Views mit abschließenden Bemerkungen bzgl. der Abhängigkeiten gegeben. Im Abschnitt 5.1.2 werden die providerinternen Aspekte umfangreich beschrieben und schließlich in 5.1.3 wird eine Anwendung des im Abschnitt 4.2 erstellten Abhängigkeitsmodells hergeleitet.

5.1.1 Service View

Wie schon bei der Beschreibung des MNM-Dienstmodells erläutert, wird als erstes der Service View (der Dienst, so wie er vom Kunde genutzt wird) modelliert; mit Hilfe der Anwendungsfalldiagramme (Use Case Diagramms). Der erste Schritt ist, alle Funktionalitäten des Dienstes aufzulisten und sie dann in den beiden Funktionalitätsklassen einzuteilen: die Nutzungs- und die Managementfunktionalität. Danach wird jede dieser Funktionalitäten in einem Aktivitätsdiagramm dargestellt. Die Nutztungsfunktionalitäten aus Dienstsicht werden im Abschnitt 5.1.1.1 betrachtet und die Managementfunktionalitäten im Abschnitt 5.1.1.2.

5.1.1.1 Die Nutzungsfunktionalität des Web Hosting-Dienstes

In diesem Abschnitt wird am Anfang eine Auflistung der Nutztungsfunktionalitäten und deren grafischen Darstellung als Anwendungsfalldiagrammen vorgestellt. Denach wird in den folgenden vier Abschnitten jede Nutztungsfunktionalität als Aktivitätsdiagramm dargestellt.

Die Nutzungsfunktionalität beinhaltet die Funktionalität „Zugriff auf Seiten“ und „Webmail“. Es wird nach der Art der zugegriffenen Seite in **Zugriff auf statische Seiten**, **Zugriff auf dynamische Seiten**

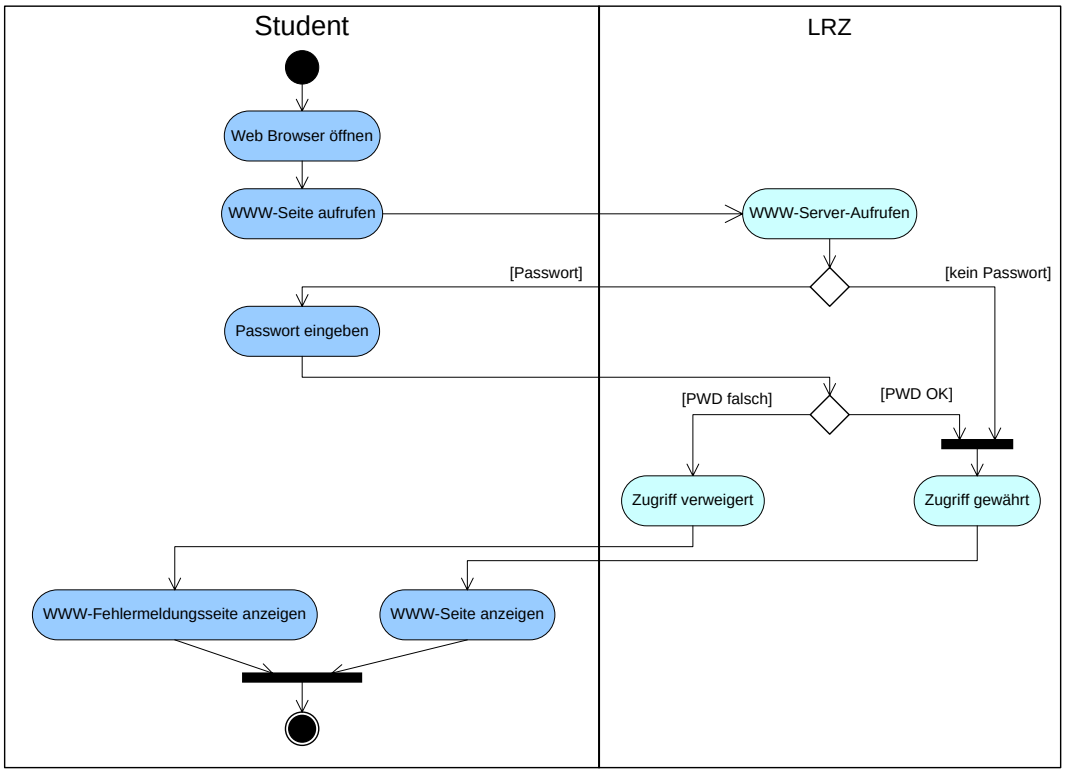

Abbildung 5.2: Aktivitätsdiagramm der Funktionalität, *Zugriff auf geschützten Bereich*

und **Zugriff auf geschützten Bereich** eingeteilt. Allerdings wird die dritte als Unterfunktionalität betrachtet, da sie bei der Realisierung den ersten beiden verwendet wird (<<uses>>-Beziehung) in dem Anwendungsfalldiagramm). Jede dieser Funktionalitäten wird detailliert beschrieben.

Wie in Abbildung 5.1 dargestellt, werden für jede der Funktionalitäten Use Cases ausgearbeitet. Für jede Funktionalität wird dann ein Aktivitätsdiagramm entworfen, das den zugrundeliegenden Ablauf darstellt. Die Funktionalität **Zugriff auf geschützten Bereich** wird als erstes beschrieben, weil sie dann in **Zugriff auf statische Seiten** bzw. **Zugriff auf dynamische Seiten** benutzt wird.

5.1.1.1.1 Zugriff auf geschützten Bereich hat vor allem mit Bereichen, die passwortgeschützt sind, oder die überhaupt nicht zugreifbar sind. Es gibt zum Beispiel sensible Daten, die nicht jedem zugänglich sein sollten. Oder es gibt Seiten, die jedem Nutzer zugänglich sind, aber nur innerhalb des Instituts aufrufbar sind.

Die Abbildung 5.2 zeigt das Aktivitätsdiagramm für den Zugriff auf einen geschützten Bereich. Die erste Aktivität besteht darin, den Web Browser zu öffnen. Die zweite Aktivität ist dann, die gewünschte Webseite in dem Webbrowser aufzurufen. In diesem Punkt sind dann drei Varianten möglich:

- die Webseite wird ohne Einschränkungen angezeigt (das wäre der Spezialfall Zugriff auf statische oder dynamische Seiten ohne Beschränkungen)
- die Webseite ist für keinen Nutzer zugänglich, das heißt, jeder, der diese Seite aufruft, wird eine Fehlermeldung lesen können

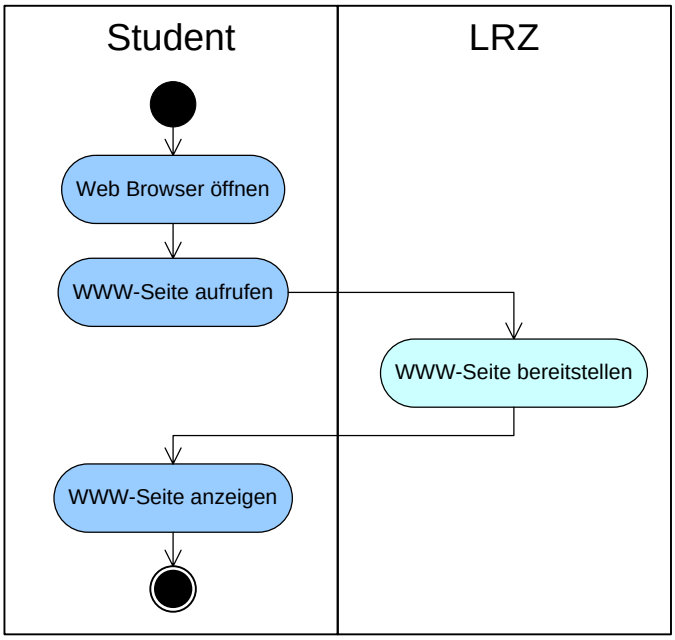

Abbildung 5.3: Aktivitätsdiagramm für den *Zugriff auf statische Seiten*

- der Zugriff auf die Webseite benötigt eine Autorisierung. Der Nutzer ist aufgefordert, Kennwort und Passwort einzugeben. Nach dessen Überprüfung wird für den Fall, dass das Passwort richtig war, die gewünschte Webseite angezeigt, ansonsten eine Fehlermeldung. Man kann die Zugangsdaten ggf. erneut eingeben.

Für diesen Fall sind folgende QoS-Parameter wichtig: die Zeit, die für die Autorisierung benötigt wird und die Verfügbarkeit der Seite.

Bemerkungen bzgl. Abhängigkeiten Abhängigkeiten sind aus der Abbildung 5.2 nicht ersichtlich. Daher muss man auf der Ebene des Realization Views ein umfangreiches Aktivitätsdiagramm erstellen. Man kann aber schon vermuten, dass ein Dateisystem und Autorisierungssystem erforderlich sind. Das heißt, es könnte einige Abhängigkeiten zwischen Webserver und dem Dateisystem geben.

5.1.1.1.2 Zugriff auf statische Seiten

Dies ist eine der einfachsten Funktionalitäten, nur eine Webseite öffnen und ihren Inhalt lesen.

Die Abbildung 5.3 zeigt den Zugriff auf statische Seiten als Aktivitätsdiagramm. Die erste Aktivität besteht darin, einen Web Browser zu öffnen. Nachdem der Web Browser offen ist, gibt man die gewünschte Seite ein, das heißt, die Seite wird mit Hilfe des Webservers aufgerufen und dann, als letzte Aktivität, wird sie angezeigt.

Wichtige QoS-Parameter für den Zugriff auf statische Seiten sind: die Zeit, die für das Herunterladen der Seite benötigt wird und die Verfügbarkeit der Seite; das heißt, ob die Seite überhaupt erreichbar ist oder nicht. Im Falle der Nichterreichbarkeit wird eine Webseite mit einer entsprechenden Fehlermeldung geladen (diese ist im Not-Server gespeichert).

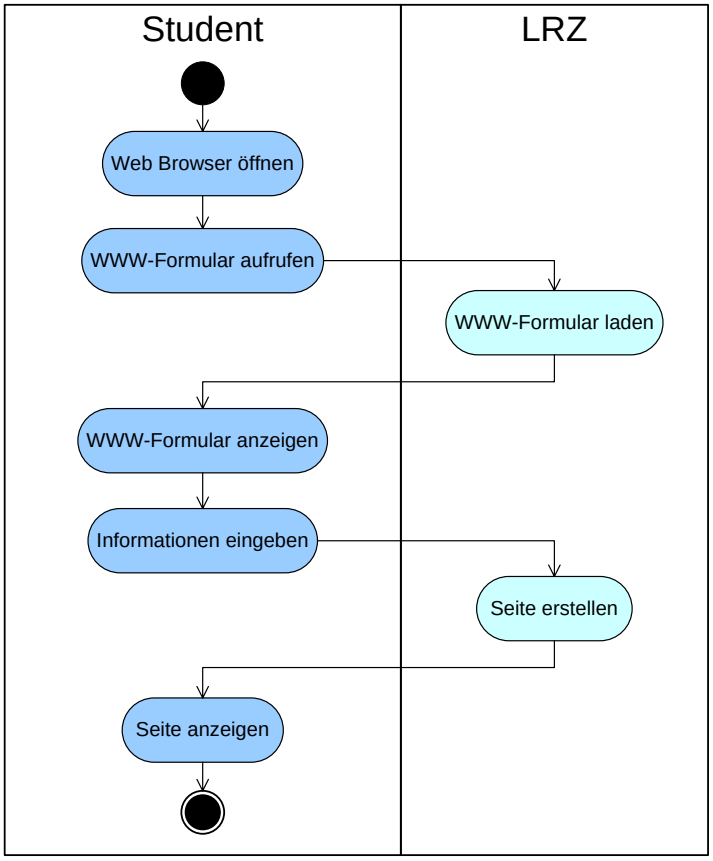

Abbildung 5.4: Aktivitätsdiagramm der Funktionalität *Zugriff auf dynamische Seiten*

Bemerkungen bzgl. Abhängigkeiten Aus dem oben beschriebenen Aktivitätsdiagramm allein können keine Arten von Abhängigkeiten hergeleitet oder modelliert werden. Daher ist eine Verfeinerung dieser Funktionalität notwendig, und zwar durch Darstellung des Aktivitätsdiagramms für ihre Realisierung wie in 4.1.1.2 beschrieben.

5.1.1.1.3 Zugriff auf dynamische Seiten

Es gibt mehrere Varianten, Websites zu realisieren und Inhalte anzubieten. In vielen Fällen reichen statische HTML-Seiten vollkommen aus. Es gibt aber auch viele Fälle, in denen der Inhalt einer Webseite davon abhängt, was ein Nutzer in ein davor aufgerufenes Formular eingegeben hat. In solchen Fällen muss anstelle einer statischen Webseite ein Programm (Skript) aufgerufen werden, das die passende Antwort erzeugt oder Aktion ausführt. Das heißt, dass die dynamischen Seiten direkt auf den Wunsch der Nutzer zugeschnitten werden und können eventuell aktuelle Informationen in Bezug auf den Nutzer miteinbeziehen.

Die Abbildung 5.4 repräsentiert das Aktivitätsdiagramm für den Zugriff auf dynamische Seiten. Die erste Aktivität ist, wie bei statischen Seiten auch, den Web Browser zu öffnen. Wenn der Web Browser offen ist, wird die statische Webseite, die das Formular beinhaltet, aufgerufen. Das Formular wird jetzt im Browser angezeigt und der Nutzer muss nur noch seine Daten eingeben. Nach der Beendigung dieser Eingabe werden die Daten bearbeitet und die von dem Nutzer gewünschte Webseite bzw. Informationen werden angezeigt.

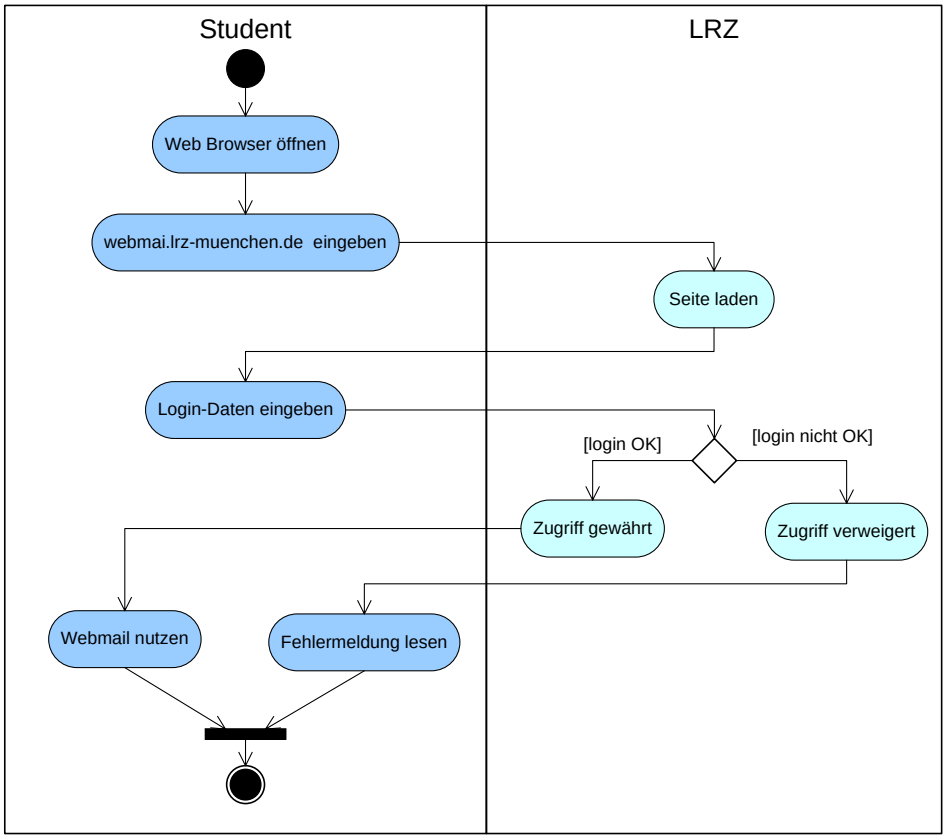

Abbildung 5.5: Aktivitätsdiagramm der Funktionalität *Webmail*

Für diesen Fall sind folgende QoS-Parameter in Betracht zu ziehen:

- die Zeit, die für die „Berechnung“ einer dynamische Seite benötigt wird (von der Eingabe der Daten bis zur Ausgabe der gewünschte Seite).
- die Verfügbarkeit der Seite; das heißt, ob die Seite überhaupt erreichbar ist oder nicht. Es ist ein bisschen anders als bei statischen Seiten, weil eine Seite z.B erreichbar sein kann, aber nach Versenden der Informationen kann die dynamische Seite nicht mehr erstellt werden (wegen eines Fehlers beim Ausführung des Skriptes, oder weil der NFS nicht erreichbar ist usw.).

Bemerkungen bzgl. Abhängigkeiten Man kann auch in diesem Fall keine Abhängigkeiten erkennen, also auch nicht modellieren. Die Erweiterung des Aktivitätsdiagramms im Abschnitt Realisation View wird aber dann mehr Informationen über die Abhängigkeiten, die für diese Funktionalität wichtig sind, aufzeigen.

5.1.1.1.4 Webmail Funktionalität

Diese Funktionalität ermöglicht dem Nutzer den Zugriff auf seine Mailbox über einen Web-Browser, also ohne dass ein spezielles Mailprogramm benutzt werden muss. Diese Funktionalität müsste eigentlich Teil des E-Mail Dienstes sein, aber beim LRZ ist sie organisatorisch dem Web Hosting-Dienst untergeordnet. Die erste Aktion des Nutzers ist Öffnen des Browser gefolgt von der Eingabe der Adresse der Webmailseite (`webmail.lrz-muenchen.de`).

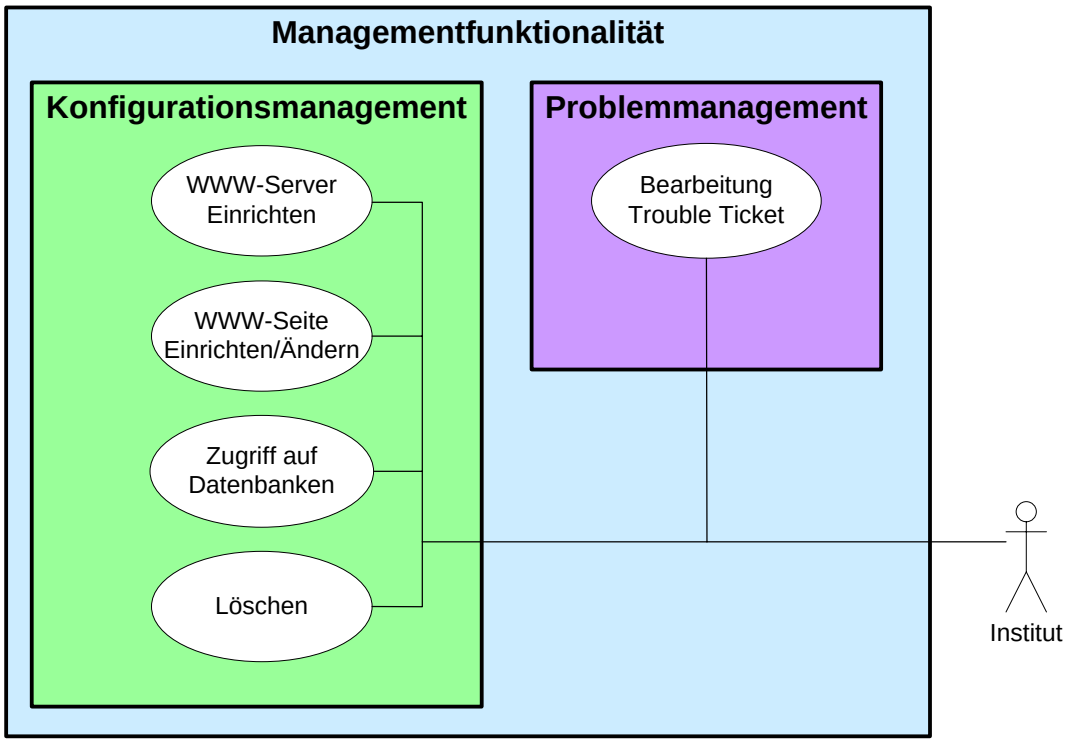

Abbildung 5.6: Anwendungsfalldiagramm für die Managementfunktionalität

Auf der Seite des Anbieters (LRZ) wird die Seite geladen und die Login Seite wird angezeigt. Nachdem der Nutzer das Kennwort und das Passwort eingegeben hat, werden seitens des LRZ die Logindaten überprüft und danach wird der Zugriff auf den Webmail Dienst gewährt (wenn die Überprüfung der Login Daten positiv war) oder nicht gewährt (wenn die Logindaten falsch sind).

Bemerkungen bzgl. Abhängigkeiten Nach dem hier dargestellten Diagramm kann man nur erahnen welche Abhängigkeiten in Betracht gezogen werden können. Eine genaue Bestimmung und Beschreibung dessen ist jedoch nicht möglich. Ein ausführliches Aktivitätsdiagramm, das die Bestimmung der Abhängigkeiten, die zur Realisierung der Webmail Funktionalität beitragen,zeigt, wird im Abschnitt 5.1.2 präsentiert.

5.1.1.2 Die Managementfunktionalität des Web Hosting-Dienstes

In diesem Abschnitt werden erst alle Managementfunktionalitäten aufgelistet und in einem Anwendungsfalldiagramm dargestellt. Danach werden jede der, in dem Diagramm dargestellten, Funktionalitäten mit Aktivitätsdiagrammen repräsentiert im Bezug auf die Dienstsicht.

Es gibt mehrere Managementfunktionalitäten, die dem Web Hosting-Dienst zuzuschreiben sind, aber es werden nur diejenigen Funktionalitäten betrachtet, die das Konfigurations- und Problemmanagement miteinbeziehen.

Die wichtigsten Aktionen, die vom Konfigurationsmanagement durchgeführt werden, sind das **Einrichten eines virtuellen Webservers**, **Einrichten oder Ändern von einzelnen Seiten**, **Löschen von**

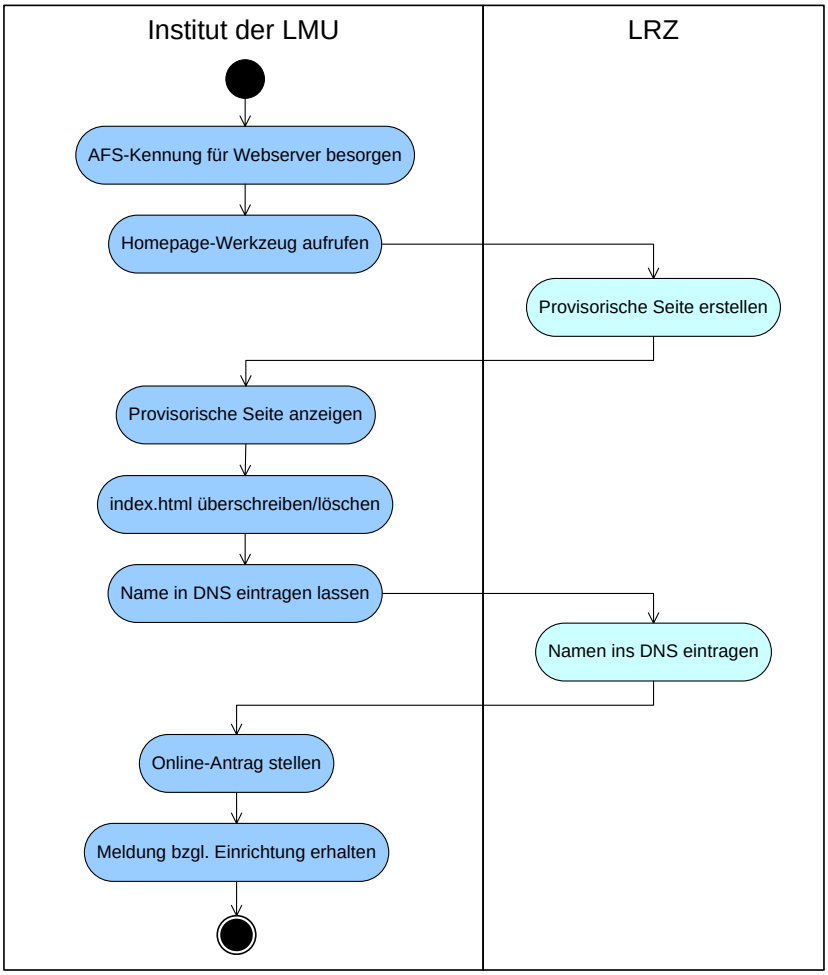

Abbildung 5.7: Aktivitätsdiagramm der Funktionalität *Einrichten eines virtuellen WWW-Servers*

Webseiten oder Webpräsenz und **Zugriff auf die Datenbanken über das PhpMyAdmin-Tool** (für den Zugang zum mySQL-Server). Für das Problemmanagement ist die **Bearbeitung eines Trouble Tickets** die Hauptaktivität.

Grafisch werden die Managementfunktionalitäten in Use-Case-Diagrammen dargestellt (siehe Abbildung 5.6).

5.1.1.2.1 Einrichten eines virtuellen WWW-Servers Das Aktivitätsdiagramm für diese Funktionalität ist in Abbildung 5.7 dargestellt. Die erste Aktion ist, eine AFS-Kennung für Webserver zu besorgen, da jeder neue Webserver seine eigene Kennung (nicht die Kennung des Master Users) benötigt.

