Algorithm Design and Application of Service-Oriented Event Correlation

Andreas Hanemann
German Research Network (DFN)
Stresemannstr. 78
10963 Berlin, Germany
hanemann@dfn.de

Patricia Marcu
Munich Network Management Team
Leibniz Supercomputing Center, Boltzmannstr. 1
85748 Garching, Germany
marcu@mnm-team.org

***Abstract*—The timely and efficient management of faults that affect the quality of services delivered to customers is an important issue for service providers with respect to their business goals. It includes the diagnosis of service faults which deals with the localization of their root causes within subservices and resources being part of the service realization.**

In this paper our service-oriented event correlation approach, which uses event correlation techniques to automate the diagnosis on the service layer is detailed. Our algorithm for the hybrid rule-based/case-based correlation methodology that also includes recently proposed active probing techniques is presented as well as its prototypical implementation at the Leibniz Supercomputing Center. This implementation is not limited to a small test environment, but has been carried out for requirements of the environment of this large service provider.

I. Introduction

Service fault management is an important part of the overall management tasks of an IT service provider. Its aims are mainly twofold: on the one hand, the service quality has to be assured by timely fault diagnosis and recovery, which is particularly needed with respect to service level agreements (SLAs) between service provider and customers. On the other hand, fault management has to be cost effective so that the effort spent on it is minimized.

Therefore, it can be seen that service fault management has critical financial impact both with respect to revenue (customer satisfaction and resulting continued service subscription) and costs (fault management costs itself and SLA violation costs). Guidelines and technical solutions to service fault management are consequently of major interest for the business perspective.

It is important to understand the different nature of faults on the service layer in contrast to the resource layer. While resource faults are usually indicated by events that are predefined by device vendors, service-related fault reports are more ambiguous. They relate to how the quality of the service is perceived by the users and are not limited to a complete failure of a service, but can also indicate that a service is provided with a low quality. While events are reported on the resource layer usually with only short time gaps due to automated reporting, there can be significant time gaps between the witnessing of service-related symptoms by users and their reporting to the provider. Performance is very critical for resource-related events where events are often encountered as event bursts, while there are usually fewer symptom messages related to services.

As a consequence of the aims of service fault management, the need arises to support it with the use of tools to allow for a (partial) automation. In our previous work we have developed a framework describing the components necessary for service fault diagnosis, which is the first part of the overall service fault management. It contains components for formalizing user reports about service symptoms, provider-internal service and resource monitoring as well as resource layer diagnosis. Its central component performs the service fault diagnosis on the service layer for which we have discussed the idea to extend event correlation techniques [1]. While this idea has only been given in a high-level manner yet, in this paper we provide a detailed correlation algorithm and present its application at the Leibniz Supercomputing Center (LRZ), as a large service provider for the scientific and academic community in Munich.

The remaining sections of the paper are organized as follows: The requirements for service fault diagnosis are given in Section II and are compared with the features of correlation techniques in Section III. The stepwise development of the correlation algorithm is detailed in Section IV, while the modeling of information to be dealt with in the context of the algorithm is given in Section V. The algorithm implementation extending the commercial event correlator *IBM Tivoli Enterprise Console* is given in Section VI. Conclusion and future work are provided in Section VII.

II. Requirements

For the design of an automated service fault diagnosis the following requirements have to be taken into account [2]. These have been derived from a