Ein Master User ist ein Verantwortlicher, der dem LRZ gegenüber für ein bestimmtes Projekt autorisiert ist. Ansprechpartner für die Master User auf Seite des LRZ sind, je nach Hochschule bzw. Fachrichtung verschieden, die so genannten Betreuer. Der Master User übernimmt die Verantwortung für den ordnungsgemäßen Gebrauch der zugeteilten Benutzerkennungen. Er kann die Benutzung der LRZ-Systeme durch Benutzer seines Bereichs kontrollieren, einschränken und im Missbrauchsfall

unterbinden. Zu diesem Zweck stehen ihm gewisse Dienste zur Verfügung, die nachfolgend näher beschrieben sind.

Wenn die AFS-Kennung vorhanden ist, dann wird das Homepagecreate Tool aufgerufen, mit dem unter AFS eine Homepage eingerichtet werden kann. Das Dateiverzeichnis `~AFS-Kennung-Webserver/webserver/` wird für den Webserver angelegt und die entsprechenden Zugriffsrechte zugewiesen. Das Unterverzeichnis `webdata/` wird als Einstiegspunkt für die Webdateien (DocumentRoot des Webservers) angelegt. In der Datei `webdata/index.htm` findet man eine provisorische Startseite.

Als nächstes werden die Dateien erzeugt bzw. zum LRZ kopiert, wobei die Datei `webserver/webdata/index.htm` überschrieben wird. Ein sinnvoller Titel für die neukreierte Webpräsenz (er erscheint später in der Liste der Virtuellen WWW-Server am LRZ) ist anzugeben. Statische HTML-Dateien kann man schon in diesem Zustand über den Webserver des LRZ unter der URL `http://www.lrz-muenchen.de/~AFS-Kennung-Webserver/webserver/webdata/` anschauen.

Die nächste Aktion befasst sich mit dem Eintragen des Namens ins DNS; es wird ein Name ausgewählt, der von der jeweiligen Universität genehmigt werden muss. Dieser Name oder Namen müssen von der zuständigen Person am LRZ ins DNS eingetragen werden. Einen Online-Antrag auf einen virtuellen Webserver zu stellen, ist die nächste Aktivität, die stattfindet. Eine E-Mail an die Adresse `AFS-Kennung-Webserver@mail.lrz-muenchen.de`, die die Bestätigung der Anmeldung beinhaltet, wird in maximal 24 Stunden erhalten.

Bemerkungen bzgl. Abhängigkeiten Man kann hier nur erkennen, dass das AFS-Dateisystem miteinbezogen wird, das „irgendwie“ von einem BMF-Tool (Basis Management Funktionalität) abhängig ist. Der DNS spielt auch eine Rolle, aber welche, ist noch nicht klar. Dafür muss man dieses Aktivitätsdiagramm, wie in Abschnitt 4.1.1.2 beschrieben, erweitern.

5.1.1.2.2 Zugriff auf die Datenbanken über das PhpMyAdmin-Tool

Dynamische Webseiten können sowohl Datenbanken am LRZ als auch externe (z.B. eines Instituts) benutzen. Am LRZ stehen MySQL und Oracle zur Verfügung. MySQL kann von PHP-Skripten und Oracle kann von PHP- oder Perl-Skripten genutzt werden. Die eigene MySQL-Datenbank am LRZ kann mit dem phpMyAdmin-Server des LRZ verwaltet werden.

Das Aktivitätsdiagramm für diese Funktionalität wird hier, aus Pltzt Gründe, nicht mehr dargestellt, sondern wird direkt der Realization View präsentiert.

5.1.1.2.3 Einrichten/Ändern von Webseiten

Diese Funktionalität erlaubt das Einrichten bzw. Ändern von Webseiten von den verschiedenen Kunden (Institute oder für Mitarbeiter). Das Einrichten und das Ändern werden nicht separat betrachtet, weil bei der Einrichtung des virtuellen WWW-Servers automatisch eine erste statische Seite erstellt wird. Das heißt, diese ursprüngliche Seite wird geändert und neue kommen noch dazu.

Das Einrichten/Ändern von Webseiten ist grafisch in der Abbildung 5.8 als Aktivitätsdiagramm dargestellt. Die erste Aktion ist, auf dem ersten Ast des Aktivitätsdiagramms, die Datei index.html zu öffnen (wenn es sich um das Ändern der zuerst erstellten statischen HTML-Seite handelt) oder, auf

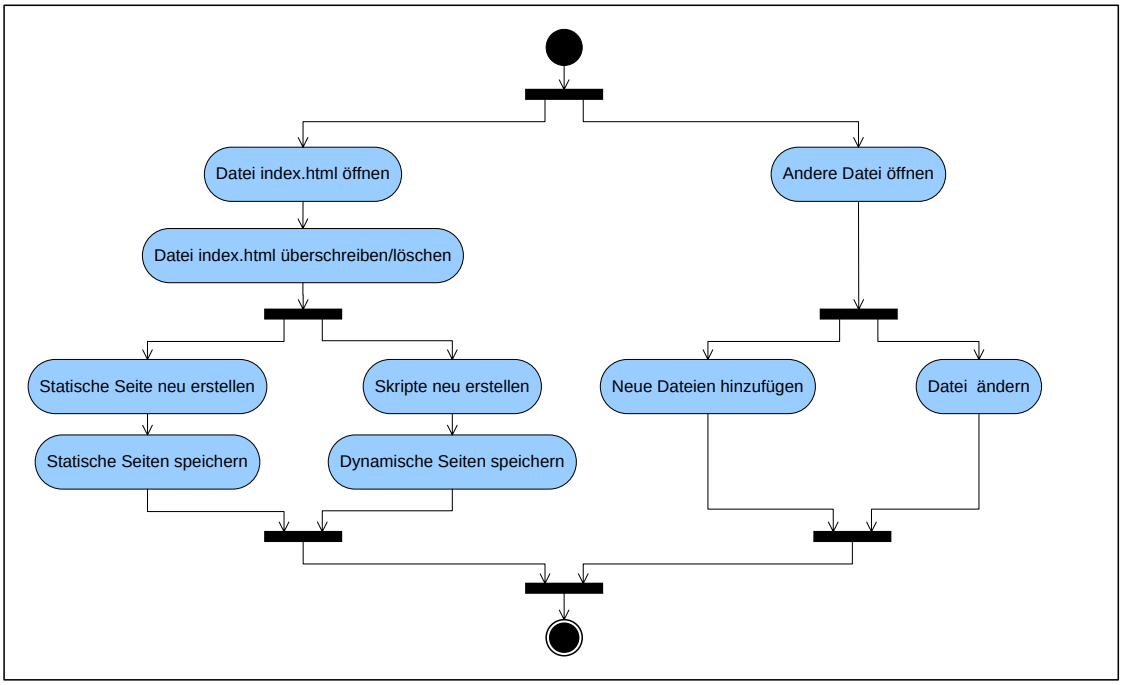

Abbildung 5.8: Aktivitätsdiagramm der Funktionalität *Einrichten/Ändern von Webseiten*

dem anderen Ast, eine andere Datei zu öffnen (eine Datei, in der eine Webseite gespeichert ist). Auf dem linken Ast folgt dann die Überschreibung bzw. das Löschen der Datei index.html. Es folgt dann die Erstellung von statischen Seiten gefolgt von deren Speicherung, oder von Skripten und dynamischen Webseiten, die anschließend ebenfalls gespeichert werden.

Auf dem rechten Ast, nach der Öffnung der Datei, folgen zwei andere mögliche Aktivitäten: die bestehenden Dateien ändern oder neue Dateien zu den schon bestehenden Webseiten hinzufügen.

Bemerkungen bzgl. Abhängigkeiten Offensichtlich muss es eine Abhängigkeit geben zwischen dem Webserver und mindestens einem Dateisystem. Wie diese aber aussehen und wie sie modelliert werden kann, kann man auf dieser Ebene des Service Views noch nicht sagen. Das wird auf der Ebene des Realization Views genauer beschrieben.

5.1.1.2.4 Löschen Diese Funktionalität ist aus zwei Teilen zusammengesetzt: das Löschen von Webseiten und das Löschen der Webpräsenz. Das Löschen von Webseiten ist nur ein Spezialfall von der Funktionalität Einrichten/Ändern von Webseiten. Demnach wird diese Funktionalität nicht mehr weiter betrachtet. Das Löschen der Webpräsenz wird realisiert, wenn der Besitzer des Webservers dem Webmaster meldet, dass die Webpräsenz gelöscht werden kann. Das ist auch empfohlen zu tun, denn sonst können bei nicht mehr aktive Webservern können Fehlermeldungen auftreten, die nicht eingeordnet werden können. Allerdings handelt es sich in diesem Fall mehr um eine organisatorische Sache, und daher sind auch die Abhängigkeiten, die hier auftreten könnten, fast nur organisatorischer Natur. Aus diesem Grund werden diese nicht weiter in dieser Arbeit betrachtet.

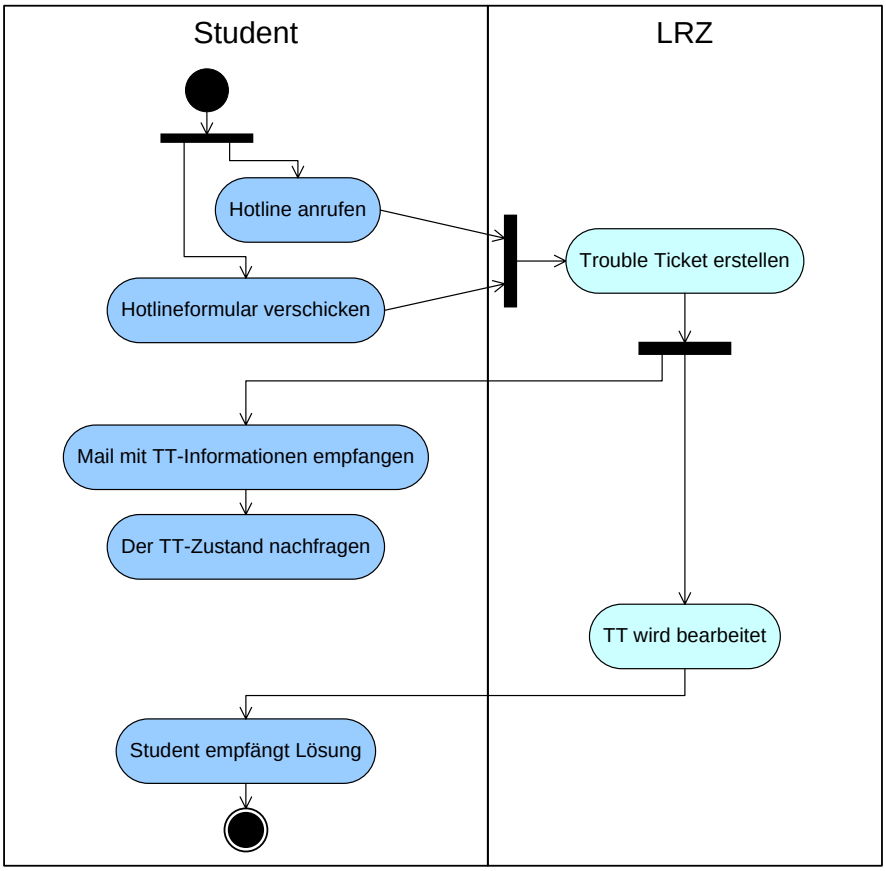

Abbildung 5.9: Aktivitätsdiagramm der Problemmanagementfunktionalität *Bearbeitung eines Trouble Tickets*

5.1.1.2.5 Bearbeitung eines Trouble Tickets Hier wird das **Action Request System-Tool** (**ARS**) für das Problemmanagement am LRZ beschrieben. Es wird für alle Arten von Problemen eingesetzt und nicht nur für Probleme, die mit dem WWW-Server zu tun haben.

Das Problemanagement wird am LRZ mit dem ARS-Tool (Action Request System) der Firma Remedy realisiert. Es wurde eine spezielle Konfiguration des Webservers realisiert (der Webdämon heißt **ars**). Über ARS werden alle empfangene Troble Tickets (TT) verwaltet. In der Abbildung 5.9 wird das Aktivitätsdiagramm dieser Problemmanagementfunktionalität dargestellt.

Der Vorgang wird dadurch gestartet, dass ein User des LRZs einen Anruf bei der Hotline macht oder das auf der LRZ Seite existierende Hotlineformular ausfüllt bezüglich eines Problems, was während der Benutzung eines Dienstes aufgetreten ist. Nach einer gewissen Zeit bekommt er eine Rückmeldung (i.a. per E-Mail) vom LRZ, die Informationen über das Trouble Ticket bezüglich seines Problems erhält (auch ein Trouble Ticket Nummer). Der User kann, anhand des TT-Nummers, über die ganze Zeit die Bearbeitung seines TT nachvollziehen. Das Problem wird umso schneller gelöst, desto größer seine Priorität ist.

Bemerkungen bzgl. Abhängigkeiten Abhängigkeiten können hier nicht entdeckt oder modelliert werden da es um die Kunden-Dienstleister Schnittstelle geht. Dieses Diagramm muss erweitert

werden, um heraus finden zu können, was für Beziehungen zwischen den unterschiedlichen Komponenten auf der Ebene der Realisierung vorhanden sind. Das einzige, was man hier erkennen kann, ist das es sich um die **ars**-Konfiguration des Web-Servers handelt.

5.1.2 Realization View

Im vorherigen Abschnitt wurden Aspekte der Interaktion Kunde-Dienst betrachtet, und die interne Realisierung des Dienstes wurde nicht betrachtet. In diesem Abschnitt werden die Providerinternen Prozesse betrachtet, mit Rücksicht auf das Aktivitätsdiagramm des Service Views. Die Einteilung in den zwei großen Interaktionsklassen wird weiter beibehalten. Daraufhin werden die Nutzung i Abschnitt 5.1.2.1 und das Management in Abschnitt 5.1.2.2 aus dem Sicht der Realisierung betrachtet.

5.1.2.1 Service Implementation und Service Logic

Es handelt sich hier um die interne Realisierung der Nutzung für den Web Hosting Dienst. Für diese Darstellung der providerinternen Aspekte die zur Realisierung der Nutzungsfunktionalität des Web Hosting-Dienstes beitragen, werden als Grundlage die im Abschnitt 5.1.1.1 realisierten Aktivitätsdiagramme betrachtet. Diese werden zu umfangreichen Aktivitätsdiagrammen erweitert, in denen die Providerseite auch ausführlich dargestellt ist. Zu jeder der Funktionalitäten werden dann zusätzlich, zur Bestimmung der Abhängigkeiten, auch Kollaborationsdiagramme erstellt.

Wie im Abschnitt 5.1.1 wird auch hier als erstes die Erweiterung der Funktionalität ,,Zugriff auf geschützten Bereich“ betrachtet weil diese dann von den anderen zwei benutzt wird.

5.1.2.1.1 Zugriff auf geschützten Bereich Die Beschreibung basiert auf dem schon existierenden Aktivitätsdiagramm 5.2, das zu einem nun viel komplexeren Aktivitätsdiagramm erweitert wird. Die Abbildung 5.10 stellt also die LRZ-interne Realisierung der Funktionalität ,,Zugriff auf geschützten Bereich“ dar.

In der Beschreibung wird nicht nur allein auf dienstspezifische Aspekte eingegangen (diese wurden allein bei der Beschreibung des Service View betrachtet), sondern vor allem auf die LRZ-interne Realisierung dieser Funktionalität. Dort existieren zwei große Verantwortlichkeitsbereiche: der Student (in der Rolle des Nutzers) und das LRZ (in der Rolle des Providers). Nach den Aktivitäten ,,Öffnen des Browsers“ und ,,WWW-Seite aufrufen“ (seitens des Nutzers) wird auf der Seite des LRZ der Virtuelle WWW-Server aufgerufen. Kommt von einem Client ein Request für eine bestimmte Seite bei dem WWW-Server an, so überprüft der Server zuerst, ob eine Zugriffskontrolle für diesen Bereich eingerichtet ist.

Wenn keine Zugriffskontrolle eingerichtet ist, dann wird der Zugriff auf die Seite gewährt (rechter Ast). Wenn aber diese eingerichtet ist, dann wird der Zugriff entsprechend dem Inhalt reglementiert (linker Ast).

Nachdem die Zugriffsinformationen gelesen worden sind, gibt es wieder drei Möglichkeiten, wie das System reagiert: falls das Verzeichnis Passwort-geschützt ist, falls es komplett gesperrt ist oder falls die Zugriffskontrolle über Domainnamen/IP-Adressen stattfindet.

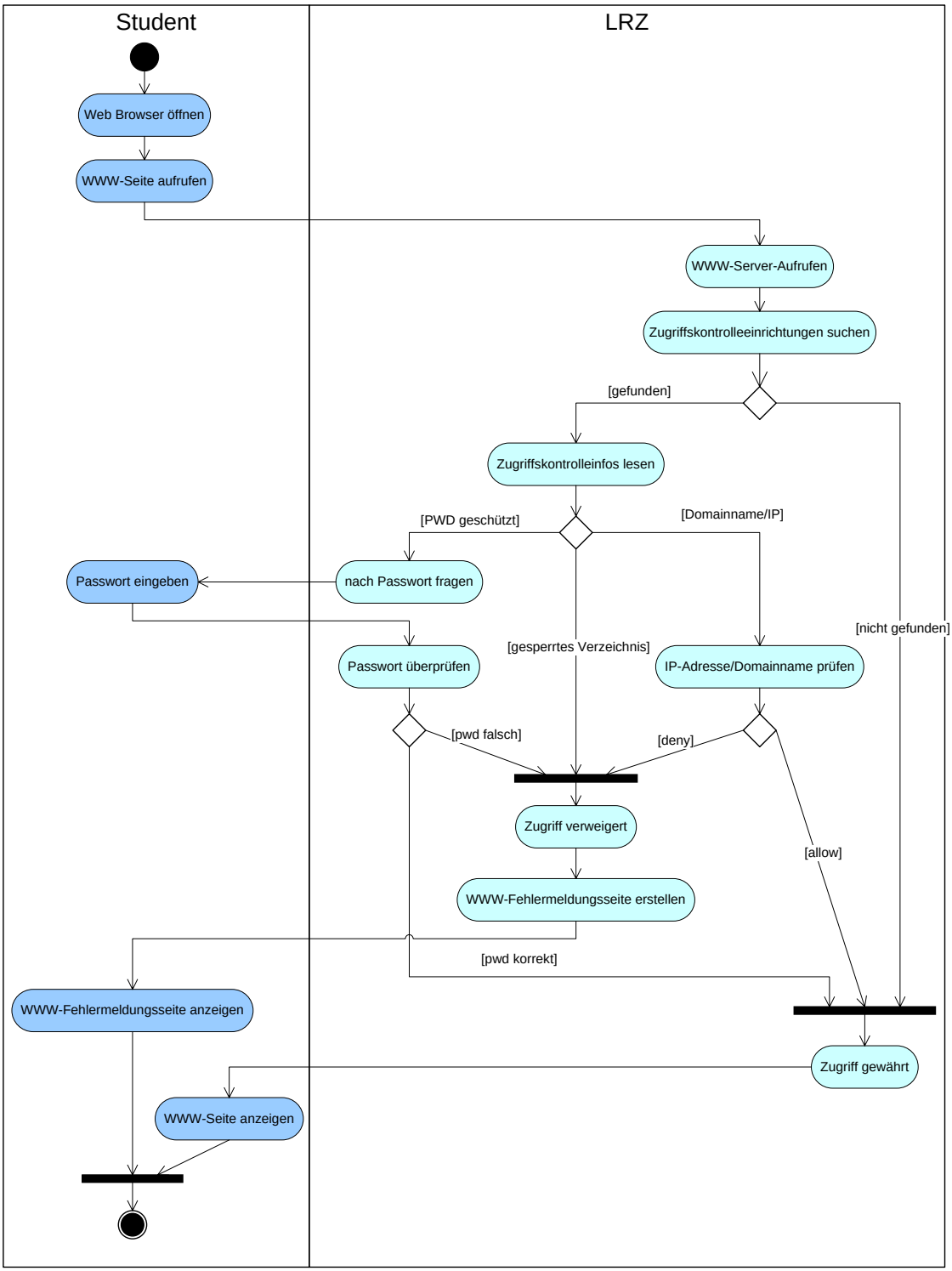

Abbildung 5.10: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Zugriff auf geschützten Bereich*

Wenn das Verzeichnis passwortgeschützt ist (linker Ast), dann wird nach dem Passwort gefragt. Der Student (User) gibt das entsprechende Passwort ein. Das eingegebene Passwort wird durch Vergleichen mit dem in AFS bzw. NFS gespeicherten Passwort überprüft. Im Falle, dass es nicht übereinstimmt mit dem Eintrag in AFS/NFS, wird der Zugriff auf der Web-Seite verweigert, sonst wird er gewährt.

Eine Passwortdatei, mit dem Namen `meine_pwd_datei.userpd`, wird mit Hilfe des Programms `/client/bin/htpasswd` in AFS erzeugt oder sie wird in NFS angelegt unter `/nfs/cgi/<a|u|t>/<Benutzerkennung>/webconfig/htpasswd`. Diese Datei enthält den (die) Namen des (der) autorisierten Benutzers zusammen mit dem jeweiligen, kodierten Passwort. Man kann sich auch selbst eine Passwortdatei erstellen, ohne die Benutzung oben genannten Programms, indem man mit einem Texteditor eine entsprechende Datei anlegt, die man dann per FTP an der entsprechenden Stelle unter AFS ablegt. Die benötigten, kodierten Passwörter kann man sich mit dem Programm 'Passwort-Codierer' erzeugen. Das Prozedere ist unter NFS (die geschützte Datei muss nur die Endung `.sec` haben und die Berechtigungen in der `htpasswd`-Datei eingetragen sein) viel einfacher wie unter AFS.

Die Datei `meine_pwd_datei.userpd` darf sich auf keinen Fall im WWW-Datenbereich befinden, sondern muss in dem Directory `$HOME/webserver/webconfig/pwd/` abgelegt werden. Dieses Verzeichnis sollte mit einer `.htaccess`-Datei geschützt sein, wie es oben beschrieben wurde und braucht mindestens Leserecht für den WWW-Server.

Ein Verzeichnis kann aber auch komplett für jeden WWW-Zugriff gesperrt (mittlerer Ast) werden. Der Grund für diese Sperrung ist die Tatsache, dass manche Verzeichnisse vom WWW-Server erreicht werden müssen, damit er dort bestimmte Konfigurationsdateien z.B. für CGI-Skripts oder für Passwortvergleiche lesen kann. Diese Dateien werden vom WWW-Server über das lokale File-System und nicht über eine URL eingelesen, d.h. dass der Server lediglich über das File-System eine Leseberechtigung und evtl. eine Schreibberechtigung benötigt. Diese Lese- bzw. Schreibberechtigung wird am LRZ über entsprechende AFS-Kommandos realisiert.

Die dritte Variante (rechter Ast) gibt die Möglichkeit, den Zugriff auf bestimmten IP-Adressen bzw. Domainnamen einzuschränken. Man kann diesen Zugriff gestatten oder verweigern. Am meisten wird die Möglichkeit dazubenutzt, explizit bestimmten Rechnern den Zugriff zu erlauben. In diesem Fall muss man erst den Zugriff im Allgemeinen verbieten (`deny from all`), und danach explizit bestimmten Adressen bzw. Domainnamen den Zugriff erlauben. Falls man explizit bestimmten Rechnern den Zugriff verweigern möchte, muss man analog vorgehen, jedoch konsequent die Begriffe 'allow' und 'deny' vertauschen.

In dem unteren Teil des Aktivitätsdiagramm 5.10 gibt es dann zwei Möglichkeiten, entweder, dass der Zugriff verweigert wird, und dann eine WWW-Fehlermeldungsseite angezeigt wird, oder der Zugriff gewährt wird und die aufgerufene Seite angezeigt wird.

Bemerkungen bzgl. Abhängigkeiten Mit dem obigen Aktivitätsdiagramm kann man offensichtlich mehrere Abhängigkeiten modellieren wie in dem Aktivitätsdiagramm des Service Views. Man kann hier bemerken, dass es sich um die Abhängigkeit zwischen dem Web Hosting Dienst und den Ressourcen AFS bzw. NFS (als Filesystem, aber auch als Authentifizierung) handelt, aber eine genaue Modellierung kann nicht erfolgen, weil zum Beispiel die zeitlichen Attribute fehlen. Man kann auch erkennen, dass es eine Abhängigkeit zwischen dem Web Hosting Dienst, AFS und NFS

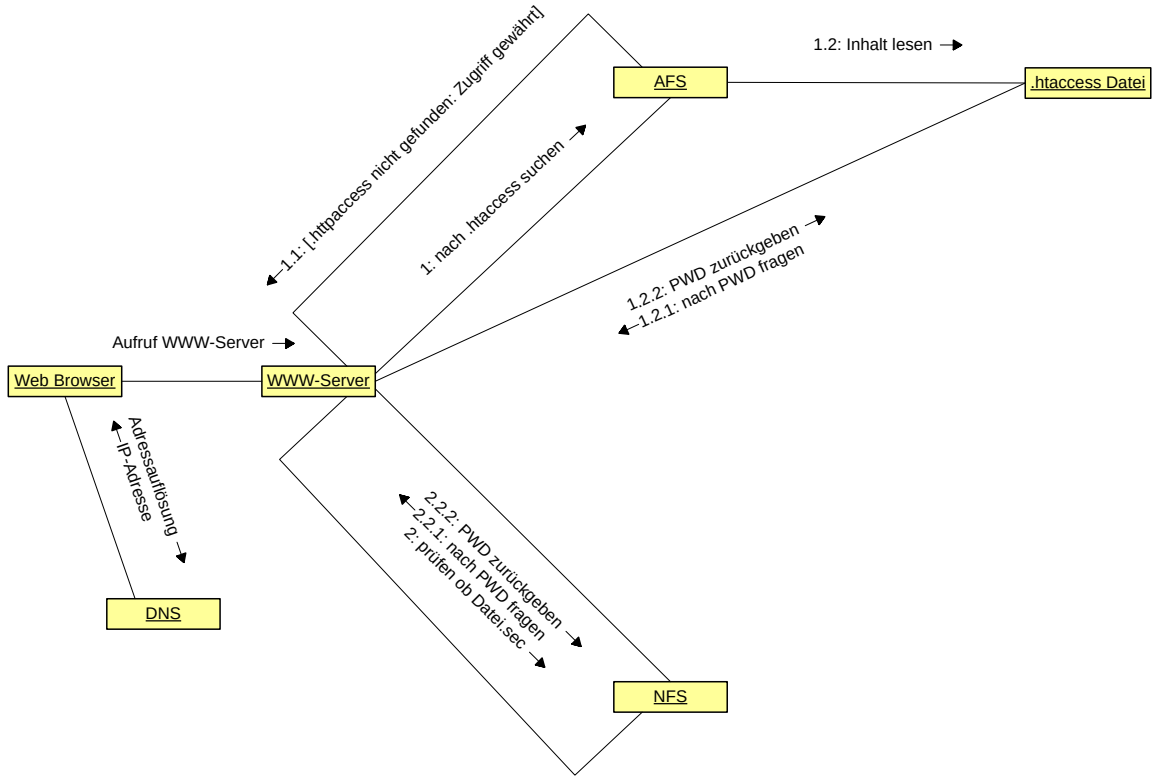

Abbildung 5.11: Kollaborationsdiagramm für die Realisierung der Funktionalität *Zugriff auf geschützten Bereich*

als Diensten gibt, aber diese können auch nicht genauer modelliert werden. Wie in 4.1.2.2 schon beschrieben für den Allgemeinfall, ist das Aktivitätsdiagramm für die Realization View immer noch nicht ausdruckstark. Dafür werden anschließend die entsprechenden Kollaborationsdiagramme entworfen.

Erstellung der Kollaborationsdiagramme Es werden im folgenden für diese interne Realisierung Kollaborationsdiagramme entworfen. Diese werden die Beziehungen zwischen den Ressourcen und Diensten, die zu der Realisierung der Funktionalität beitragen, widerspiegeln. In der Abbildung 5.11 wird das Kollaborationsdiagramm für die Realisierung der Funktionlität Zugriff auf geschützten Bereich dargestellt. Dort findet man folgende Objekte: WWW-Server, DNS, AFS, NFS, .htaccess Datei, htpasswd Datei, und Web Browser. Die in dem Web Browser eingegebene Adresse wird von DNS in eine IP-Adresse umgewandelt und zum Browser zurückgegeben. Dieser ruft dann den WWW-Server auf. Es wird entschieden, in welchen Dateisystem die gesuchte Datei sich befindet. Wenn die Datei im AFS liegt, dann wird die Nachricht 1 Suche nach .htaccess Datei zum AFS geschickt. Wenn die .htaccess Datei nicht gefunden worden ist, dann ist der Zugriff auf die Seite gewährt (Antwort 1.1). Wenn die .htaccess Datei gefunden worden ist, dann wird der Inhalt dieser Datei gelesen (Nachricht 1.2).

Wenn sich die Seite aber im NFS befindet, wird es die Nachricht 2 (prüfe ob die gesuchte Datei eine `.sec`-Endung hat) zum NFS geschickt. Wenn die Datei keine .sec-Endung hat dann ist der Zugriff auf der Seite gewährt (Antwort 2.1).

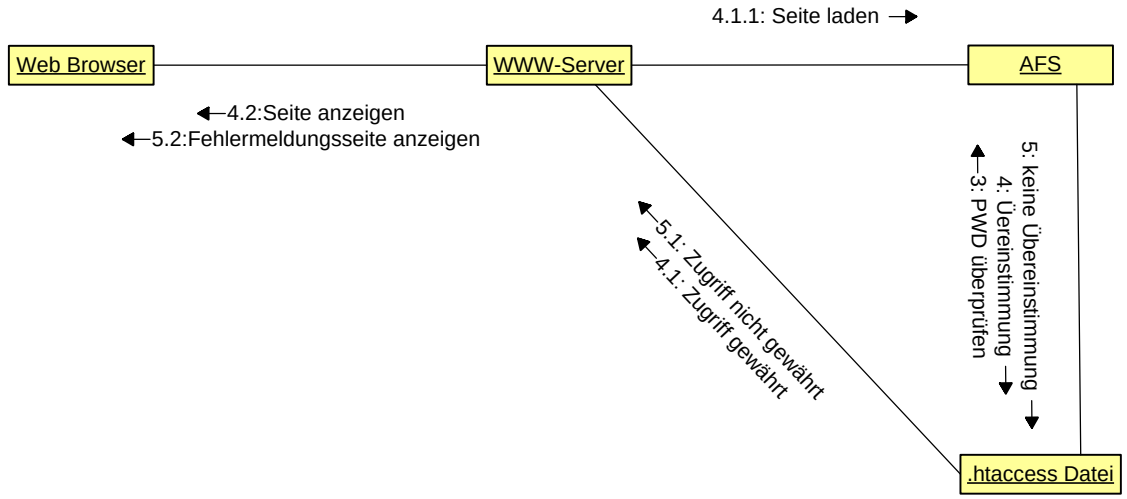

Abbildung 5.12: Ergänzung zur Abbildung 5.11. Verhalten des Systems nach Eingabe des Passwortes für die Seiten aus dem AFS

In dem Aktivitätsdiagramm 5.10 werden nach der Aktivität „Zugriffskontrollinformationen lesen“ drei mögliche Abläufe angegeben. Als erstes wird der Ast „passwortgeschützte Datei“ betrachtet. Die Abbildung 5.11 ist für jede der drei Verzweigungen im Aktivitätsdiagramm, bis inklusive Nachrichten 1.2 identisch. Differenzierungen kommen erst danach. Die Abbildung 5.12 gilt für den Fall, dass die Seite passwortgeschützt ist un sich in der AFS befindet und die Abbildung 5.13 gilt für den Fall, dass die Seite passwortgeschützt ist und sich im NFS befindet. Diese beide werden als Fortsetzung der Abbildung 5.11 dargestellt, so dass die Übersichtlichkeit der Darstellung beibehalten wird. Ebenso zeigt die Abbildung 5.14 die Fortsetzung für den Fall, gesperrtes Verzeichnis, und die Abbildung 5.15 für den Fall, Zugriffskontrolle über Domainname/IP-Adresse.

Passwortgeschützte Seite Für eine Seite, die sich im AFS befindet, sieht die interne Realisierung folgendermaßen aus: nach dem Lesen des Inhalts der .htaccess Datei wird nach dem Passwort gefragt (Nachricht 1.2.1 am WWW-Server in seiner Rolle als Access Point für den Web Browser(Client)). Die Antwort 1.2.2 von dem WWW-Server wird zurückgegeben und das Passwort wird dann im AFS geprüft (Nachricht 3). Es wird als Antwort entweder eine Übereinstimmung (Antwort 4) oder keine Übereinstimmung (Antwort 5) zurückgegeben.

Im Falle der Übereinstimmung (siehe Abbildung 5.12) wird der Zugriff auf die Seite gewährt (Antwort 4.1 an WWW-Server) und die Seite aus AFS (Antwort 4.1.1) geladen. Die Seite wird dann im Browser angezeigt (Antwort 4.2). Im Falle der Nichtübereinstimmung des Passwortes mit dem gespeicherten Eintrag in AFS, wird der Zugriff auf die Webseite nicht gewährt (Antwort 5.1). Es wird eine Antwort 5.2 an den Browser geschickt mit einer Fehlermeldungsseite.

Wenn sich die Seite im NFS befindet dann wird, wenn die Datei die Endung `.sec` hat, nach dem Passwort gefragt (Nachricht 2.2.1) und wird dann mit dem Passwort geantwortet (Antwort 2.2.2). Das Passwort wird in der Passwortdatei als nächstes überprüft (Nachricht 3 in der Abbildung 5.13) und die Antwort ist entweder eine Übereinstimmung (Antwort 4) oder eine Nichtübereinstimmung (Antwort 5). Es wird hier unterschieden zwischen Dienst und Ressource, dass heißt die `htpasswd`-Datei (als Authentisierungsdienst) ist eigentlich Bestandteil des NFS, aber es wird in diesem Kollaborationsdiagramm getrennt von NFS (Dateisystem) betrachtet. Es ist nur zufällig, dass im Fall des Szenarios beim LRZ kein übergeordneter Authentisierungsdienst vorhanden ist.

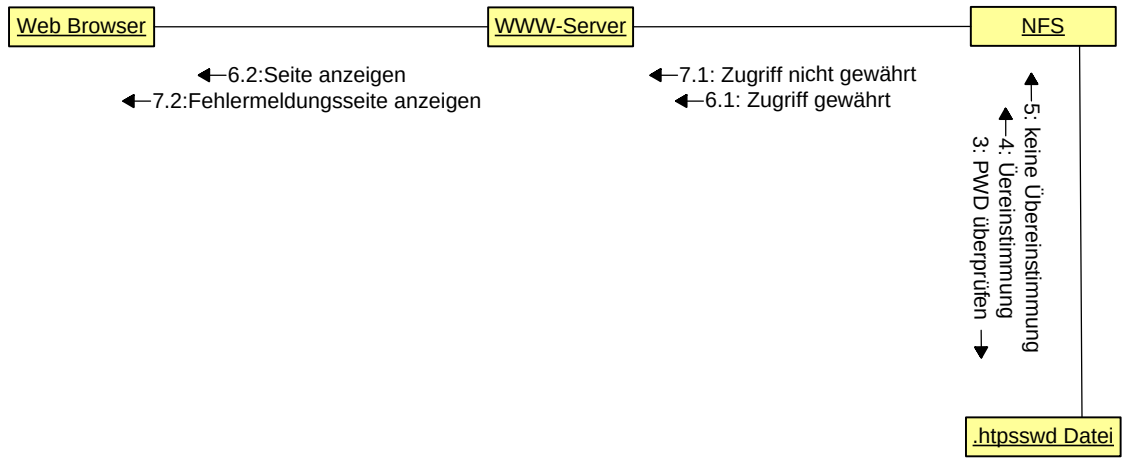

Abbildung 5.13: Ergänzung zur Abbildung 5.11. Verhalten des Systems nach Eingabe des Passwortes für die Seiten aus dem NFS

Gesperrtes Verzeichnis In der Abbildung 5.11 wird bis inklusive dem Lesen des Inhalts von Datei .htaccess alles genauso ablaufen. Danach wird die Nachricht 4 der Abbildung 5.14 folgen, indem dem WWW-Server mitgeteilt wird, dass der Zugriff auf dieses Verzeichnis verboten ist, und daraufhin wird in dem Browser eine Fehlermeldungsseite angezeigt.

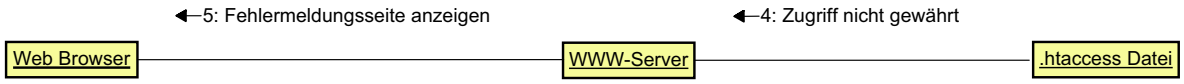

Abbildung 5.14: Ergänzung zur Abbildung 5.11. Verhalten des Systems für den Fall des gesperrten Verzeichnisses

Zugriffskontrolle über Domainname/IP-Adresse Für diesen Fall wird ebenso die Nachricht 3 (siehe Abbildung 5.15) nach dem Lesen des Inhalts von .htaccess Datei gesendet. (Dieser Vorgang gilt nur für den Fall, wenn die Seite im AFS gespeichert ist) Diese Nachricht ist an DNS gerichtet zur Ermittlung der IP-Adresse des Clients. Der DNS ermittelt diese Adresse, und dann wird in der Datei .htaccess überprüft, ob für diesen Domainname oder IP-Adresse der Zugriff auf die Seite erlaubt ist. Wenn ja, dann wird die Nachricht 6 an den WWW-Server zugeschickt, indem der Zugriff auf die Seite gewährt wird. Wenn aber die Antwort nein ist, dann wird die Nachricht 8 an den WWW-Server geschickt, wo mitgeteilt wird, dass der Zugriff nicht gewährt wird, und daraufhin wird eine Fehlermeldungsseite angezeigt (Nachricht 7.1).

Abhängigkeitsmodellierung nach dem Kollaborationsdiagramm Es werden hier alle der abhängigkeitsbezogenen Anforderungen analysiert:

Abhängigkeiten zwischen Ressourcen Offensichtlich sind auf diese Art Abhängigkeiten seht gut beschreibbar, z.B. sind der NFS und AFS als Dateisysteme abhängig voneinander für diese Funktionalität. In dem Fall, dass es sich um eine dynamische Seite handelt, wird als erstes das

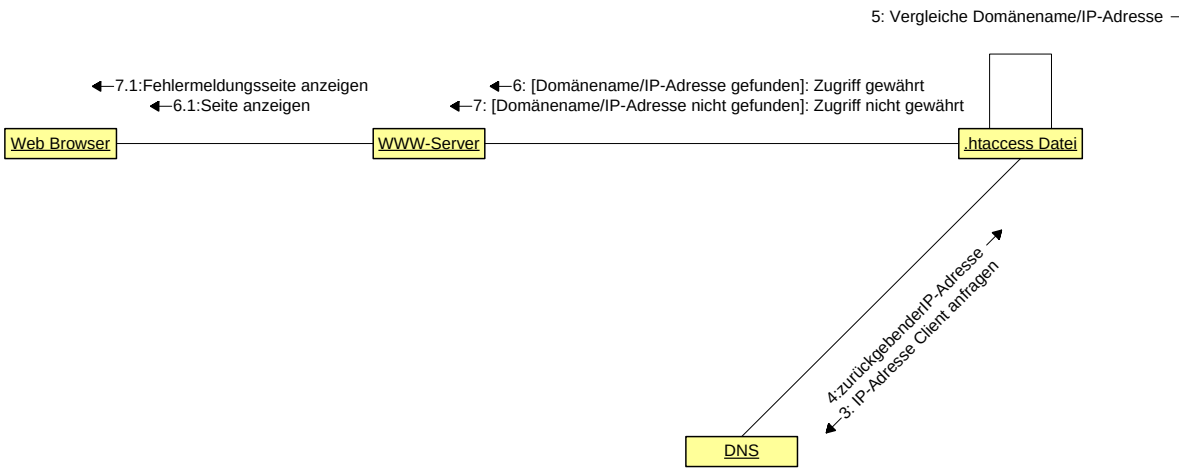

Abbildung 5.15: Ergänzung zur Abbildung 5.11. Verhalten des Systems für den Fall der Zugriffskontrolle über Domainname/IP-Adresse

AFS und dann das NFS durchsucht. Es geht hier um eine zeitliche Abhängigkeit. Es gibt noch eine andere Eigenschaft dieser Abhängigkeit (benutzt in der Authentisierung), wenn es sich um eine dynamische Seite handelt, die Passwort-geschützt ist. Das heißt, dass ein und dieselbe Abhängigkeit auf zwei unterschiedliche Weisen „genutzt“ wird: 1. zeitlich gesehen wird NFS nach AFS durchsucht; 2. um auf eine Passwort-geschützte dynamische Seite Zugriff zu erlangen, wird erst mit dem in AFS gespeicherten Passwort verglichen, und dann wird abhängig davon die Seite in NFS freigeschaltet oder nicht.

Abhängigkeiten zwischen Ressourcen und Diensten Diese Arten von Abhängigkeiten sind auch gut darstellbar mit einem Kollaborationsdiagramm. Beispielsweise kann man NFS als Resssource ansehen, die einen Authentisierungsdienst braucht, und dieser wird hier durch die Datei *.htpasswd* realisiert. Oder die Ressource Webserver, die von einem Dateisystem (als Dienst) wie AFS abhängig ist.

Abhängigkeiten zwischen Diensten und Subdiensten Diese Anforderung ist auch völlig erfüllt. Beispielsweise braucht der Web Hosting Dienst immer einen DNS-Dienst.

Dienstfunktionalität spezif. Abhängigkeiten Sind auch gut darzustellen, da für jede Funktionalität ein oder mehreren Kollaborationsdiagrammen dargestellt werden. Beispielsweise kann man sagen die Funktionalität Zugriff auf geschützten Bereich ist vom AFS oder NFS abhängig.

Statische Abhängigkeiten kann man auch ganz gut modellieren. Man merkt, dass ein Web Hosting Dienst eine DNS braucht.

Dynamische Abhängigkeiten ist nur teilweise erfüllt, da die Redundanzen bisher überhaupt nicht berücksichtigt worden sind. Trotzdem kann man mit dem Kollaborationsdiagrammen die dynamischen Abhängigkeiten ersichtlich machen. Je nachdem, was für eine Seite aufgerufen wird, wird die .htaccess Datei nur in AFS, aber auch in NFS gesucht. Das wird während des Ablaufs entschieden. Eine andere Sache ist auch die Entscheidung, was passiert, nachdem der Inhalt der Datei .htaccess gelesen worden ist. Wenn das Verzeichnis komplett gesperrt ist, dann werden keine anderen Aktionen durchgeführt, sondern es erscheint direkt die Fehlermeldung. Wenn es sich aber um eine passwortgeschützte Seite oder um eine Zugriffs-

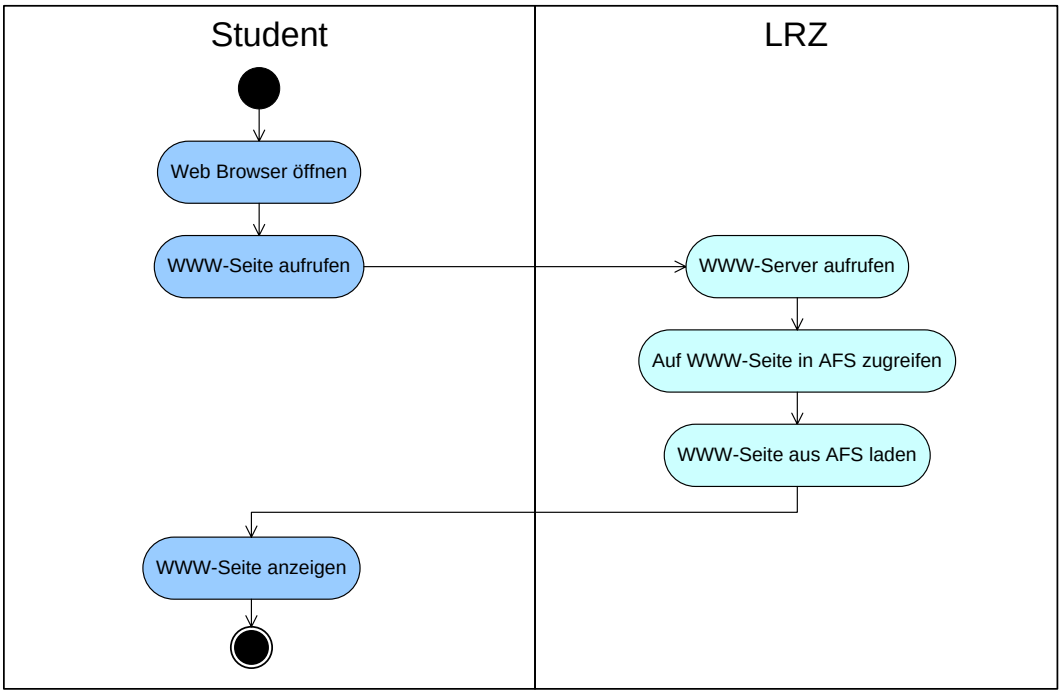

Abbildung 5.16: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Zugriff auf statische Seiten*

kontrolle über Domainname/IP-Adresse handelt, werden andere Aktivitäten benötigt. Bei der Zugriffskontrolle durch Passwort wird dann dynamisch der Authentisierungsdienst aufgerufen, und wenn es eine Passwortübereinstimmung gibt, dann wird die Abhängigkeit zwischen dem WWW-Server und AFS bzw. NFS dynamisch ausgenutzt, sonst nicht (in diesen Fall wird nur eine Fehlermeldung in Browser erscheinen). Dasselbe gilt auch bei der Zugriffskontrolle über Domainname/IP-Adresse; da wird noch einmal die Abhängigkeit zur DNS ausgenutzt, und wenn die IP-Adresse bzw. Domainname mit dem aus der Datei .htaccess übereinstimmen, und nur dann, wird noch auf AFS oder NFS zugegriffen (je nachdem, um was für eine statische oder dynamische Seite es sich handelt).

5.1.2.1.2 Zugriff auf statische Seiten Für die interne Realisierung der Funktionalität ,,Zugriff auf statischen Seiten“ wurde als Referenz die schon existierende Abbildung 5.3 verwendet. Diese wurde erweitert durch den Verantwortlichkeitsbereich LRZ (in Allgemeinfall Provider). Hiermit stellt die Abbildung 5.16 die interne Realisierung dieser Funktionalität dar.

Die ersten Aktivitäten ,,Öffnen des Browsers“ und ,,WWW-Seite aufrufen“ werden von Student (user) durchgeführt. Danach wird auf den Verantwortlichkeitsbereich LRZ gewechselt. Hier wird der WWW-Server aufgerufen, und danach folgt der Zugriff aufs AFS. Im AFS sind die statischen Seiten gespeichert. Nach dem Finden der Seite wird diese geladen und im Browser angezeigt. Diese ist eine der einfachsten Nutzungsfunktionalitäten.

Bemerkungen bzgl. Abhängigkeiten Die Abhängigkeit zwischen dem WWW-Server und dem AFS ist hiermit eindeutig dargestellt. Es gibt aber noch einige Abhängigkeiten, die mit dem Aktivitäts-

diagramm nicht dargestellt werden können, und deswegen wird anschließend ein Kollaborationsdiagramm erstellt als Verfeinerung.

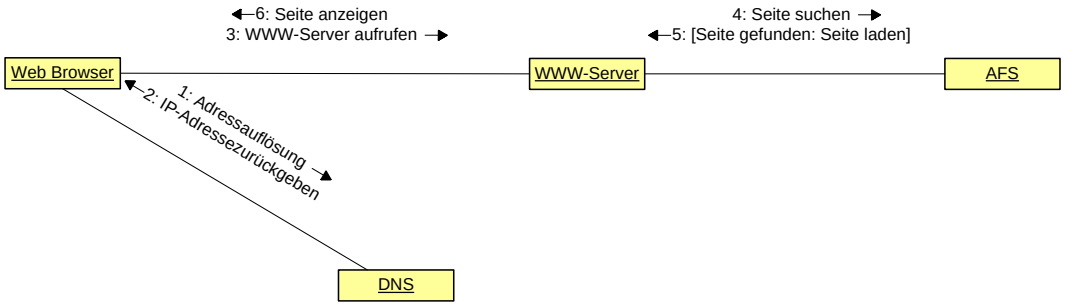

Abbildung 5.17: Kollaborationsdiagramm für die Realisierung der Funktionalität *Zugriff auf statische Seiten*

Erstellung der Kollaborationsdiagramme In der Abbildung 5.17 ist die interne Realisierung der Funktionalität „Zugriff auf statischen Seiten“ dargestellt in Form eines Kollaborationsdiagramms. Die Objekte die bei der Realisierung beteiligt sind, sind: WWW-Server, DNS, AFS, und Web Browser. Die Nachricht 1, forward lookup (die Name muss in einer IP-Adresse umgewandelt werden) wird zum DNS geschickt. Nachdem der DNS-Server die Domainname zur IP-Adresse umgewandelt hat und sie zurück zum Web Browser geschickt hat (Nachricht 2), ruft der Web Browser den WWW-Server auf (Nachricht 3). Der WWW-Server sucht dann die Seite in AFS (der Speicherort für statische Seiten). Auf dieser letzte Nachricht 4 folgt eine Antwort 5: Seite laden, und danach eine Antwort 6. zum Web Browser durch den die Seite angezeigt wird.

Abhängigkeitsmodellierung durch Kollaborationsdiagramm In dem oberen Kollaborationsdiagramm kann man die existierenden Abhängigkeiten sehr einfach darstellen. Allerdings sind bei dieser Funktionalität ziemlich wenige Abhängigkeiten in Betracht zu ziehen. Es gibt da die statische Abhängigkeit zwischen dem Web Hosting Dienst und dem DNS-Service. Egal in welchem Kollaborationsdiagramm, ist diese Abhängigkeit immer anwesend (da der DNS-Dienst ein notwendiger Dienst für die Realisierung ist). Eine andere Abhängigkeit ist die zwischen dem Web Server und dem AFS. Das ist eine statische Abhängigkeit, weil jedes mal, wenn der Web Server aufgefordert wird, eine Seite zu finden, wird er im AFS suchen (auch dann, wenn es sich um eine dynamische Seite oder um einen geschützten Bereich handelt).

Dynamische Abhängigkeiten sind für diesen Fall nicht notwendig. Die einzigen Probleme die auftreten könnten bei dem Aufruf einer statischen Seite wären: dass der WWW-Server oder das AFS (als verteilter Datei-Dienst) nicht erreichbar sind. Für diesem Fall wird dann dynamisch die Problemmanagement Funktionalität durchgeführt werden.

5.1.2.1.3 Zugriff auf dynamische Seiten

Aufgrund der Abbildung 5.4 die das Aktivitätsdiagramm für die Funktionalität Zugriff auf dynamische Seiten darstellt (Service View), wurde die Abbildung 5.18 hergeleitet indem man auch das providerinterne Vorgehen repräsentiert. Nachdem der Nutzer den Web Browser öffnet und z.B. ein bestimmtes WWW-Formular aufruft, wird auf der Providerseite der WWW-Server aufgerufen (die *virt*-Konfiguration). Anschließend wird auf die Seite, wie bei den statischen Seiten, zugegriffen, um das WWW-Formular zu laden.

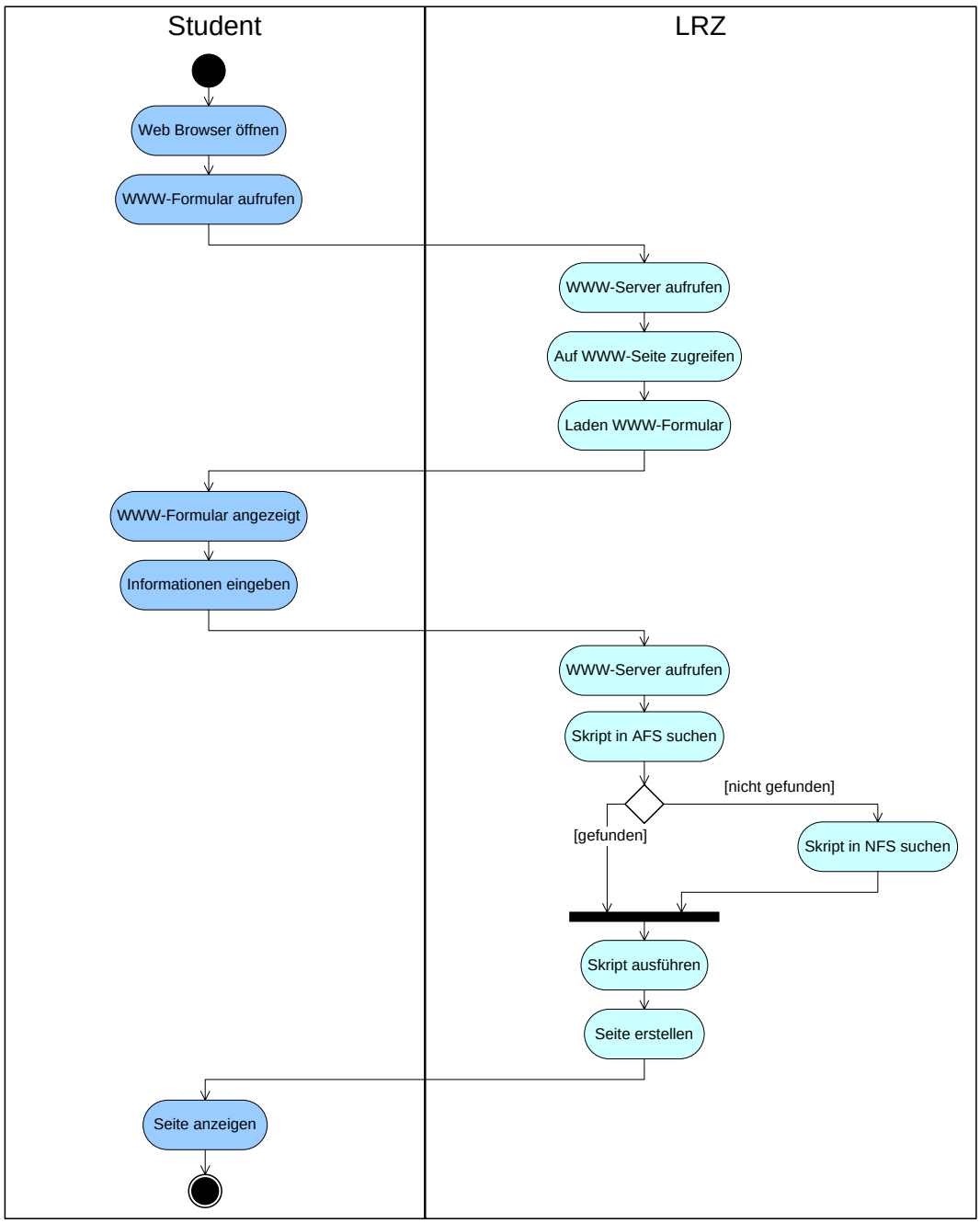

Abbildung 5.18: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Zugriff auf dynamische Seiten*

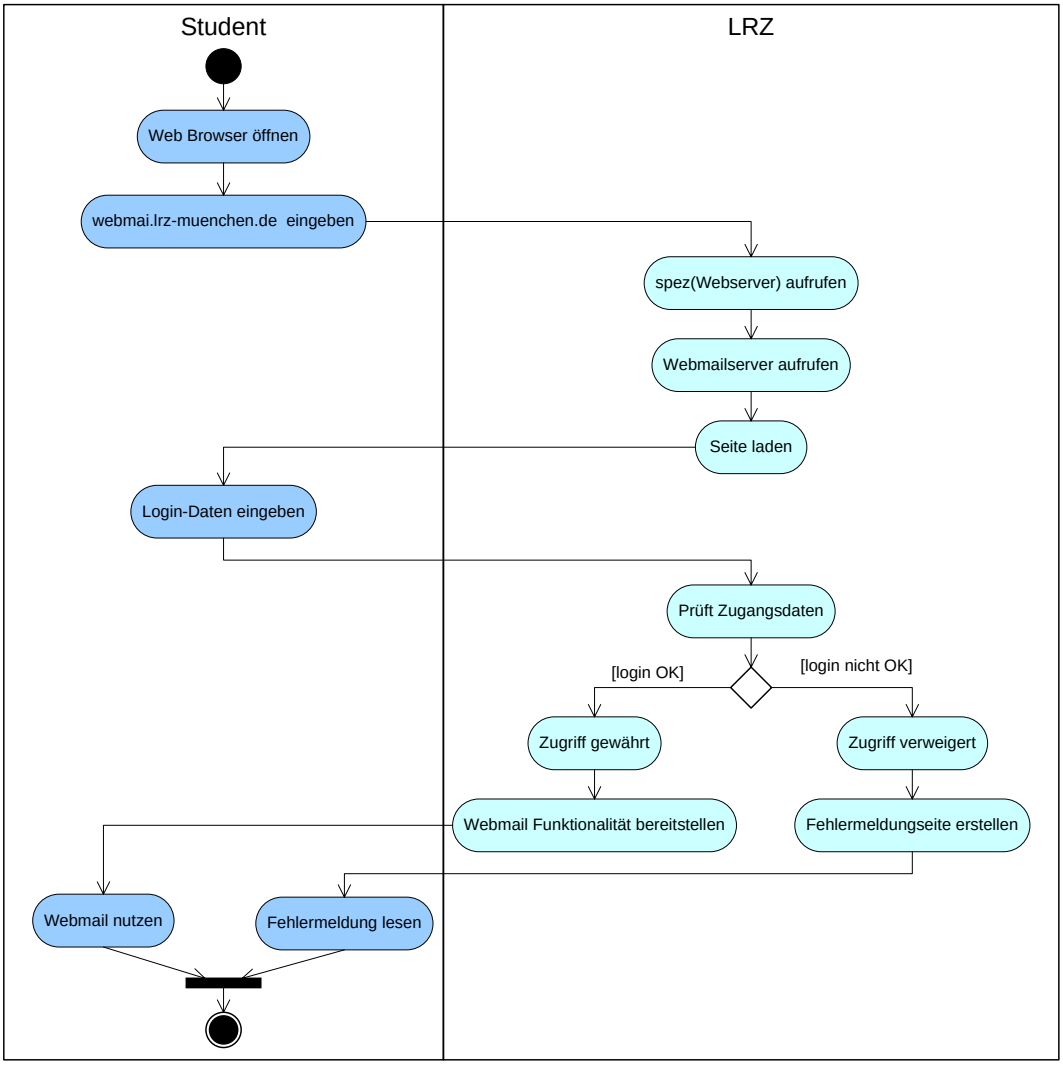

Abbildung 5.19: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Webmail*

Das Formular wird dann im Browser angezeigt und der Nutzer kann dann seine Informationen eingeben und verschicken. Daraufhin wird der WWW-Server wieder aufgerufen, und dann das entsprechende Bearbeitungsskript, gesucht erst im AFS und, wenn es nicht gefunden wird im NFS. Wenn das Skript gefunden worden ist (entweder ins AFS oder ins NFS), wird es ausgeführt, und die dynamische Seite wird erstellt und angezeigt.

Bemerkungen bzgl. Abhängigkeiten Man kann hier beobachten dass es sich um eine Abhängigkeit zwischen Webserver und einem/zwei Filesystem(en) –AFS und/oder NFS– handelt aber auch zwischen den zwei Dateisysteme. Das Diagramm hilft schon mehr für das Finden und Modellieren der Abhängigkeiten aber immer noch nicht genügend. Dafür werden die Kollaborationsdiagramme zur Verfeinerung benutzt. Die Kollaborationsdiagramme sind ähnlich wie beim Funktionalitäten *Zugriff auf geschützten Bereich* und *Zugriff auf statischen Seiten*

5.1.2.1.4 Webmail Funktionalität Das Aktivitätsdiagramm in der Abbildung 5.5 wird in der Abbildung 5.19 mit Aspekten der providerinternen Realisierung der Funktionalität Webmail erweitert.

Nachdem der Nutzer den Web Browser öffnet und die `webmail.lrz-muenchen.de`-Adresse der Webmailseite eingibt, wird auf der Providerseite die **spez**-Konfiguration des Webservers aufgerufen. Teil der **spez**-Konfiguration ist auch der Webmail Server, der hiermit auch aufgerufen wird. Die Webmailseite wird geladen und die Login Seite wird angezeigt. Nachdem der Nutzer das Kennwort und das Passwort eingegeben hat, werden seitens des LRZ die Logindaten überprüft. Wenn die Logindaten korrekt sind, wird die Webmail Funktionalität bereitgestellt. Wenn die Logindaten inkorrekt sind wird eine Fehlermeldungseite erstellt, und daraufhin kann der Nutzer entweder diese neu eingeben oder die Seite verlassen.

Offensichtlich handelt es sich hier um Abhängigkeiten zwischen der **spez**-Konfiguration und dem Webmail Server, aber auch zwischen der Webmail Funktionalität und dem Authentisierungsdienst, aber auch am wichtigsten, zwischen der Webmail Funktionalität und demE-Mail Dienst. Auf ein Kollaborationsdiagramm wird in diesen Fall (aus Platzgründen) verzichtet, aber das Abhängigkeitsmodell wird hierfür angegeben (siehe Abschnitt 5.1.3.3).

5.1.2.2 Service Management und Service Management Logic

Service Management und Service Management Logic sind für die interne Realisierung der Managementfunktionalität zuständig. Für diese Darstellung der providerinternen Aspekte, die zur Realisierung des Managements des Web Hosting Dienstes beitragen, werden als Grundlage die im Abschnitt 5.1.1.2 vorgestellten Aktivitätsdiagramme betrachtet. Diese werden erweitert zu detaillierten Aktivitätsdiagrammen, in denen die Providerseite vorwiegend dargestellt wird. Zu jedem der Anwendungsfälle werden dann zusätzlich, zur Bestimmung der Abhängigkeiten, auch Kollaborationsdiagramme erstellt.

5.1.2.2.1 Einrichten eines virtuellen WWW-Servers Nachdem die AFS-Kennung besorgt worden ist, kann der Nutzer das Homepage-Tool, das zur Verfügung steht, öffnen um seine provisorische Webseite einzurichten. Dieses Tool wird durch Aufruf des Web-Servers (genauer die **lrz**-Konfiguration) realisiert. Die provisorische Webseite wird im AFS abgelegt (Prozedur ist beim Service View schon beschrieben). Nachdem diese Seite angeschaut wurde kann man (empfolen) die Datei **index.html** überschreiben und einen sinnvollen Titel für die neu kreierte Präsenz angeben.

Die nächste Aktion ist die Eintragung des Namen des virtuellen Servers ins DNS durch den zuständigen DNS-Verwalter. Als letztes wird durch Senden des Antrags auf einen virtuellen Webserver die Einrichtungsaktion beendet.

Bemerkungen bzgl. Abhängigkeiten Man kann daraus die Komponenten, die dazu bei der Realisierung dieser Funktionalität beitragen, erkennen. Zwischen dem Webserver (**lrz**-Konfiguration) und AFS gibt es mit Sicherheit eine Abhängigkeit, auch gibt es eine Abhängigkeit zum DNS-Dienst (obwohl diese hier mehr organisatorische Natur hat). Mehr kann man aus der Aktivitätsdiagramm erkennen.

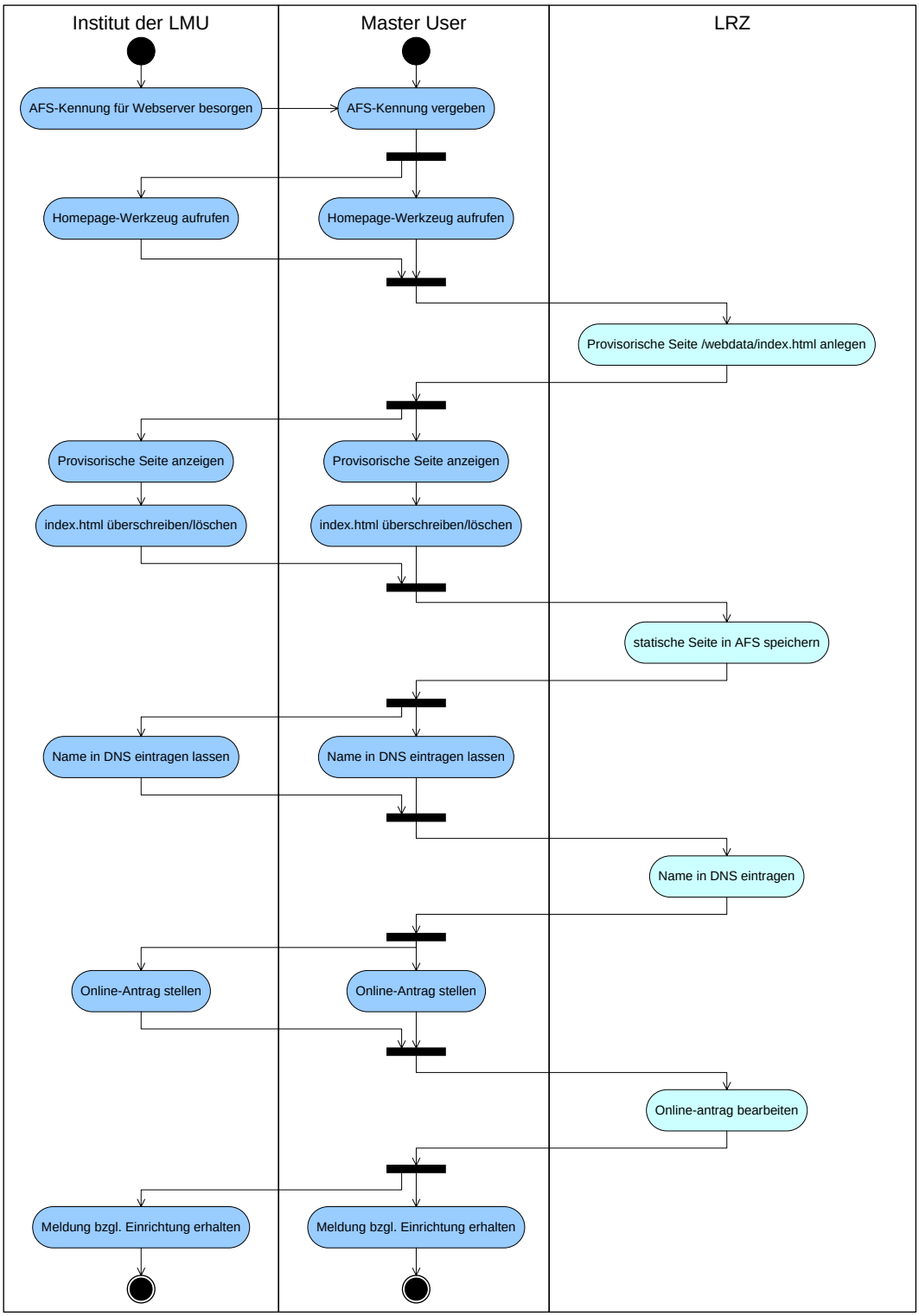

Abbildung 5.20: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Einrichten eines virtuellen WWW-Servers*

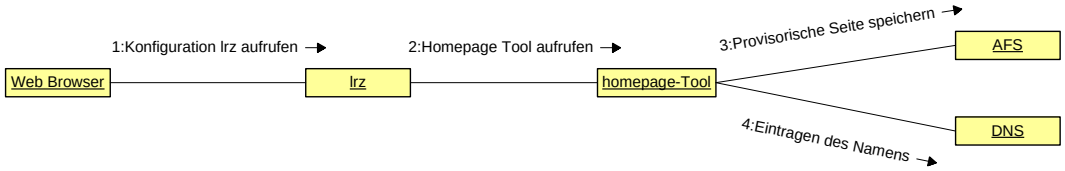

Abbildung 5.21: Kollaborationsdiagramm für die Realisierung der Funktionalität *Einrichten eines virtuellen WWW-Servers*

Erstellung der Kollaborationsdiagramme Das Kollaborationsdiagramm in der Abbildung 5.21 stellt die interne Realisierung der Funktionalität *Einrichten eines virtuellen WWW-Servers* dar. Nachricht 1 bezieht sich auf dem Aufruf der **lrz**-Konfiguration. Das Homepage-Tool, Teil der **lrz**-Konfiguration, ist für die Einrichtung neuer Webserver zuständig. Nachricht 2 ruft das homepage-Tool auf. Es wird dann eine neue provisorische Webseite erstellt, die im AFS gespeichert wird (Nachricht 3). Nach der Speicherung der Seite wird der Name des Webservers im DNS eingetragen. Die zuständige Person für den DNS-Dienst trägt dies ein und daher ist diese Abhängigkeit mehr eine organisatorische wie eine funktionale. Nichtsdestotrotz ist diese Abhängigkeit wichtig für die Einrichtung neuer Server.

Bemerkungen bzgl. Abhängigkeiten Abhängigkeiten zwischen dem homepage-Tool und der **spez**-Konfiguration des Webservers, zwischen der Funktionalität Einrichten eines virtuellen WWW-Servers und AFS und DNS sind ersichtlich. Eine ausführliche Darstellung dessen wird im Abschnitt 5.1.3.4 realisiert.

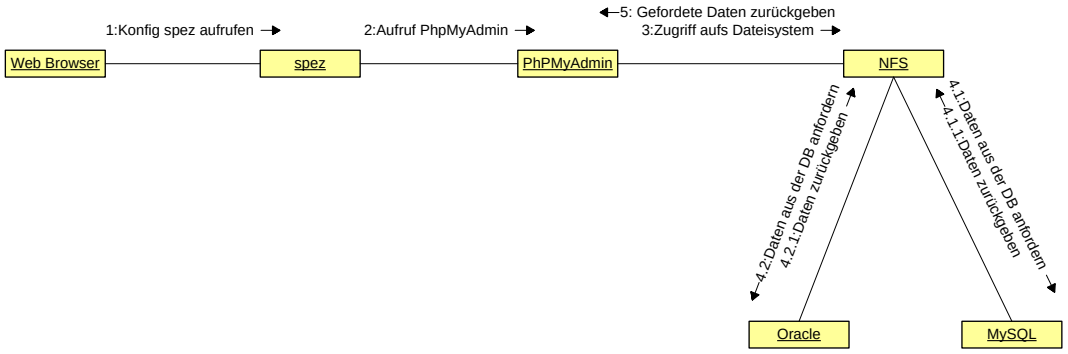

Abbildung 5.22: Kollaborationsdiagramm für die Realisierung der Funktionalität *Zugriff auf die Datenbanken über das PhpMyAdmin-Tool*

5.1.2.2.2 Zugriff auf die Datenbanken über das PhpMyAdmin-Tool Die Aktivitätsdiagramme sind hier nicht mehr dargestellt, aber sie würden auf eine ähnliche Weise wie für die vorherigen Funktionalitäten aufgebaut. Es wird hier nur das Kollaborationsdiagramm (siehe Abbildung 5.22) dargestellt und beschrieben. Indem der Kunde das PhpMyAdmin Tool aufruft, ruft schließlich der Web Browser die **spez**-Konfiguration des Webservers auf. Das Tool ist Teil dieser Konfiguration. Über das PhpMyAdmin Tool wird der Zugriff auf das Dateisystem (NFS) gewährleistet (Nachricht 3). Das Dateisystem fordert dann die gefragten Daten aus der Datenbanken an. Das PhpMyAdmin Tool hat Zugriff auf die Oracle und die MySQL Datenbanken. Daher verzweigt sich die Nachricht von

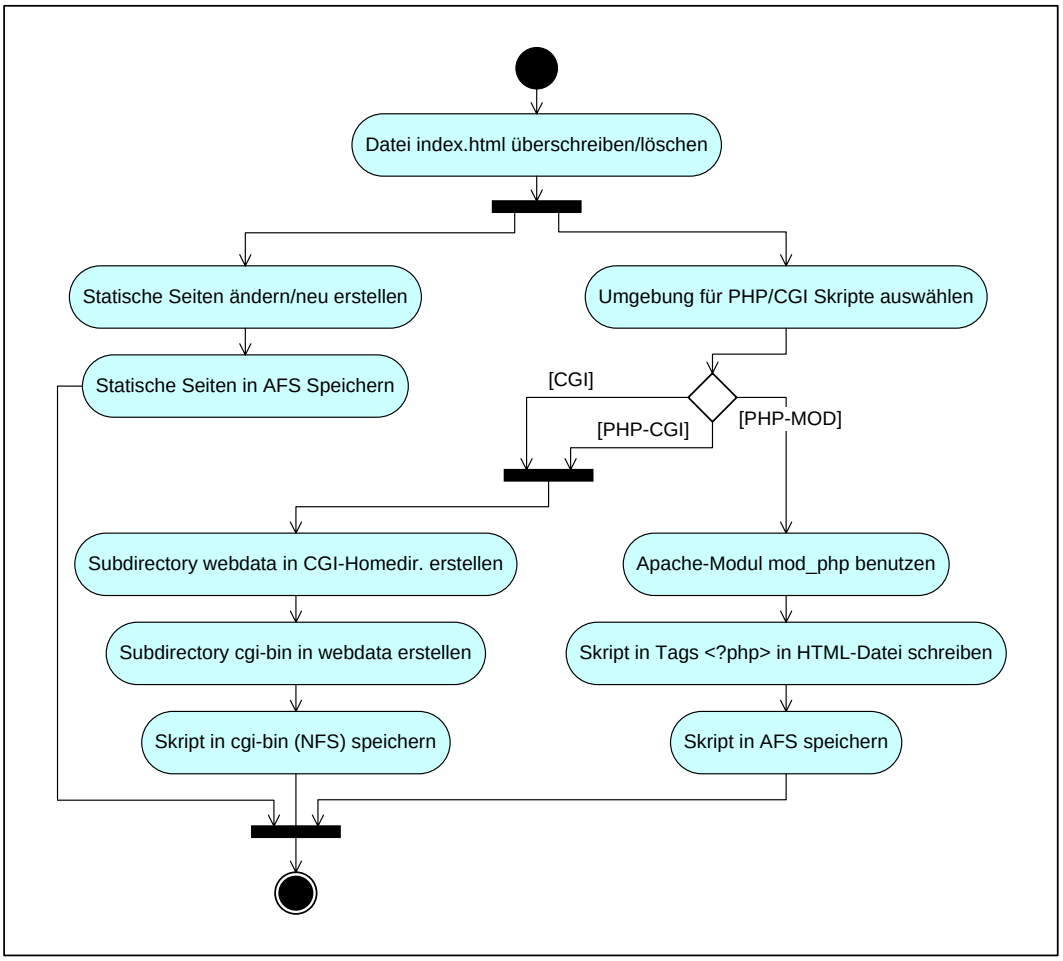

Abbildung 5.23: Aktivitätsdiagramm der internen Realisierung der Funktionalität *Einrichten/Ändern von Webseiten*

NFS aus in 4.1, inder die Daten aus der MySQL Datenbank oder 4.2 aus der Oracle Datenbank angefordert werden. Die Antworten für die Rückgabe der Daten sind entsprechend 4.1.1 aus der MySQL Datenbank und 4.2.1 aus der Oracle Datenbank.

Bemerkungen bzgl. Abhängigkeiten Man kann hier schon sehr gut die Abhängigkeiten ablesen. Es geht um Abhängigkeiten zwischen dem PhpMyAdmin Tool und dem Webserver (genauer die Konfiguration **spez**) aber auch zwischen der Funktionalität an sich und dem Dateisystem (auch als Funktionalität gesehen). Ebenfalls sind ersichtlich die Abhängigkeiten zwischen dem Dateisystem und den Datenbanken. Eine genaue Beschreibung davon und die möglichen Interaktionen zwischen diesen Abhängigkeiten werden im Abschnitt 5.1.3.5 dargestellt.

5.1.2.2.3 Einrichten/Ändern von Webseiten Als erste Aktion muss die **index.html**-Datei überschrieben werden, wenn es sich um die erste Änderung des erstellten Webseite handelt. Wenn das nicht der Fall ist, dann geht man direkt zum Schritt zwei. Hier trennen sich die möglichen Varianten der Änderung, je nachdem, ob es sich um statische oder dynamische Seiten handelt. In Falle

der statischen Seiten werden die Dateien im AFS-Cluster gespeichert. Im Falle der dynamischen Seiten gibt es mehreren Varianten wo diese gespeichert werden, weil unterschiedliche Konfigurationen und Einstellungen benötigt werden, je nachdem welche Art von Skript man benutzt. Es gibt viele Skriptsprachen die sehr bequem in der HTML-Webseiten integriert werden.

Die CGI-Skripten und PHP-Seiten bieten aus Sicht eines Web-Entwicklers sehr unterschiedliche Möglichkeiten, dynamische Inhalte zu erzeugen, aber aus technischer Sicht sind sie annähernd gleich, so dass man sie gemeinsam betrachten kann. In gewisser Weise sind PHP-Seiten ein Spezialfall von CGI-Skripten. Jeder, der einen virtuellen Server am LRZ betreibt, erhält automatisch die Möglichkeit, sowohl PHP-Skripte als auch CGI-Skripte einzusetzen. Ein gesonderter Antrag ist nicht notwendig.

Auf den virtuellen Webservern am LRZ können Skripte in folgenden Umgebungen verwenden werden:

CGI-Skripte (über einen CGI-Wrapper unter NFS) Das CGI-Homedirectory ist auf den Workstations `sun1.lrz-muenchen.de` und `sun2.lrz-muenchen.de` und am FTP-Server des LRZ (`ftp.lrz-muenchen.de`) über den Pfad `/nfs/cgi/<a|u|t>/<Benutzerkennung>` erreichbar. Der Anfangsbuchstabe der Benutzerkennung gibt Auskunft darüber, in welchem Directory (a, u oder t) sich das Homedirectory befindet. Im CGI-Homedirectory befindet sich ein Subdirectory mit dem Namen **webdata** und in diesem wiederum ein Subdirectory **cgi-bin**. Das Directory **webdata** enthält den Verzeichnisbaum mit den notwendigen Daten, das Subdirectory **cgi-bin** die CGI-Skripte (cgi-bin darf ebenfalls in Subdirectories untergliedert sein). Die Voraussetzungen für die Ausführung von CGI-Skripten sind: Ablage im CGI-Bereich unter: `/nfs/cgi/<a|u|t>/<Benutzerkennung>/webdata/cgi-bin/`, Dateiname beliebig (Sonderbehandlung bei Endung .php), Datei ausführbar (x-Bit gesetzt), Datei ausführbar, Eigentümer Datei = `Kennung Webserver` und Gruppe Datei = `Gruppe Webserver`.

PHP-Skripte (über einen CGI-Wrapper unter NFS) PHP-Skripte im CGI-Bereich werden gesondert behandelt, falls ihr Dateiname wie üblich auf `.php` endet. In diesem Fall sollte die Skriptdatei im Gegensatz zu sonstigen CGI-Skripten nicht ausführbar sein (kein x-Bit) und dann auch nicht den Pfad zum PHP-Interpreter in der ersten Zeile enthalten. Die Voraussetzungen für die Ausführung von PHP-Skripten im CGI-Bereich sind also: Ablage im CGI-Bereich unter: `/nfs/cgi/<a|u|t>/<Benutzerkennung>/webdata/cgi-bin/php`, Dateiname endet auf `.php`, Datei nicht ausführbar (kein x-Bit), Eigentümer Datei = `Kennung Webserver`, Gruppe Datei = `Gruppe Webserver`.

PHP-MOD (PHP-Skripte über das Apache-Modul mod_php unter AFS) PHP-Skripte können im Dokumentenbereich (also unter AFS) abgelegt werden, in dem auch die HTML-Seiten, Bilder etc. liegen. Der PHP-Code kann innerhalb der Datei an beliebiger Stelle eingefügt werden, oder auch ganz ohne HTML. Sie werden vom Webdämon an der Endung `.php` erkannt. Für die Ausführung sucht der Webdämon in den AFS-Verzeichnissen automatisch nach Dateien index.htm(l) (was er als statische HTML-Seite sofort ausliefert) oder index.php (was als PHP-Skript interpretiert wird). Die Voraussetzungen für die Ausführung von PHP-Skripten im Dokumentenbereich sind also: Ablage im Dokumentenbereich `/afs/lrz-muenchen.de/home/<a|u|t>/<Benutzerkennung>/webserver/webdata/`, Dateiname endet auf `.php` und AFS-Leserecht für Webdämon (“somebody“).

Abhängigkeiten, die hier in Betracht gezogen werden können, bestehen zwischen der Funktionalität Einrichten/Ändern von Webseiten und dem AFS bzw. NFS, aber auch vom Authentisierungsdienst.

Es gibt eine große Ähnlichkeit zur internen Realisierung der Funktionalität Zugriff auf Seiten bzw. Zugriff auf geschützten Bereich. Auf diesem Grund werden in dieser Arbeit keine weitere Aspekte der Abhängigkeiten für diese Funktionalität betrachtet.

5.1.2.2.4 Bearbeitung eines Trouble Tickets Es wird eine Erweiterung des in 5.9 dargestelltes Diagramm realisiert. Abbildung 5.24 stellt die Vorgehensweise vom Eingang bis zur Lösung eines Trouble Tickets dar.

Die Nachricht bezüglich einer Störung wird per Telefon oder per Hotmailformular an das LRZ eingereicht. Die Meldung wird dann vom Operateur aufgenommen und mit Hilfe des ARS-Systems wird das Trouble Ticket erst einmal erfasst und in der Datenbank gespeichert (das ist auch der erste Zustand den das TT nehmen kann). Eine automatische E-Mail an den betroffenen Studenten folgt. In dieser E-Mail werden Informationen über das gemeldete Problem mitgeteilt. Sehr wichtig ist dabei den TT-Identifikationsnummer mit Hilfe deren der Bearbeitungszustand des TT jederzeit nachgefragt werden kann.

Nicht nur an den Student wird eine Mail geschickt, sondern auch den Verantwortlichen am LRZ. Dieser wird dann ab diesem Zeitpunkt das TT bearbeiten. Die Bearbeitung wird auch priorisiert und zwar die *kritischen* TTs müssen in maximal 4 Stunden bearbeitet werden (das heißt, dass dieses Problem einen großen Teil des Netzes oder die Arbeit vieler Nutzer beeinträchtigt.

Während dieser Phase befindet sich das Trouble Ticket im Zustand *in Bearbeitung*, wenn man effektiv an dem Finden des Fehlers arbeitet, oder im Zustand *Ausgesetzt* wenn man nicht direkt an diesen arbeitet.

Nach gewisser Zeit wird der Fehler ausfindig gemacht und daraufhin wird der Zustand des TTs zu *Fehler gefunden*. Der Fehler wird danach beseitigt, und im Falle dass der Nutzer in der ursprünglichen Formular angegeben hat dass er um eine Antwort bittet, wird ihm die Antwort zu seinem Problem zugeschickt. Sonst wird der Trouble Ticket abgeschlossen und hiermit geht der TT in den Zustand *geschlossen*.

Bemerkungen bzgl. Abhängigkeiten Man kann natürlich anhand diesem Aktivitätsdiagramm erkennen, dass eine Abhängigkeit zwischen den Webserver (mit seiner speziellen Konfiguration **ars**) und der Datenbank besteht. Man kann sich jedoch diese Abhängigkeit nicht vertieft anschauen.

Realisierung der Kollaborationsdiagramme In der Abbildung 5.25 sind die Beziehungen zwischen den unterschiedlichen Ressourcen und Diensten, die dazu beitragen, dass die Funktionalität Bearbeitung eines Trouble Tickets realisiert werden kann.

Die Anfrage wird von einen *Client* zur **ars** (Action Request System) gesendet (Nachricht 1). Dieser Client kann entweder eine Person (direkt am Schalter), das Telefon oder das Hotmailformular auf der Webseite des LRZ sein.

Das entsprechende Trouble Ticket (aufgrund der Anfrage) wird erfasst (Nachricht 2) und danach in der Oracle Datenbank gespeichert. Das Trouble Ticket wird dann zur verantwortliche Person weitergeleitet (Nachricht 4) und nach dem Herausfinden des Fehlers wird die Antwort 5, Fehler gefunden, übermittelt. Hiermit wird der TT geschlossen (Nachricht 6) und gespeichert in der Oracle Datenbank,

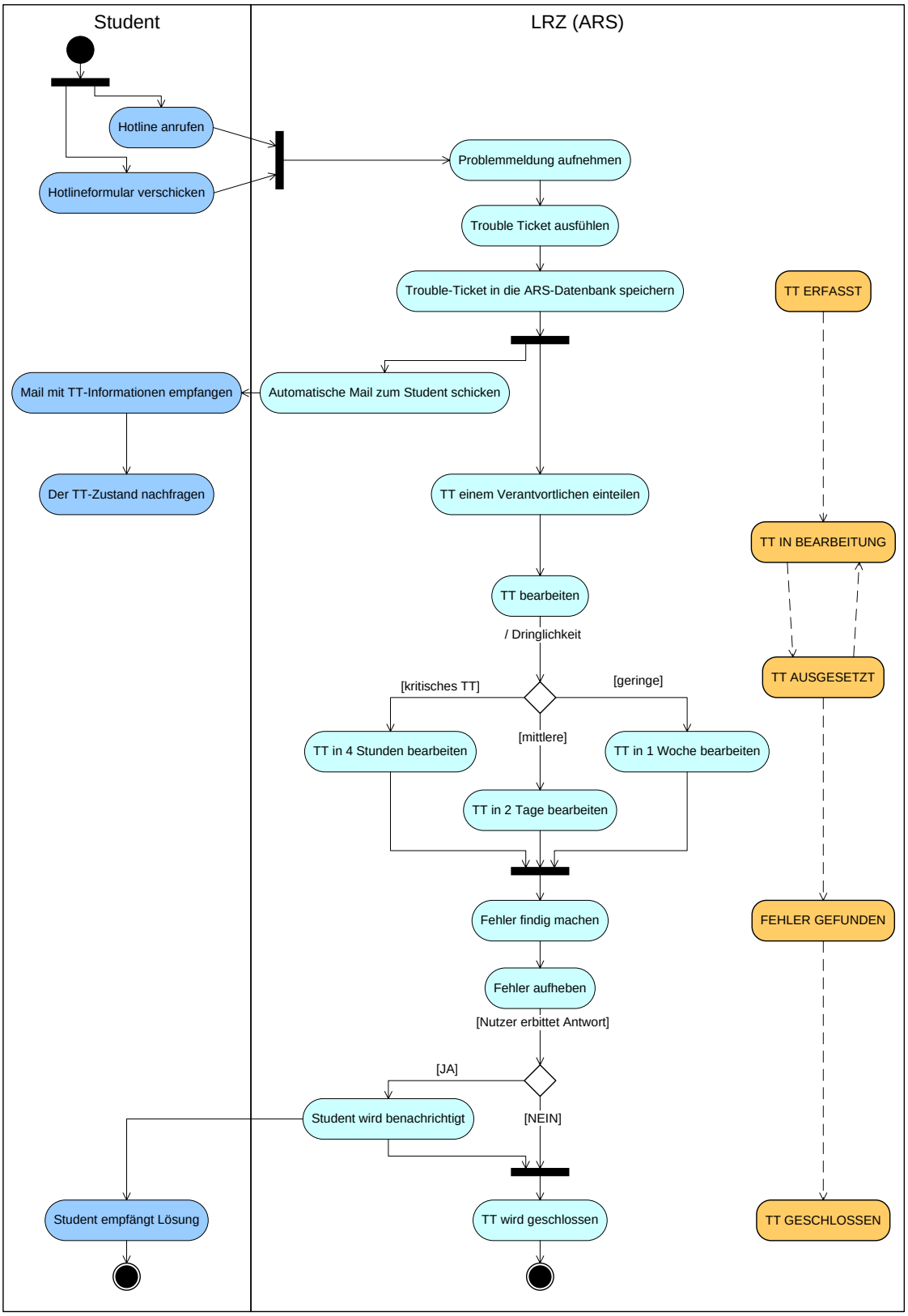

Abbildung 5.24: Aktivitätsdiagramm für die interne Realisierung der Funktionalität *Bearbeitung eines Trouble Tickets*

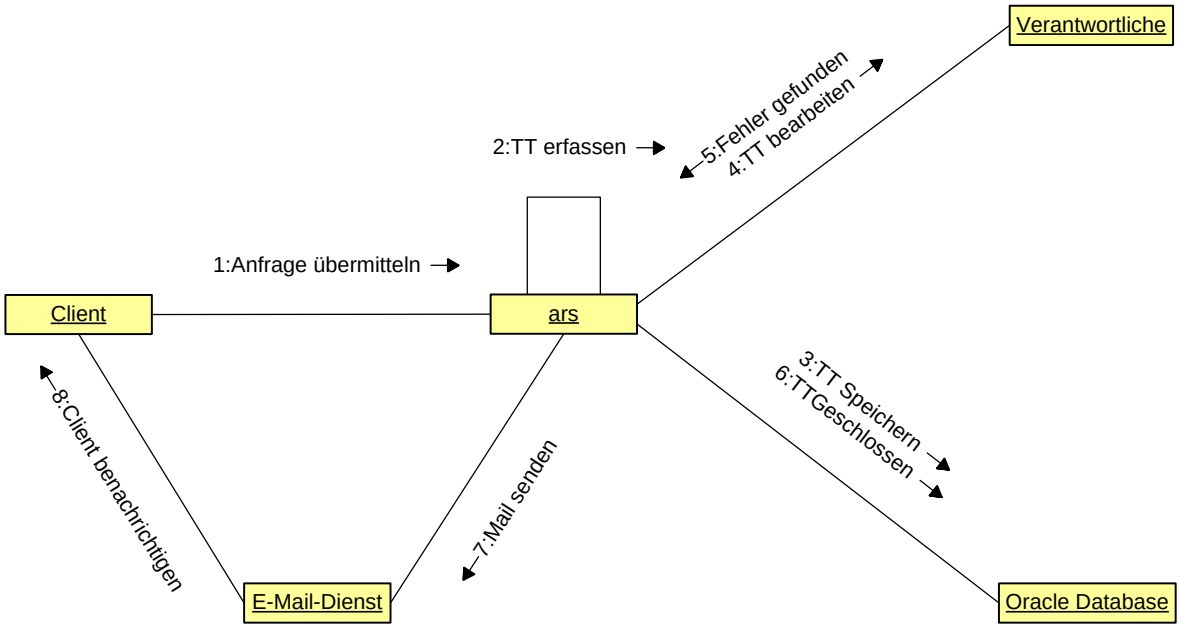

Abbildung 5.25: Kollaborationsdiagramm für die Realisierung der Funktionalität *Bearbeitung eines Trouble Tickets*

eine E-Mail zum Nutzer über die Aufklärung der Störung gesendet wird. Es wird als „Zwischenstation“ der E-Mail Dienst benutzt um die Mail an dem Nutzer weiterleiten zu können (Nachricht 7 und Nachricht 8).

Abhängigkeitsmodellierung nach Kollaborationsdiagramm Man erkennt, dass eine Abhängigkeit zwischen dem Web-Server (seine **ars** Konfiguration) und der Oracle Datenbank existiert. Aus Sicherheitsgründen werden diese zwei, beim LRZ, auf derselben Maschine laufen. Es existiert ebenfalls eine Abhängigkeit zwischen **ars** und der verantwortlichen Person (Bereich der organisatorischen Abhängigkeiten). Beim Versendung von automatisch erstellten E-Mails ist ein gut funktionierender E-Mail-Dienst notwendig.

5.1.3 Anwendung des Abhängigkeitsmodells

In diesem Abschnitt wird die Instantiierung des im Abschnitt 4.2 entworfenen Abhängigkeitsmodells für den Web Hosting Dienst durchgeführt. Es werden hier nicht alle Funktionalitäten betrachtet. Zunächst wird das Modell für die Funktionalität *Zugriff auf geschützten Bereich* (Abschnitt 5.1.3.1), gefolgt von *Zugriff auf Web Seiten* (Abschnitt 5.1.3.2) und *Webmail* Funktionalität im Abschnitt 5.1.3.3 betrachtet. Von den Managementfunktionalitäten werden das *Einrichten eines virtuellen WWW-Servers* im Abschnitt 5.1.3.4 und der *Zugriff auf die Datenbanken über das PhpMyAdmin-Tool* im Abschnitt 5.1.3.5 beschrieben. Als letztes wird das Abhängigkeitsmodell, zusammenfassend auf den Web Hosting Dienst insgesamt (Abschnitt 5.1.3.6) angewendet.

5.1.3.1 Zugriff auf geschützten Bereich

Das Abhängigkeitsmodell wird jetzt auf die tatsächlichen Abhängigkeiten angewendet. Es werden die Klassen aus Abschnitt 4.2.3 instantiiert. Die Abbildung 5.26 zeigt die Anwendung des Abhängigkeitsmodell auf der Funktionalität *Zugriff auf geschützten Bereich*. Wichtige Abhängigkeitsobjekte für die Realisierung dieser Funktionalität sind: ***getSite-accCtrl*** vom Typ *Inter-Service-Dep*, ***accCtrl-virt***, ***accCtrl-DNS*** und ***accCtrl-authService*** alle drei vom Typ *CompositeDependency*.

Das Objekt ***getSite-accCtrl*** repräsentiert die Abhängigkeit der Funktionalität *Zugriff auf Seiten* (getSite) von der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung

- *antecedent*=***accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *dependent*=***getSite*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf Seiten*
- *strength*=***1*** vom Typ Float
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-virt*** stellt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich*(accCtrl) von der Ressource *Web Server* (hier **virt**, nach der Konfiguration was der Webserver in diesem Fall einnimmt) dar. Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung

- *antecedent*=***virt*** vom Typ *Resource* repräsentiert die Ressource *Web Server* eigentlich alle sechs gleichkonfigurierte virt Servern
- *dependent*=***accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength*=***1*** vom Typ Float
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-virt*** hat sechs Kinderelemente ***accCtrl-virt1*** bis ***accCtrl-virt6*** alle vom Typ *Serv-Resource-Dep* (Abhängigkeit zwischen Dienst und Abhängigkeit). Alle diese sechs Abhängigkeiten haben dieselbe Attributbelegung, bis auf den *antecedent* der jeweils die werte jedes den Servern (Notation: virt1,..., virt6). Die Attributen haben für das Objekt ***accCtrl-virt1*** folgende Werte:

- *antecedent*=***virt1*** vom Typ *Resource* repräsentiert der erste der sechs virtuellen Servern mit der Konfiguration *virt*
- *dependent*=***accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength*=***0.5*** vom Typ Float. Es gibt noch keine genaue Bestimmungsmethode für diese Wert. Tatasache ist aber dass dieses Wert nicht 1 sein muss weil der Ausfall einer der sechs Servern allein kann nicht zu einem totallen ausfall des Dienstes führen.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Zu bemerken ist her der Wert des Attributs *strength*. Die Summe der Attributenwerte ist größer wie 1 (der Wert des Attributs *strength* in dem Objekt ***accCtrl-virt***). Das kommt daher, dass die Abhängigkeit zwischen *Zugriff auf geschützten Bereich* und *Web Server* eine sehr wichtige, unabdingbare Beziehung darstellt, aber der Ausfall eines Servers (z.B. virt4) bringt nur eine Umleitung der Arbeiten auf einen der anderen funktionsfähigen Server. Das heißt das seine Wichtigkeit nicht mit 1 bezeichnet werden

kann, weil diesen Wert steht nur deren Abhängigkeiten zu die obligatorisch (verbindlich) sind.

Das Objekt ***accCtrl-DNS*** beschreibt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich*(accCtrl) vom *DNS*-Dienst. Dieses Objekt ist vom Typ *CompositeDependency*, also zusammengesetzt aus den Objekten ***accCtrl-DNSLRZ*** und ***accCtrl-DNSMWN***. Die Werte der Attribute für dieses Objekt sind:

- *antecedent=**DNS*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst*
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Es besteht hier eine zusammengesetzte Abhängigkiet weil die DNS-Service des LRZ redundant verbunden ist mit dem des MWN.

Das Objekt ***accCtrl-DNSLRZ*** repräsentiert die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl) vom *DNS-Dienst* des LRZ (DNSLRZ). Dieses Objekt ist vom Typ *Inter-Service-Dep* (Abhängigkeit zwischen Dienste untereinander) mit den folgenden Attributbelegung:

- *antecedent=**DNSLRZ*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst* des LRZ.
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength=**0.7*** vom Typ Float (beim Ausfall dieses Dienstes wird automatisch auf den DNS-dienst des MWN)
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-DNSMWN*** stellt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl) vom *DNS-Dienst* des MWN (DNSMWN) dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* (Abhängigkeit zwischen Dienste untereinander) mit den folgenden Attributbelegung:

- *antecedent=**DNSMWN*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst* des MWN.
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength=**0.5*** vom Typ Float (Der DNS-Dienst des MWN übernimmt beim Ausfall die Funktionalität des DNS-Dienstes des LRZ)
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-authService*** beschreibt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl) vom *authentisierung*-Dienst. Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent=**authService*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *authentisierung-Dienst* des LRZ.
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength=**1*** vom Typ Float. Diese Abhängigkeit ist von große Wichtigkeit, weil um auf geschützten Bereich zuzugreifen braucht man ein Authentisierung-Dienst.

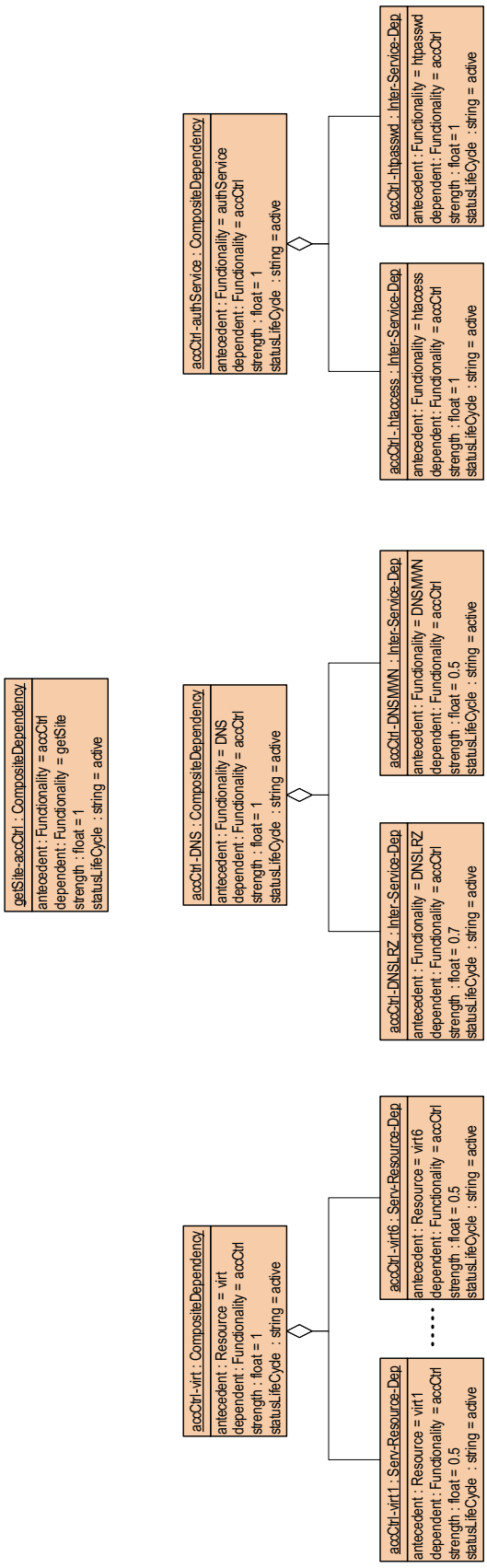

Abbildung 5.26: Abhängigkeitsmodell für die Funktionalität *Zugriff auf geschützten Bereich*

- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-authService*** ist aus ***accCtrl-htaccess*** und ***accCtrl-htpasswd*** zusammengesetzt.

Das Objekt ***accCtrl-htaccess*** stellt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl) vom *authentisierung-Dienst* für den Zugriff auf statischen Seiten dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* (Abhängigkeit zwischen Dienste untereinander) mit den folgenden Attributbelegung:

- *antecedent=**htaccess*** vom Typ *Functionality* (genauer *SingleFunctionality* also Dienst) repräsentiert eine Datei mit der Funktion *authentisierung-Dienst* für den Zugriff auf statischen Seiten.
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***accCtrl-htpasswd*** beschreibt die Abhängigkeit der Funktionalität *Zugriff auf geschützten Bereich* (accCtrl) vom *authentisierung-Dienst* für den Zugriff auf dynamischen Seiten. Das Objekt hat den Typ *Inter-Service-Dep* (Abhängigkeit zwischen Dienste untereinander) mit den folgenden Attributbelegung:

- *antecedent=**htpasswd*** vom Typ *Functionality* (genauer *SingleFunctionality* also Dienst) repräsentiert eine Datei mit der Funktion *authentisierung-Dienst* für den Zugriff auf dynamischen Seiten.
- *dependent=**accCtrl*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf geschützten Bereich*
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

5.1.3.2 Zugriff auf statischen und dynamischen Seiten

Das Abhängigkeitsmodell wird gleichzeitig (denselben Graph) auf den beiden Funktionalitäten *Zugriff auf statische Seiten* und *Zugriff auf dynamischen Seiten* angewendet. Dabei wird das Modell aus 4.2.3 instantiiert. Die Abbildung 5.27 zeigt die Anwendung des Abhängigkeitsmodell auf der Funktionalität *Zugriff auf Webseiten*. Für die Realisierung dieser Funktionalität sind folgende Objekte von Bedeutung: ***wh-getStat*** und ***wh-getDyn***, ***getStat-Storage***, ***getSite-accCtrl***, ***getDyn-Storage***, ***getDyn-AFS*** und ***getDyn-AFS***.

Das Objekt ***wh-getSite*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Zugriff auf Webseiten* (getSite). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung

- *antecedent=**getSite*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert die Funktionalität *Zugriff auf Seiten*
- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Abbildung 5.27: Abhängigkeitsmodell für die Funktionalitäten *Zugriff auf statische Seiten* und *Zugriff auf dynamische Seiten*

Das Objekt ***wh-getStat*** zeichnet die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Zugriff auf statischen Seiten* (getStat) aus. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung

- *antecedent=**getStat*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf statischen Seiten*
- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getSite-accCtrl*** ist im Abschnitt 5.1.3.1 ausführlich beschrieben worden und dadurch wird es hier nicht mehr erklärt.

Das Objekt ***getStat-Storage*** stellt die Abhängigkeit der Funktionalität *Zugriff auf statischen Seiten* (getStat) von der Funktionalität *Speicherung* (Storage) dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent=**Storage*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Speicherung* von statischen Seiten.
- *dependent=**getStat*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf statischen Seiten*
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getStat-Storage*** ist im composition-Beziehung (UML) mit dem Objekt ***getStat-AFS***.

Das Objekt ***getStat-AFS*** stellt die Abhängigkeit der Funktionalität *Zugriff auf statischen Seiten* (getStat) von der Funktionalität *Speicherung in einem File System* (AFS) dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit folgenden Attributen:

- *antecedent=**AFS*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität des Dateisystems AFS bei der Speicherung von Webseiten.
- *dependent=**getStat*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf statischen Seiten*
- *strength=**1*** vom Typ Float. dadurch dass die statische Seiten in AFS gespeichert sind ist diese Abhängigkeit obligatorisch.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getStat-AFS*** ist aus sechs Abhängigkeiten ***virt1-AFS*** bis ***virt6-AFS*** zusammengesetzt.

Wie schon oben erklärt, die Konfiguration der sechs virtuellen Servern **virt1** bis **virt6** ist identisch und dadurch gibt es keine Unterschiede zwischen den sechs Abhängigkeiten. Demnach werden die Abhängigkeit ***virt1-AFS*** bis ***virt6-AFS*** vom Typ *Inter-Resource-Dep* hier nicht mehr beschrieben beschrieben (analog zur Beschreibung von Seite 101).

Das Objekt ***wh-getDyn*** zeichnet die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Zugriff auf dynamischen Seiten* (getDyn) aus. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung

- *antecedent=**getDyn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf dynamischen Seiten*

- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getSite-accCtrl*** ist im Abschnitt 5.1.3.1 ausführlich beschrieben worden und dadurch wird es hier nicht mehr erklärt.

Das Objekt ***getDyn-Storage*** repräsentiert die Abhängigkeit der Funktionalität *Zugriff auf dynamischen Seiten* (getDyn) von der Funktionalität *Speicherung* (Storage). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit folgende Belegung der Attribute:

- *antecedent=**Storage*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Speicherung* von dynamischen Seiten.
- *dependent=**getDyn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf dynamischen Seiten*
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getDyn-AFS*** stellt die Abhängigkeit der Funktionalität *Zugriff auf dynamischen Seiten* (getDyn) von der Funktionalität *Speicherung in einem File System* (AFS) dar. Es geht hier nur um den PHP-Skripten unter das Apache-Modul aus AFS. Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent=**AFS*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität des Dateisystems AFS bei der Speicherung von dynamischen Seiten (PHP-Skripten über Apache-Modul).
- *dependent=**getDyn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf dynamischen Seiten*
- *strength=**1*** vom Typ Float. Dadurch dass der webserver so konfiguriert ist, dass er erst nach einer dynamischen Seite in AFS und erst danach ins NFS sucht, ist diese Abhängigkeit obligatorisch.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***getDyn-AFS*** ist aus sechs Abhängigkeiten ***virt1-AFS*** bis ***virt6-AFS*** zusammengesetzt. Diese wurden ausführlich bei der Funktionalität *Zugriff auf statischen Seiten* beschrieben (siehe Beschreibung auf der Seite 101).

Das Objekt ***getDyn-NFS*** ist auch von Typ *CompositeDependency* und repräsentiert die Abhängigkeit der Funktionalität *Zugriff auf dynamischen Seiten* (getDyn) von der Funktionalität *Speicherung in einem File System* (NFS). Es geht hier um den CGI-Skripten und PHP-skripten die über einen CGI-Wrapper unter NFS gespeichert werden. Die attributen dieses Objekts haben folgende Belegung:

- *antecedent=**NFS*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität des Dateisystems NFS bei der Speicherung von dynamischen Seiten (CGI- und PHP-Skripten über CGI-Wrapper).
- *dependent=**getDyn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf dynamischen Seiten*
- *strength=**1*** vom Typ Float. Diese Abhängigkeit obligatorisch, da die meisten dynamischen Seiten unter NFS gespeichert sind.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

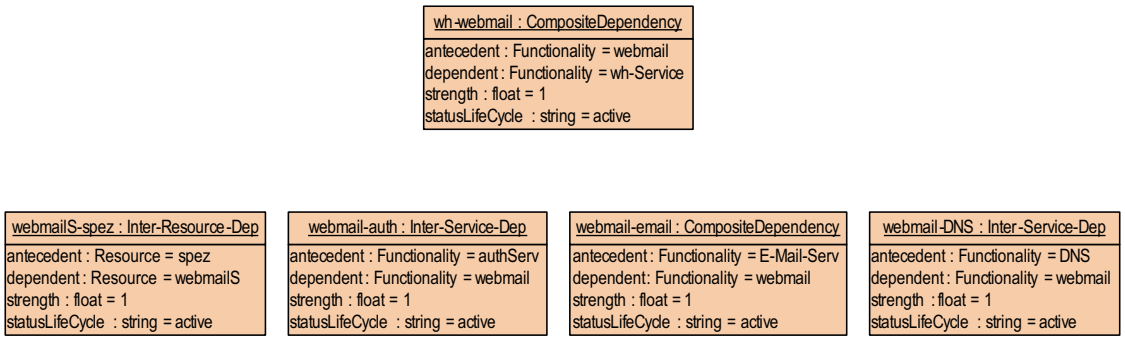

Abbildung 5.28: Abhängigkeitsmodell für die Funktionalität *Webmail*

Diese Abhängigkeit ist aus sechs (Anzahl der Webservern) *Inter-Resource-Dep* Abhängigkeiten zusammengesetzt (***virt1-NFS*** bis ***virt6-NFS***). Diese Objekte sind vom Typ *Inter-Resource-Dep*, aber werden hier nicht mehr beschrieben da die Darstellung ähnlich zu der auf der Seite 101 ist.

5.1.3.3 Webmail Funktionalität

Die Webmail Funktionalität gehört organisatorisch dem Web Hosting Dienst ist aber von dargestellten Funktionalität her, Teil des E-Mail Dienstes. Diese Funktionalität ermöglicht dem Nutzer den Zugriff auf seine Mailbox über einen Web-Browser, also ohne dass ein spezielles Mailprogramm benutzt werden muss.

Die Abbildung 5.28 stellt das Abhängigkeitsmodell für die Funktionalität Webmail dar. Die wichtigsten Abhängigkieten für die Funktionalität *Webmail* sind von den vier Objekten ***webmailS-spez***, ***webmail-auth***, ***webmail-DNS*** und ***webmail-email*** dargestellt.

Das Objekt ***wh-webmail*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Webmail* (webmail). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***webmail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert die Funktionalität *Webmail*
- *dependent*=***wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***webmailS-spez*** repräsentiert die Abhängigkeit des *Webmailservers* (webmailS) vom Webserver in seiner speziellen Konfiguration (spez). Dieses Objekt ist vom Typ *Inter-Resource-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***spez*** vom Typ *Resource* repräsentiert die Konfiguration **spez** des Webservers.
- *dependent*=***webmailS*** vom Typ *Resource* repräsentiert den Webmailserver.
- *strength*=***1*** vom Typ Float. Diese Abhängigkeit muss auf jeden Fall existieren sonst gibt es keine Kommunikation zwischen Funktionalitäten und Dienste auf höheren Ebenen.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***webmail-auth*** zeichnet die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) *authentisierung*-Dienst (authServ) aus. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***authService*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *authentisierung-Dienst* des LRZ.
- *dependent*=***webmail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert die Funktionalität *Webmail*
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt ist einen Blattelement vom Typ *Inter-Service-Dep* aber je nach dem wie komplex der Authentisierungsdienst ist kann er zu einen *CompositeDependency* Objekt erweitert werden.

Das Objekt ***webmail-email*** beschreibt die Abhängigkeit der Funktionalität *Webmail*(webmail) vom *E-Mail*-Dienst (E-Mail-Serv). Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent*=***E-Mail-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *E-Mail-Dienst*
- *dependent*=***webmail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert die Funktionalität *Webmail*
- *strength*=***1*** vom Typ Float. Es wird einen voll funktionsfähigen *E-Mail-Dienst* verlangt für die Funktionalität *Webmail*.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Die ganzen Kinderelemente werden hier nicht mehr ausgeführt weil es bezieht sich auf den E-Mail Dienst der im Abbschnitt 5.2 beschrieben wird.

Das Objekt ***webmail-DNS*** stellt die Abhängigkeit der Funktionalität *Webmail*(webmail) vom *DNS*-Dienst dar. Dieses Objekt ist vom Typ *Inter-Service-Dep*. Die Werte der Attribute für dieses Objekt sind:

- *antecedent*=***DNS*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst*
- *dependent*=***webmail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert die Funktionalität *Webmail*.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt kann auch von Typ *CompositeDependency* sein wenn der DNS-Dienst von mehreren redundanten DNS-Servern realisiert wird.

5.1.3.4 Einrichten eines virtuellen WWW-Servers

In diesem Abschnitt wird das Abhängigkeitsmodell für die interne Realisierung der Funktionalität Einrichten eines virtuellen WWW-Servers beschrieben. Die grafische Darstellung dieses Modells befindet sich in der Abbildung 5.29. Von Bedeutung für die Realisierung dieser Funktionalität sind folgende Objekte: ***wh-wwwSetUp***, ***homepageTool-lrz***, ***wwwSetUp-DNS*** und ***wwwSetUp-AFS***

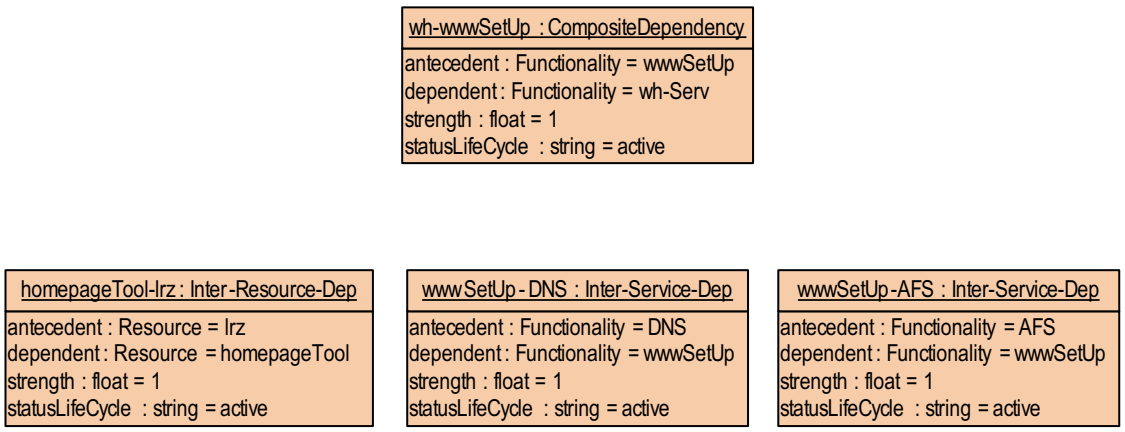

Abbildung 5.29: Abhängigkeitsmodell für die Funktionalität *Einrichten eines virtuellen WWW-Servers*

Das Objekt ***wh-wwwSetUp*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Einrichten eines virtuellen WWW-Servers* (wwwSetUp). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***wwwSetUp*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Einrichten eines virtuellen WWW-Servers*
- *dependent*=***wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***homepageTool-lrz*** repräsentiert die Abhängigkeit des *Homepage Tools* (homepageTool) vom Webserver in seiner speziellen Konfiguration (lrz). Dieses Objekt ist vom Typ *Inter-Resource-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***lrz*** vom Typ *Resource* repräsentiert die Konfiguration **lrz** des Webservers.
- *dependent*=***homepageTool*** vom Typ *Resource* repräsentiert den Werkzeug mit dem eine Webseite neu erstellt wird.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***wwwSetUp-DNS*** stellt die Abhängigkeit der Funktionalität *Einrichten eines virtuellen WWW-Servers*(wwwSetUp) vom *DNS*-Dienst dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***DNS*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst*
- *dependent*=***wwwSetUp*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Einrichten eines virtuellen WWW-Servers*.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt wurde hier als *Inter-Service-Dep* Objekt betrachtet kann aber zu einem *CompositeDependency* Objekt erweitert werden indem auch alle Ressourcen und alle Redundante DNS-Dienste

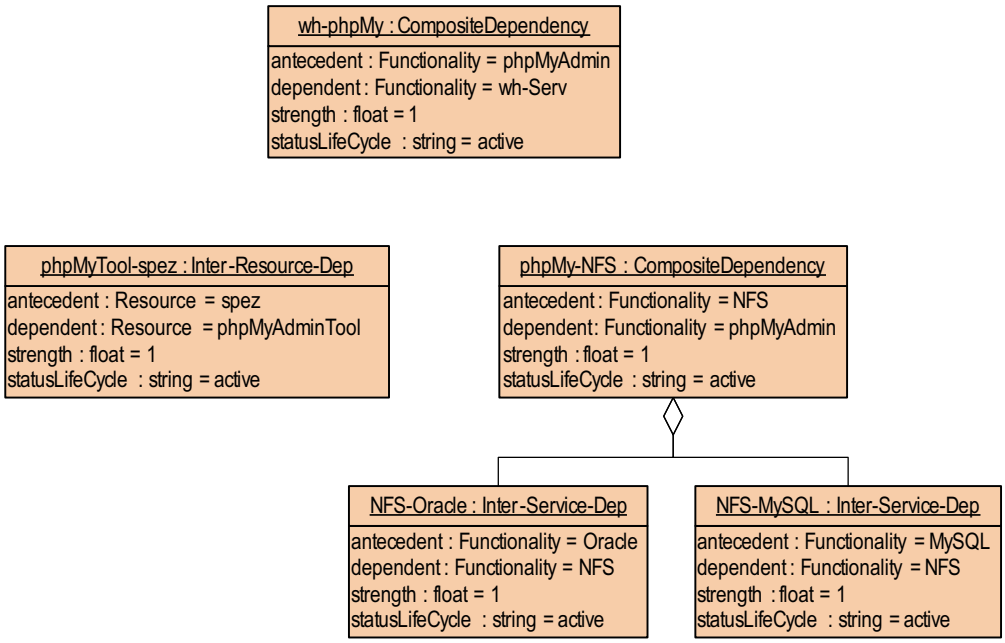

Abbildung 5.30: Abhängigkeitsmodell für die Funktionalität *Zugriff auf die Datenbanken über das PhpMyAdmin-Tool*

noch abgebildet werden können (so wie bei den Funktionalitäten *Zugriff auf geschützten Bereich* und *Zugriff auf Webseiten*).

Das Objekt ***wwwSetUp-AFS*** beschreibt die Abhängigkeit der Funktionalität *Einrichten eines virtuellen WWW-Servers*(wwwSetUp) der Funktionalität *Speicherung in einem File System* (AFS). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent=**AFS*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität des Dateisystems AFS bei der Speicherung von Webseiten (hier die provisorische Webseite).
- *dependent=**wwwSetUp*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Einrichten eines virtuellen WWW-Servers*.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt vom Typ *Inter-Service-Dep* kann ebenfalls zu einem vom Typ *CompositeDependency* erweitert werden durch Einfügen von Kindelementen.

5.1.3.5 Zugriff auf die Datenbanken über das PhpMyAdmin-Tool

Die Abhängigkeiten die aus dem Kollaborationsdiagramm abgelesen worden sind werden jetzt dem Abhängigkeitsmodell zusammengefasst. Die Abbildung 5.30 stellt das Abhängigkeitsmodell für die Funktionalität Zugriff auf den Datenbanken über das PhpMyAdmin-Tool dar. Die beschribene Objekte sind: ***wh-phpMy***, ***phpMy-spez*** und ***phpMy-NFS***.

Das Objekt ***wh-phpMy*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der

Funktionalität *Zugriff auf den Datenbanken über das PhpMyAdmin-Tool* (phpMyAdmin). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung

- *antecedent*=***phpMyAdmin*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf den Datenbanken über das PhpMyAdmin-Tool*
- *dependent*=***wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***phpMy-spez*** repräsentiert die Abhängigkeit des *PhpMyAdmin*-Werkzeug (phpMyAdmin-Tool) vom Webserver in seiner speziellen Konfiguration (spez). Dieses Objekt ist vom Typ *Inter-Resource-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***spez*** vom Typ *Resource* repräsentiert die Konfiguration **spez** des Webservers.
- *dependent*=***phpMyAdminTool*** vom Typ *Resource* repräsentiert das *PhpMyAdmin*-Werkzeug.
- *strength*=***1*** vom Typ Float. Diese Abhängigkeit muss auf jeden Fall existieren sonst gibt es keine Kommunikation zwischen Funktionalitäten und Dienste auf höheren Ebenen.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***phpMy-NFS*** beschreibt die Abhängigkeit der Funktionalität *Zugriff auf den Datenbanken über das PhpMyAdmin-Tool* (phpMyAdmin) von der Funktionalität *Speicherung in einem File System* (NFS). Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent*=***NFS*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität des Dateisystems NFS bei der Speicherung von Webseiten aber auch den Zugang zu den Datenbanken.
- *dependent*=***phpMyAdmin*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Zugriff auf den Datenbanken über das PhpMyAdmin-Tool*.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***phpMy-NFS*** ist ein zusammengesetztes Objekt. Die Kinderelemente sind: ***NFS-Oracle*** und ***NFS-MySQL***.

Das Objekt ***NFS-Oracle*** beschreibt die Abhängigkeit der Funktionalität *File System* (NFS) von der Funktionalität *Speicherung* in einer Oracle Datenbank (Oracle). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***Oracle*** vom Typ *Functionality* repräsentiert die Funktionalität *Speicherung* in einer Oracle Datenbank.
- *dependent*=***NFS*** vom Typ *Functionality* repräsentiert die Funktionalität des Dateisystems NFS bei der Speicherung von Webseiten aber auch den Zugang zu den Datenbanken.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***NFS-MySQL*** beschreibt die Abhängigkeit der Funktionalität *File System* (NFS) von der Funktionalität *Speicherung* in einer MySQL Datenbank (MySQL). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***MySQL*** vom Typ *Functionality* repräsentiert die Funktionalität *Speicherung* in einer MySQL Datenbank.

- *dependent=**NFS*** vom Typ *Functionality* repräsentiert die Funktionalität des Dateisystems NFS bei der Speicherung von Webseiten aber auch den Zugang zu den Datenbanken.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Die zwei letzten Objekte können abhängig von der Granularität der Abhängigkeiten weiter verfeinert werden und dadurch werden sie *CompoditeDependency* Objekte.

5.1.3.6 Anwendung des Abhängigkeitsmodells auf dem Web Hosting Dienst insgesamt

In der Abbildung 5.31 wurden alle Funktionalitäten des Web Hosting Dienstes zusammengefasst. Der Web Hosting Dienst ist von seinen allen Funktionalitäten abhängig.

Das Objekt ***wh-whFunc*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von seinen allen Funktionalitäten (wh-Functionality). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent=**wh-Functionality*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert alle Funktionalitäten des Web Hosting Dienstes.
- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***wh-usageFunc*** stellt die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von den Nutzungsfunktionalitäten (usageFunctionality) dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent=**usageFunctionality*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert alle Nutzungsfunktionalitäten des Web Hosting Dienstes.
- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Die anderen Objekte, die für den Web Hosting-Dienst wichtig sind, wurden in den vorherigen Abschnitten schon definiert und ausführlich beschrieben (***wh-getSite*** in Abschnitt 5.1.3.2, ***wh-accCtrl*** im Abschnitt 5.1.3.1 und ***wh-webmail*** im Abschnitt 5.1.3.3).

Das Objekt ***wh-mgmtFunc*** stellt die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von den Managementfunktionalitäten (mgmtFunctionality) dar. Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent=**usageFunctionality*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert alle Managementfunktionalitäten des Web Hosting Dienstes.
- *dependent=**wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Abbildung 5.31: Abhängigkeitsmodell für den Web Hosting Dienst

Diese vier Objekte ***wh-wwwSetUp***, ***wh-wChange***, ***wh-ars*** und ***wh-phpMy*** sind für die Realisierung der Managementfunktionalitäten von Bedeutung. Die Objekte ***wh-wwwSetUp*** und ***wh-phpMy*** wurden in den Abschnitten 5.1.3.4 bzw. Abschnitt 5.1.3.5 detailliert beschrieben.

Das Objekt ***wh-wChange*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Einrichten/Ändern von Webseiten* (wwwChange). Dieses Objekt ist vom Typ *Inter-Service-Dep* mit den folgenden Attributbelegung:

- *antecedent*=***wwwChange*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Einrichten/Ändern von Webseiten*
- *dependent*=***wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt wurde aus Platzgründe hier nicht mehr dargestellt.

Das Objekt ***wh-ars*** repräsentiert die Abhängigkeit des *Web Hosting*-Dienstes (wh-Serv) von der Funktionalität *Bearbeitung eines Trouble Tickets* (ars). Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent*=***ars*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Bearbeitung eines Trouble Tickets*
- *dependent*=***wh-Serv*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Web Hostig Dienst insgesamt.
- *strength*=***1*** vom Typ Float.
- *statusLifeCycle*=***active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Eine Erweiterung dieses Objekts wurde, aus Platzgründen, in dieser Arbeit nicht realisiert.

5.2 Modellierung des E-Mail-Dienstes

In diesen Kapitel wird die Anwendung des Abhängigkeitsmodell auf das E-Mail Szenario durchgeführt. Aus Platzgründen wird die MNM-Dienstmethodik, bis auf das Erstellen von Kollaborationsdiagrammen, nicht mehr angewendet. Ebenfalls bleiben auch die Managementaspekte außer Betracht. Das Abhängigkeitsmodell wird auf die Nutzungsfunktionalitäten angewendet.

Wie in [Bern 04] beschrieben, wird die Nutzungsfunktionalität des E-Mail-Dienstes am LRZ in drei Kategorien eingeteilt, die sich durch das eingesetzte Anwendungsprotokoll unterscheiden: Mailsenden (SMTP-Protokoll), Mailempfangen (SMTP- bzw. POP/IMAP-Protokoll) und Webmail (HTTP-Protokoll). Die Webmail Funktionalität wurde schon im Abschnitt 5.1 betrachtet, da sie, beim LRZ, organisatorisch dem Web Hosting Dienst zugeordnet ist. Die anderen Funktionalitäten werden in den folgenden zwei Abschnitten detailliert dargestellt.

5.2.1 Mail senden

Diese Funktionalität bezieht sich auf das Senden von Mails vom LRZ. Es wird eine SMTP-Verbindung zum Mailrelay aufgebaut, Daten werden übertragen. Wichtig zu beachten ist hier: die Verfügbarkeit des SMTP-Dienstes (Subdienst), Verfügbarkeit des Mailrelays, Verfügbarkeit der Verbindung.

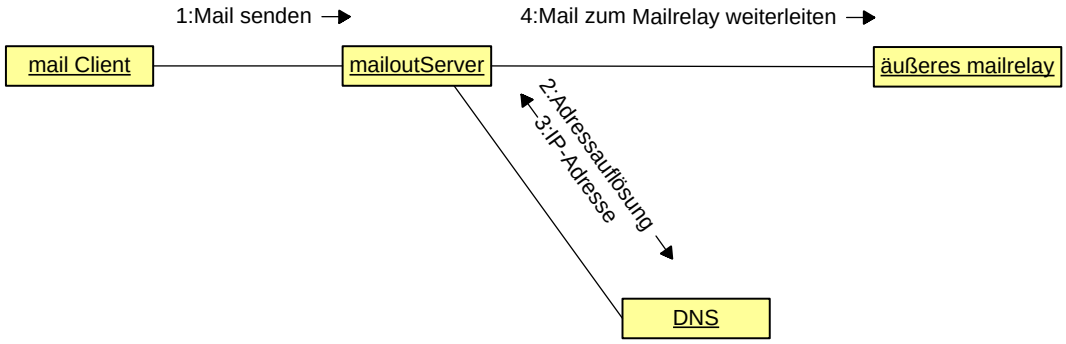

Abbildung 5.32: Kollaborationsdiagramm für die Realisierung der Funktionalität *Mail senden*

Abbildung 5.32 stellt das Kollaborationsdiagramm für die Realisierung der Funktionalität *Mail senden* dar. Von einem E-Mail-Client wird eine Mail versendet. Erst wird eine Adressauflösung in DNS realisiert und hiermit wird die IP-Adresse des Empfangmailservers (bzw. Mailrelay) ermittelt. Dann muss eine TCP/IP-Verbindung zu diesem aufgebaut werden.

In der Abbildung 5.33 wird die Anwendung des im Abschnitt 4.2 dargestellten Abhängigkeitsmodell durchgeführt. Die folgenden Objekte sind für die Realisierung der Funktionalität *Mail senden* von Bedeutung: ***EMail-MailOut***, ***MailOut-mailoutS***, ***MailOut-DNS***, ***MailOut-SMTP*** und ***MailOut-IP***.

Das oberste Objekt in der Hierarchie, ***EMail-MailOut***, ist vom Typ *Inter-Service-Dep* und stellt die Abhängigkeit des *E-Mail*-Dienstes (EMail) von der Funktionalität *Mail senden* (MailOut) dar. Die Attributbelegung für dieser Klasse ist:

- *antecedent=**MailOut*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail senden*.

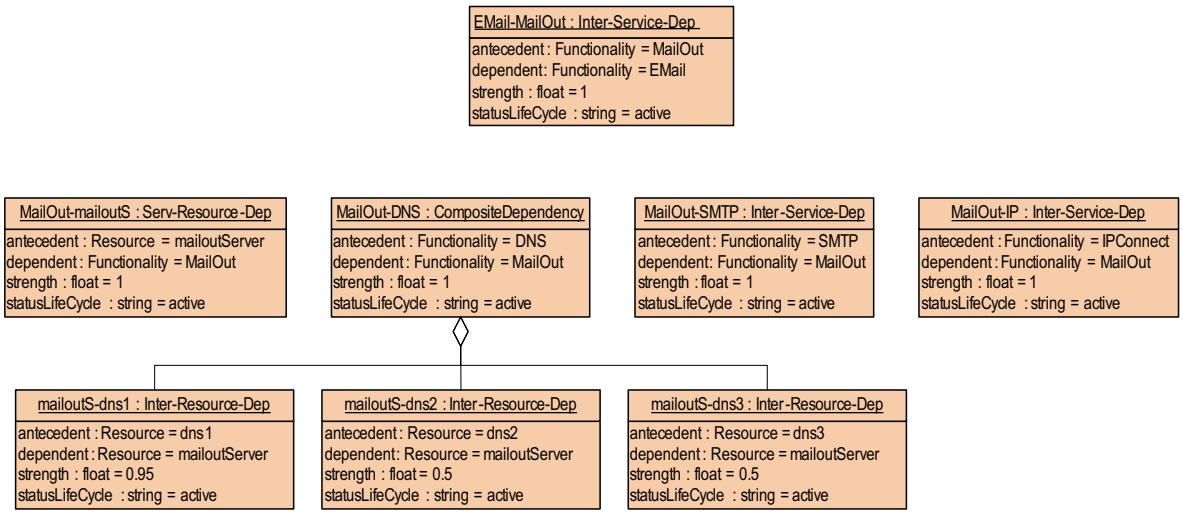

Abbildung 5.33: Abhängigkeitsmodell für die Funktionalitäten *Mail senden*

- *dependent=**EMail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den E-Mail-Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailOut-mailoutS*** stellt die Abhängigkeit der Funktionalität *Mail senden*(MailOut) von der Ressource *mailout-Server* (mailoutS) dar. Dieses Objekt ist vom Typ *Serv-Resource-Dep* mit den folgenden Attributbelegung

- *antecedent=**mailoutS*** vom Typ *Resource* repräsentiert die Ressource *mailout-Server*.
- *dependent=**MailOut*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail senden*
- *strength=**1*** vom Typ Float
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailOut-DNS*** beschreibt die Abhängigkeit der Funktionalität *Mail senden*(MailOut) vom *DNS*-Dienst. Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung:

- *antecedent=**DNS*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *DNS-Dienst*
- *dependent=**MailOut*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail senden*.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailOut-DNS*** ist aus drei Abhängigkeiten ***mailoutS-dns1***, ***mailoutS-dns2*** und ***mailoutS-dns3*** zusammengesetzt. Die Servern *dns1*, *dns2*, und *dns3* sind gleich konfiguriert allerdings ist *dns1* der primäre DNS-Server und die anderen die sekundären.

Da die Konfiguration der drei DNS-Servern identisch ist, und dadurch auch fast keine Unterschiede zwischen den drei Abhängigkeiten, wird nur die Abhängigkeit ***mailoutS-dns1*** vom Typ *Inter-Resource-Dep* beschrieben. Die Attributbelegung dieses Objekts ist:

- *antecedent=**dns1*** vom Typ *Resource* repräsentiert der primäre DNS-Server.
- *dependent=**mailoutS*** vom Typ *Resource* ist der mailout-Server.
- *strength=**0.95*** vom Typ Float. Weil dieser der primäre DNS-Server ist, bekommt die Abhängigkeit eine höherer Wichtigkeit. Die anderen Abhängigkeiten des Verbundes haben nur noch den Wert 0.5.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailOut-SMTP*** beschreibt die Abhängigkeit der Funktionalität *Mail senden*(MailOut) vom *SMTP*-Dienst. Das Objekt ist vom Typ *Inter-Service-Dep* und hat die folgenden Attributbelegung:

- *antecedent=**SMTP*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *SMTP-Dienst*
- *dependent=**MailOut*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail senden*.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailOut-IP*** stellt die Abhängigkeit der Funktionalität *Mail senden*(MailOut) vom die IP-Connectivity (IPConnect) dar. Das Objekt ist vom Typ *Inter-Service-Dep* und hat die folgenden Attributbelegung:

- *antecedent=**IPConnect*** vom Typ *Functionality* (genauer *CompositeFunctionality* also Dienst) repräsentiert den *IP-Connectivity-Dienst*
- *dependent=**MailOut*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail senden*.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Die oberen zwei Objekte können gegebenfalls noch erweitert werden zum *CompositeDependency* Objekt.

5.2.2 Mail empfangen

Das Empfangen von Mails wird im Zusammenhang betrachtet mit: der Erreichbarkeit der Mailbox und dem Abrufen von E-Mails. Das Empfang von E-Mails wird, wie in der Abbildung 5.34 dargestellt, realisiert.

Ein Mailserver von „Außen“ sendet eine E-Mail an der *mailrelay* der LRZ. Als erstes wird eine Anfrage am Resolver (mit der IP-Adresse) zur Namensauflösung gestellt. Der Domänenname wird daraufhin zurückgegeben und hier können sich die Aktionen verzweigen. Auf einer Seite kann die E-Mail weitergeleitet werden (nachricht 4.1) zu eine anderen Mailrelay wenn sich die Empfängeradresse nicht in den Domäne des LRZ befindet. Auf der anderen Seite, wen der Empfänger bekannt ist, wird die E-Mail in dem ensprechenden Mailbox abgelegt (Nachricht 4.2).

Abbildung 5.35 stellt die Anwendung des Abhängigkeitsmodells auf der interne Realisierung der Funktionalität *Mail empfangen* dar. Wichtige Objekte für die Realisierung der *Mail empfangen* Funktionalität sind: ***EMail-MailIn***, ***MailIn-mailrelay*** und ***MailIn-Resolver***. Das Objekt ***EMail-MailIn*** ist

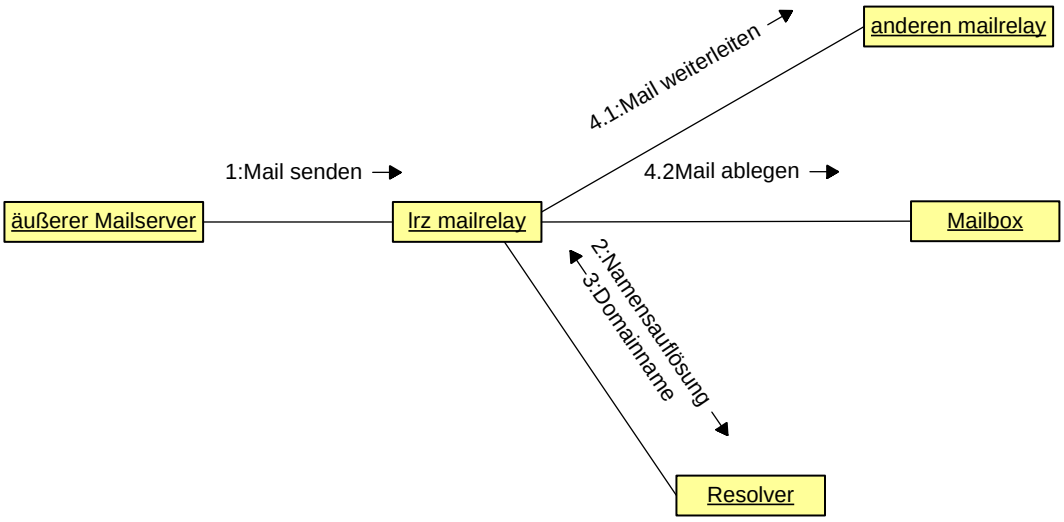

Abbildung 5.34: Kollaborationsdiagramm für die Realisierung der Funktionalität *Mail empfangen*

vom Typ *Inter-Service-Dep* und stellt die Abhängigkeit des *E-Mail*-Dienstes (EMail) von der Funktionalität *Mail empfangen* (MailIn) dar. Die Attributbelegung für dieser Klasse ist:

- *antecedent=**MailIn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail empfangen*.
- *dependent=**EMail*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den E-Mail-Dienst insgesamt.
- *strength=**1*** vom Typ Float.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailIn-mailrelay*** stellt die Abhängigkeit der Funktionalität *Mail senden*(MailIn) von der Ressource *Mailrelay* des LRZ (Mailrelay) dar. Dieses Objekt ist vom Typ *CompositeDependency* mit den folgenden Attributbelegung

- *antecedent=**Mailrelay*** vom Typ *Resource* repräsentiert die Ressource *Mailrelay* des LRZ.
- *dependent=**MailIn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail empfangen*
- *strength=**1*** vom Typ Float
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dadurch das beim LRZ sich zwei Mailrelays befinden (mailrelay1 und mailrelay2) handelt es sich hier um einer Verbundabhängigkeit. Die zwei Mailrelays sind identisch konfiguriert, un dadurch sind die zwei Objekte ***MailIn-mailrelay1*** und ***MailIn-mailrelay2*** mit identischen Attributbelegungen erstellt.

Das Objekt ***MailIn-mailrelay1*** stellt die Abhängigkeit der Funktionalität *Mail senden*(MailIn) von der Ressource *mailrelay1* des LRZ (mailrelay1) dar. Dieses Objekt ist vom Typ *Serv-Resource-Dep* mit den folgenden Attributbelegung:

- *antecedent=**mailrelay1*** vom Typ *Resource* repräsentiert die Ressource *mailrelay1* des LRZ.
- *dependent=**MailIn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail empfangen*
- *strength=**0.8*** vom Typ Float. Dadurch dass der Ausfall eines des Mailrelays nicht zu einem

Abbildung 5.35: Abhängigkeitsmodell für die Funktionalitäten *Mail empfangen*

totalen Ausfalls des MailIn Funktionalität führt sondern die Last verteilt sich auf dem anderen Mailrelay, wird der Wert 0.8 zugewiesen.
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Das Objekt ***MailIn-Resolver*** ist ein *CompositeDependency* objekt das die Abhängigkeit der Funktionalität *Mail senden*(MailIn) von dem *Resolver*-Dienst des LRZ (Resolver) dar. Die Attributbelegung für dieses Objekt ist:

- *antecedent=**Resolver*** vom Typ *Functionality* (genauer *CompositeFunctionality*) repräsentiert den Resolver-Dienst des LRZ.
- *dependent=**MailIn*** vom Typ *Functionality* (genauer *SingleFunctionality*) repräsentiert die Funktionalität *Mail empfangen*
- *strength=**1*** vom Typ Float
- *statusLifeCycle=**active*** vom Typ string weil der Dienst sich in Usage-Phase befindet.

Dieses Objekt ist aus zwei gleichartige Abhängigkeiten ***MailIn-res1*** und ***MailIn-res2*** zusammengesetzt. Das kommt davon dass beim LRZ zwei Resolver-Servern existieren. Die Objekte ***MailIn-res1*** und ***MailIn-res2*** können als *Serv-Resource-Dep* betrachtet Sie können aber auch als *CompositeDependency* Objekte betrachtet werden men man die zwei mailrelays wieder miteinbezieht. In diesem Fall würde das Objekt ***MailIn-res1*** aus ***mailrelay1-res1*** und ***mailrelay2-res1*** und der ***MailIn-res2*** aus ***mailrelay1-res2*** und ***mailrelay2-res2***, alle vom typ *Inter-Resource-Dep* zusammengesetzt.

5.3 Zusammenfassung

In diesem Kapitel wurde die prototypische Anwendung der entwickelten Abhängigkeitsmodellierung durchgeführt. Im ersten Teil wurde für den Web Hosting-Dienst erst die MNM-Dienstmethodik(in den Abschnitten 5.1.1 und 5.1.2) angewendet, um die Abhängigketen, die zur Realisierung dieses Dienstes beitragen, zu bestimmen. Danach wurde das Abhängigkeitsmodell (Abschnitt 5.1.3) erstellt. In dem letzten Teil (Abschnitt 5.2) wurde für den E-Mail-Dienst das Abhängigkeitsmodell für die Nutztungsfunktionalitäten angewendet. Wie es ersichtlich ist, ist die Anwendung des Abhängigkeitsmodells ziemlich aufwendig und daher die Erstellung von Templates wünschenswert. Diese Templates wurden es erlauben, für unterschiedlichen Klassen von Anwendungen würden diese Templates es erlauben, häufig auftretende Abhängigkeiten zu beschreiben. Sie könnten in einem nächsten Schritt szenariospezifisch verfeinert werden (siehe Kapitel 6).

Kapitel 6

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein systematischer, strukturierter Ansatz zur Modellierung von Abhängigkeitsbeziehungen entwickelt. Grundlage für den Ansatz stellt das MNM-Dienstmodell dar, das im Laufe der Arbeit grundlegend verfeinert worden ist.

Zunächst wurden, als Referenz für die Beschreibung der Anforderungen, die zwei Szenarien Web Hosting- und E-Mail-Dienst des LRZ betrachtet. In Bezug auf die beschriebenen Szenarien wurde als nächstes die Anforderungsanalyse realisiert. Die Anforderungen sind in eine dienstbezogene und eine abhängigkeitsbezogene Kategorie eingeteilt. Im Laufe des Nachforschens für diese Arbeit wurde weiterhin zwischen dem Bestimmen und dem Modellieren von Abhängigkeiten differenziert. Diese Art der Einteilung erlaubt eine systematische Betrachtung von Abhängigkeiten, die bisher in der akademischen Literatur fehlte.

Die dienstbezogenen Anforderungen sind für die allgemeine Einhaltung des Dienstkonzepts, die Technologieunabhängigkeit sowie die Erweiterbarkeit und Anwendbarkeit gedacht. Zusätzlich wurden noch diejenigen Aspekte des Dienstes betrachtet die mit der Erbringung des Dienstes zusammenhängen; wie z.B. der Lebenszyklus, die Einteilung eines Dienstes in Dienstfunktionalitäten sowie die Parameter, die zur Realisierung des Dienstes beitragen. Die wichtigsten dieser Parameter sind die QoS-Parameter, die immer eingehalten werden müssen und deren Verletzung ein Problem für den Provider darstellen. Die dienstbezogenen Abhängigkeiten sind die grundlegenden Anforderungen, auf denen dann die abhängigkeitsbezogenen Anforderungen aufbauen. Die abhängigkeitsbezogenen Anforderungen beziehen sich auf die Arten von Abhängigkeiten, die in Betracht gezogen werden können, auf deren Dynamik und auf den Redundanzen unterschiedlicher Ressourcen bzw. Diensten. Es wurde im Abschnitt 2.3.2 zwischen folgenden drei unterschiedlichen Arten von Abhängigkeiten unterschieden: Abhängigkeiten zwischen Ressourcen, Abhängigkeiten zwischen Ressourcen und Diensten und Abhängigkeiten zwischen Diensten. Zusammenfassend wurde sowohl ein Dimensionenstern der Abhängigkeiten als auch ein Anforderungskatalog erstellt.

Die vorhandenen Ansätze aus der Forschungswelt und der Standardisierung sowie die kommerziellen Produkte wurden als Grundlage für die Weitererforschung der Abhängigkeiten benutzt. Jeder der Ansätze ist anhand der Anforderungen analysiert und je nachdem welche Vorteile er erbracht hat, weiterbenutzt worden. Das MNM-Dienstmodell wurde als Grundlage zur Erweiterung gewählt weil es das einzige umfassende Dienstmodell ist und weil es sehr gut auf die dienstbezogenen Anforderungen anspricht.

Für die Abhängigkeitsmodellierung wurde als erstes die Bestimmung der Abhängigkeiten betrachtet. Die Instantiierungsmethodik des MNM-Dienstmodells, die als Werkzeug zur Bestimmung der Abhängigkeiten benutzt wurde, ist in Abschnitt 4.1 ausführlich erklärt. Die Instantiierung des MNM-Dienstmodells für das Web Hosting-Dienst Szenario ist in sich einen Beitrag für die Anwendbarkeit

des MNM-Dienstmodells, da keine andere so detaillierte Anwendung (bis zum Kollaborationsdiagramm) bisweilen existiert. Nach jedem Schritt der Instantiierungsmethodik wurde eine Bewertung der Ausdrucksmöglichkeit in Bezug auf Abhängigkeiten realisiert. Nachdem die Abhängigkeiten bestimmt wurden entsteht die Notwendigkeit sie zu organisieren, um Einfachheit und Übersichtlichkeit zu gewährleisten. Ebenfalls muss ein Management-Tool die Möglichkeit haben sowohl einfache als auch komplexe Abhängigkeiten gleich zu betrachten.

Zur Erfüllung der genannten Anforderungen wurde das Composite Design Pattern aus dem Bereich des Softwaredesigns gewählt. Dadurch ist das Composite Pattern *Dependency* entstanden. Es betrachtet auf ähnliche Art und Weise sowohl einfache Abhängigkeiten als auch komplexe Kombinationen davon. Die einfachen Abhängigkeiten werden im Modell als Blattelemente (***Inter-Resource-Dep***, ***Service-Resource-Dep***, ***Inter-Service-Dep***) und die komplexen Elemente als ***CompositeDepedency*** (zusammengesetzte Elemente) betrachtet. Durch die Anwendung des Composite Design Pattern wurde eine strukturierte Darstellung geschaffen, die Einfachheit und Übersichtlichkeit einbringt.

Die Attribute die diese Klassen beschreiben sind sehr wichtig, weil sie den Eigenschaften der Abhängigkeiten entsprechen. Die Methoden sind in zwei Kategorien eingeteilt: die kindbezogenen Methoden und die Methoden für das Fehlermanagement. In der ersten Kategorie sind die Methoden für die Speicherung und Verwaltung der Kindelemente verantwortlich. Die Methoden der zweiten Kategorie sind für das IT Service Management von Bedeutung, weil durch deren Anwendung die Fehlerursache gefunden (*root cause analysis*) oder die Auswirkungen eines Ausfalls vorausgesehen werden kann (*impact-analysis*). Die Anwendung dieser Methoden ist allerdings umfangreicher, so dass deren Optimierung dieser Methoden als Anregung für weitere aufbauende Arbeiten gesehen werden kann.

Die Anwendung der MNM-Dienstmethodik und des Abhängigkeitsmodells auf die Szenarien wurde anschließend realisiert. Es wurden die zwei Szenarien: Web Hosting- und E-Mail-Dienst separat betrachtet. Für den Web Hosting-Dienst wurde sehr ausführlich sowohl das MNM-Dienstmodell (mit Use Case, Aktivitäts- und Kollaborationsdiagrammen) als auch das Abhängigkeitsmodell angewendet. Damit wurde die Praxistauglichkeit der Abhängigkeitsmodellierung unter Beweis gestellt.

Weiterführende Arbeiten könnten im Bereich der Top Down- bzw. Bottom Up-Analyse realisiert werden, indem man eine Methodik zum Besetzen der Attribute entwickelt. Hierbei geht es hauptsächlich um das Attribut *strength*, wobei sich u.a. folgende Fragen stellen: Welchen Wert muss es haben, um sich den gegebenen Situationen anpassen zu können? Wie belegt man diesen Wert, um die Effizienz des Systems zu gewährleisten? Auch die Methode *getImpact()* könnte ausführlich konzipiert werden, um die tatsächlichen Auswirkungen berechnen zu können.

Diese Arbeit befasste sich ausschließlich mit den funktionalen Abhängigkeiten. In großen IT-Unternehmen, in denen unterschiedliche Dienste von unterschiedlichen Abteilungen realisiert werden, sind die organisatorischen Abhängigkeiten ein sehr wichtigen Faktor, der betrachtet werden sollte. Eine Erweiterung des Modells, auch auf organisatorische Abhängigkeiten, wäre daher wünschenswert.

Eine weitere Anregung für andere Arbeiten wäre die Erweiterung der bestehenden Informationsmodelle durch das hier entwickelte Abhängigkeitsmodell. Wie in Kapitel 5 schon gezeigt ist die Anwendung des Abhängigkeitsmodells ziemlich aufwendig und daher wäre die Erstellung von Templates wünschenswert. Für unterschiedliche Klassen von Anwendungen würden diese Templates es erlauben, häufig auftretende Abhängigkeiten zu beschreiben. Sie könnten in einem nächsten Schritt szenariospezifisch verfeinert werden.

Anhang A

CIM-Schemata in UML

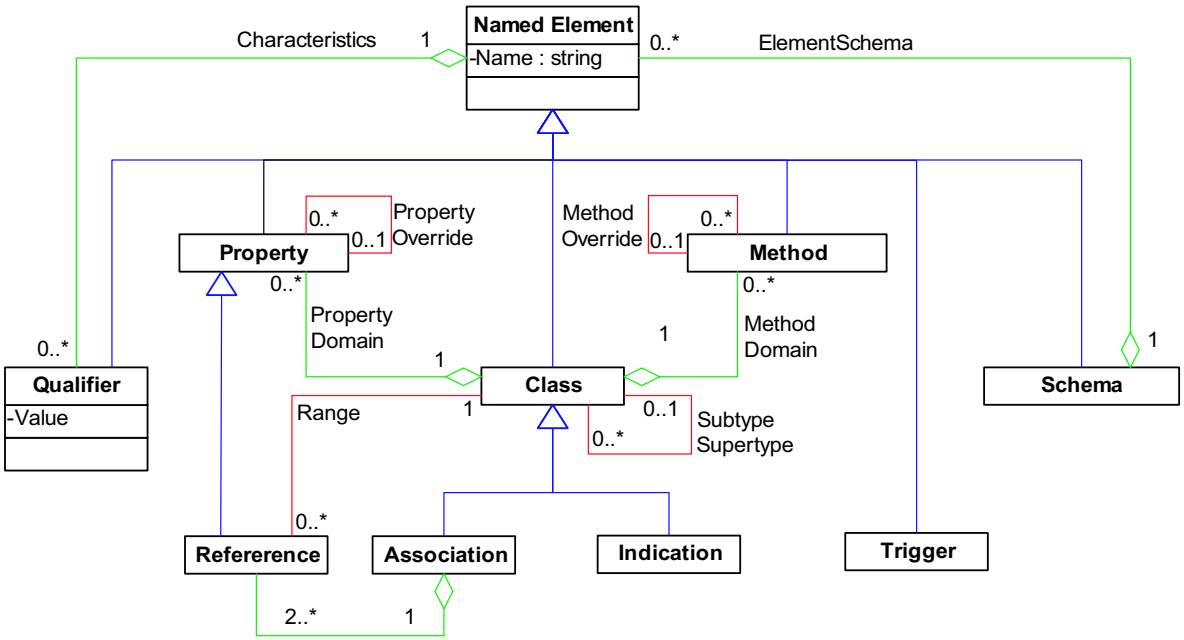

Abbildung A.1: CIM-Metaschema in UML

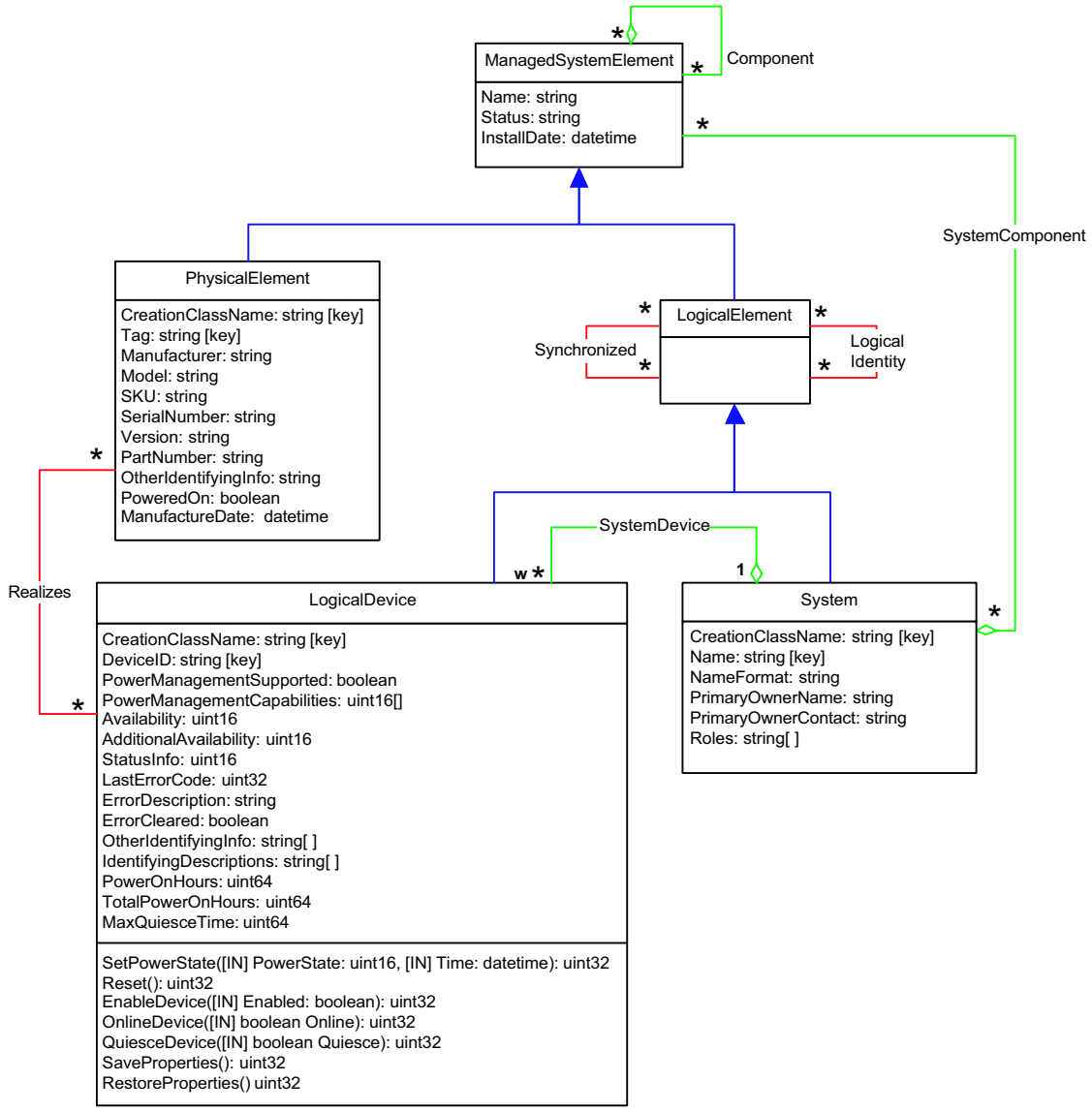

Abbildung A.2: Die ManangedSystemElement - Hierarchie

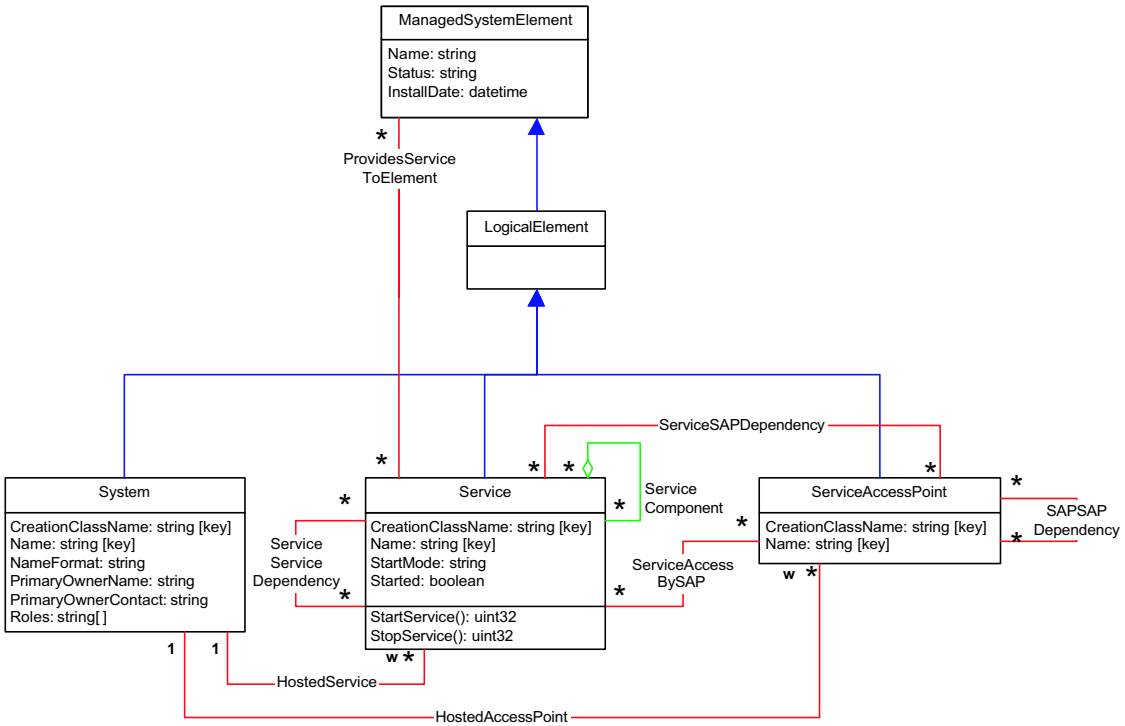

Abbildung A.3: Die CIM Dienst/SAP Klassen

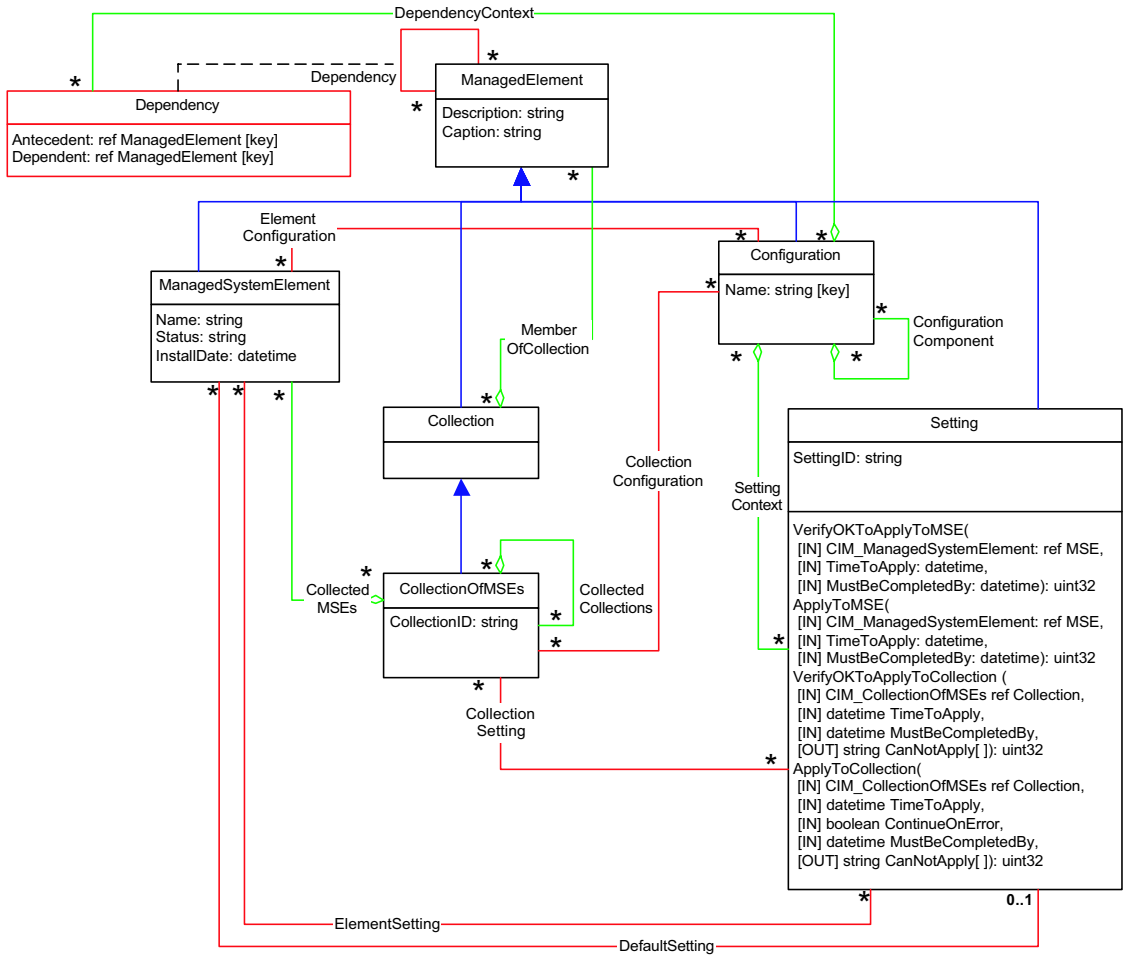

Abbildung A.4: Die Einstellungs-, Konfigurations und Kollektionsklassen

Glossar

ADD Active Dependency Discovery

AFS Das **Andrew File System** ist ein verteiltes Dateisystem, das dafür ausgelegt ist, für eine sehr große Zahl von Workstations (mehr als 10.000) einen transparenten Datenzugriff zu ermöglichen. Dabei ist es unerheblich von welchem Rechner auf eine Datei zugegriffen wird.

API Application Programming Interface

ARS Action Request System

ASCII American Standard Code for Informantion Interchange

BMF Basic Management Functionality

CI Configuration Item

CMDB Configuration Management Database

Common Information Model Das CIM ist ein von der Distributed Management Task Force (DMTF) entwickelter und verabschiedeter Standard für das Management von IT-Systemen. Durch ihn soll es vor allem verteilten Anwendungen möglich sein, eine einheitliche anbieter- und plattformunabhängige Managementschnittstelle zur Verfügung zu stellen. Es behandelt das Netz-, System- und Anwendungsmanagement. CIM stellt ein Datenmodell zur Verfügung, um die Managementinformationen und Funktionen in einem Softwaresystem zu beschreiben.

CSM Customer Service Management

DAG Directed Acyclic Graph

DMTF Distributed Management Task Force

DNS Domain Name Service

Domain Name System Das DNS ist einer der wichtigsten Dienste im Internet. Das DNS ist eine verteilte Datenbank, die den Namensraum im Internet verwaltet. DNS läuft standardmäßig auf Port 53. Hauptsächlich wird das DNS zur Umsetzung von Domainnamen in IP-Adressen benutzt.

E-Mail (v. engl. electronic mail, ,,elektronische(r) Post/Brief“), kurz. auch Mail, bezeichnet eine auf elektronischem Weg in Computernetzwerken übertragene, briefartige Nachricht.

eTOM enhanced Telecom Operations Map

FTP Das File Transfer Protocol (engl. für „Dateiübertragungsverfahren“), ist ein Netzwerkprotokoll zur Dateiübertragung über TCP/IP-Netzwerke. FTP ist in der Anwendungsschicht des TCP/IP Protokollstapels angesiedelt. Es wird benutzt um Dateien von Server zu Client (Download), von Client zu Server (Upload) oder clientgesteuert, zwischen zwei Servern, zu übertragen.

GUI Graphical User Interface (graphische Benutzeroberfläche)

IP Internet Protocol

ITIL IT Infrastructure Library

LMU Ludwig-Maximilians-Universität

LRZ Leibniz-Rechenzentrum

MHN Münchner Hochschulnetz

MNM Munich Network Management

MO Managed Object

MOF Managed Object File

MWN Münchener Wissenschaftsnetz

NFS Der **Network File Service** (früher: Network File System) ist ein von Sun Microsystems entwickeltes Protokoll, das den Zugriff auf Dateien über ein Netzwerk ermöglicht. Dabei werden die Dateien nicht (wie z.B. bei FTP (File Trasfer Protocol)) jedesmal vollständig übertragen, sondern die Benutzer können auf Dateien, die sich auf einem entfernten Rechner befinden, so zugreifen als ob sie auf ihrer lokalen Festplatte abgespeichert wären.

NGOSS New Generation Operations Systems and Software

NSP Network Service Providers

QoS Quality of Service (Dienstgüte)

RDF Resource Description Framework

SAP Service Access Point

SID Shared Information and Data Model

SIP Strategy, Infrastructure & Product

SLA Service Level Agreement

TMF TeleManagement Forums

TUM Technische Universität München

UML Unified Modelling Language

VPN Virtual Private Network

WAN Wide Area Network

Web-Hosting die Unterbringung von Webseiten auf einem an das Internet angeschlossenen Server eines Providers. Der Webhoster stellt dabei als Dienstleister Speicherplatz auf einem Server zur Verfügung, welcher von einem Internetnutzer angemietet werden kann. Der Nutzer hat dann die Möglichkeit, Dateien wie seine Homepage oder auch Bilder in seinem Webspace abzulegen.

XML Extensible Markup Language

Literaturverzeichnis

[Bern 04] BERNSAU, D.: *Erstellung von Entscheidungsbäumen für den Intelligent Assistant der BMW AG*. Diplomarbeit, Ludwig–Maximilians–Universität München, Dezember 2004.

[BeSc96] BECKER, J. und R. SCHÜTTE: *Handelsinformationssysteme*. Landsberg/Lech, 1996.

[Bitt 05] BITTERICH, C.: *Klassifizierung und Modellierung von Dienstmanagement–Informationen — ein Design–Pattern basierter Ansatz* . Diplomarbeit, Ludwig–Maximilians–Universität München, Dezember 2005, http://www.nm.ifi.lmu.de/pub/Diplomarbeiten/domb97 .

[BKH 01] BAGCHI, S., G. KAR und J. HELLERSTEIN: *Dependency Analysis in Distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment*. In: *12th International Workshop on Distributed Systems: Operations and Management DSOM 2001*, Oktober 2001, http://www.loria.fr/ festor/DSOM2001/proceedings/S5-2.pdf .

[Bren 04a] BRENNER, M.: *Vom LAN zum Kommunikationsnetz — Netze und Protokolle*, Kapitel enhanced Telecom Operations Map (eTOM). Interest Verlag, Deutschland, Juni 2004.

[CaRa 99] CASWELL, D. und S RAMANATHAN: *Using Service Models for Management for Internet Services*. In: *HP Technical Report HPL-1999-43*, Palo Alto, CA, USA, März 1999. HP.

[cim00] DMTF: *Common Information Model (CIM) Core Model*. White Paper, Dezember 2000.

[cim 03] *Common Information Model*, 2003, http://www.wbemsolutions.com/tutorials/CIM .

[DHHS06] DANCIU, V., A. HANEMANN, H.-G. HEGERING und M. SAILER: *IT Service Management: Getting the View*. In: *Kern E. M., Brügge B., Hegering H.G.: Managing Development and Application of Digital Technologies*, Munich, Germany, May 2006. CDTM, Springer.

[Domb 97] DOMBACH, T.: *Erstellung und Anwendung eines Kriterienkataloges zur Klassifikation von Managementplattformen*. Diplomarbeit, Ludwig–Maximilians–Universität München, Juni 1997, http://www.nm.ifi.lmu.de/pub/Diplomarbeiten/domb97/PDF-Version/domb97.pdf .

[Dreo 02] DREO, G.: *A Framework for IT-Service Management*. Habilitationsschrift, Ludwig–Maximilians–Universität München, Juli 2002.

[Dreo 95] DREO, G.: *A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to*

Problem–Solving Expertise. Dissertation, Ludwig–Maximilians–Universität München, Juli 1995.

[DrHe 04] DREO RODOSEK, G. und H.-G. HEGERING: *IT–Dienstmanagement: Herausforderungen und Lösungsansätze*. Praxis der Informationsverarbeitung und Kommunikation (PIK), 2004(02/04):85–92, Februar 2004.

[DrKa 97] DREO-RODOSEK, GABI und THOMAS KAISER: *Determining the Availability of Distributed Applications*. In: *Integrated Network Management*, Seiten 207–218, 1997.

[EnKe 01] ENSEL, C. und A. KELLER: *XML–based Monitoring of Services and Dependencies*. In: *Proceedings of GLOBECOM'01 — IEEE Global Telecommunications Conference*, November 2001, http://www.mnm-team.org/pub/Publikationen/enke01b .

[EnKe 01a] ENSEL, C. und A. KELLER: *Managing Application Service Dependencies with XML and the Resource Description Framework*. In: *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, Seiten 661–674, Seattle, Washington, USA, Mai 2001. IFIP/IEEE, IEEE Publishing, http://www.mnm-team.org/pub/Publikationen/enke01 .

[eTOM 04] TMF: *enhanced Telecom Operations Map*. Technischer Bericht, TeleManagement Forum, Februar 2004. Member Evaluation Version 4.0.

[GaKe 01] GARSCHHAMMER, M. und B. KEMPTER: *Application of a Generic Service Model*. In: *8th International Workshop of the HP OpenView University Association (HPOVUA 2001)*, Berlin, Germany, Juni 2001. HPOVUA.

[Gamma 95] GAMMA, E., R. HELM, R. JOHNSON und J. VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, 1995.

[GHHK 01] GARSCHHAMMER, M., R. HAUCK, H.-G. HEGERING, B. KEMPTER, M. LANGER, M. NERB, I. RADISIC, H. ROELLE und H. SCHMIDT: *Towards generic Service Management Concepts — A Service Model Based Approach*. In: *Proceedings of the 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, Seiten 719–732, Seattle, Washington, USA, Mai 2001. IFIP/IEEE, IEEE Publishing, http://www.mnm-team.org/pub/Publikationen/smtf01/ .

[GHHK 02] GARSCHHAMMER, M., R. HAUCK, H.-G. HEGERING, B. KEMPTER, I. RADISIC, H. ROELLE und H. SCHMIDT: *A Case–Driven Methodology for Applying the MNM Service Model*. In: STADLER, R. und M. ULEMA (Herausgeber): *Proceedings of the 8th International IFIP/IEEE Network Operations and Management Symposium (NOMS 2002)*, Seiten 697–710, Florence, Italy, April 2002. IFIP/IEEE, IEEE Publishing, http://www.mnm-team.org/pub/Publikationen/ghhk02 .

[GHKR 01] GARSCHHAMMER, M., R. HAUCK, B. KEMPTER, I. RADISIC, H. ROELLE und H. SCHMIDT: *The MNM Service Model — Refined Views on Generic Service Management*. Journal of Communications and Networks, 3(4):297–306, Dezember 2001, http://www.mnm-team.org/pub/Publikationen/ghkr01/ .

[Grus 98b] GRUSCHKE, B.: *Integrated Event Management: Event Correlation using Dependency Graphs*. In: *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed*

Systems: Operations & Management (DSOM 98), Newark, DE, USA, Oktober 1998. , http://www.mnm-team.org/pub/Publikationen/grus98a/PDF-Version/grus98a.pdf .

[Grus 99] GRUSCHKE, B.: *Entwurf eines Eventkorrelators mit Abhängigkeitsgraphen*. Dissertation, Ludwig–Maximilians–Universität München, November 1999.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[HAN 99a] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt-Verlag, ISBN 3-932588-16-9, 1999, http://www.dpunkt.de/produkte/management.html . 607 S.

[HaSa 05] HANEMANN, A. und M. SAILER: *Towards a Framework for Service–Oriented Event Correlation*. In: *Proceedings of the International Conference on Service Assurance with Partial and Intermittent Resources (SAPIR 2005)*, Lisbon, Portugal, Juli 2005. IARIA/IEEE.

[HaSc 03] HANEMANN, A. und D. SCHMITZ: *Is Service–Orientation Necessary for Event Correlation?* In: *Proceedings of the FMOODS/DAIS 2003 PhD Student Workshop*, Paris, France, November 2003. IFIP.

[HaSc 03a] HANEMANN, A. und D. SCHMITZ: *Approaching Automated Workflow Support for the Application of Generic Service Management Models*. Band 2003, Geneva, Switzerland, Juli 2003. HP OpenView University Association.

[HaSc 04] HANEMANN, A. und D. SCHMITZ: *Service–Oriented Event Correlation — the MNM Service Model Applied to E–Mail Services*. In: *11th International Workshop of the HP OpenView University Association (HPOVUA 2004)*, Band 2004, Paris, France, Juni 2004. , http://www.mnm-team.org/pub/Publikationen/hasc04a .

[HaSc 04a] HANEMANN, A. und D. SCHMITZ: *Service–Oriented Event Correlation — Workflow and Information Modeling Approached*. In: *Third International Workshop on Distributed Event Based Systems (DEBS2004)*, Edinburgh, Scotland, Mai 2004. IEE, http://www.mnm-team.org/pub/Publikationen/hasc04 .

[HLN 01] HEGERING, H.-G., M. LANGER und M. NERB: *Kunde vs. Dienstleister: Die unterschiedlichen Sichten auf Dienstqualität und SLA–Management*. Congressband III:C320.1–C320.13, Februar 2001.

[HSS 04] HANEMANN, A., M. SAILER und D. SCHMITZ: *Assured Service Quality by Improved Fault Management — Service–Oriented Event Correlation*. In: *Proceedings of the 2nd International Conference on Service–Oriented Computing (ICSOC04)*, Seiten 183–192, New York City, NY, USA, November 2004. ACM SIGSOFT and SIGWEB, ACM Press, http://www.mnm-team.org/pub/Publikationen/hss04a .

[HSS 04a] HANEMANN, A., M. SAILER und 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)*, Band 2004, Paris, France, Juni 2004. , http://www.mnm-team.org/pub/Publikationen/hss04 .

[HSS 05] HANEMANN, A., M. SAILER und D. SCHMITZ: *A Framework for Failure Impact Analysis and Recovery with Respect to Service Level Agreements*. In: *Proceedings of the IEEE International Conference on Services Computing (SCC 2005)*, Orlando, Florida, USA, Juli 2005. IEEE.

[HSS 05a] HANEMANN, A., M. SAILER und D. SCHMITZ: *Towards a Framework for IT Service Fault Management*. In: *Proceedings of the European University Information Systems Conference (EUNIS 2005)*, Manchester, England, Juni 2005. EUNIS.

[HSS 05b] HANEMANN, A., M. SAILER und 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, Mai 2005. IFIP/IEEE.

[ITEC] *IBM Tivoli Ebterprise Console*, http://www-306.ibm.com/software/tivoli/products/enterprise-console/ .

[ITS 04] ITSMF: *IT Service Management, eine Einführung basierend auf ITIL*. Van Haren Publishing, Zweite Auflage, Dezember 2004.

[ITSS 00] OGC: *Service Support*. ITIL Foundation, Juni 2000.

[Kais 99] KAISER, T.: *Methodik zur Bestimmung der Verfügbarkeit von verteilten anwendungsorientierten Diensten*. Dissertation, Technische Universität München, April 1999.

[KeKa 01] KELLER, A. und G. KAR: *Determining Service Dependencies in Distributed Systems*. In: *Proceedings of the IEEE International Conference on Communications(ICC) 2001*, Helsinki, Finland, Juni 2001. IFIP/IEEE, IEEE Publishing.

[Kell 98] KELLER, A.: *CORBA–basiertes Enterprise Management: Interoperabilität und Managementinstrumentierung verteilter kooperativer Managementsysteme in heterogener Umgebung*. Dissertation, Technische Universität München, Dezember 1998, http://www.nm.ifi.lmu.de/pub/Dissertationen/kell98/PDF-Version/kell98.pdf .

[KKS01] KELLER, A., H. KREGER und K SCHOPMEYER: *Towards a CIM Schema for Run Time Application management*. In: *Proceedings of the 12th IFIP/IEEE International Workshop on Distributed Systems: Operatios & Management (DSOM 2001)*, Nancy, France, Oktober 2001. IFIP/IEEE.

[LaNe 00] LANGER, M. und M. NERB: *Customer Service Management: An Information Model for Communication Services*. In: *Trends in Distributed Systems: Towards a Universal Service Market. Proceedings of the third International IFIP/GI Working Conference, USM 2000*, September 2000.

[OWSN 05] *HP OpenView Service Navigator Value Pack 8.0 software*, http://www.managementsoftware.hp.com/products/servnav/prod_servnav_0001.html .

[OWSN 05a] *HP OpenView Service Navigator for HP OpenView Operations 8.0 for Unix software*, http://www.managementsoftware.hp.com/products/servnav/index.html .

[Sail05] SAILER, M.: *Towards a Service Management Information Base*. In: *Proceedings of the IBM PhD Student Symposium at the 3rd International Conference on Service-Oriented*

Computing (ICSOC 2005); IBM Research Report, Amsterdam, The Netherlands, December 2005. .

[Schm 01] SCHMIDT, H.: *Entwurf von Service Level Agreements auf der Basis von Dienstprozessen*. Dissertation, Ludwig–Maximilians–Universität München, Juli 2001.

[SID 04] TMF: *Shared Information/Data(SID) Model Addendum 0 - SID Primer - GB922*. Technischer Bericht, TeleManagement Forum, Januar 2004. Member Evaluation Version 1.0.

[WEB 05] WEBMASTER: *Webhosting: virtueller WWW-Server am LRZ*. Leibniz-Rechenzentrum der Bayrischen Akademie der Wissenschaften, Oktober 2005, http://www.lrz-muenchen.de/services/netzdienste/www/v-server/v-server.pdf .

Index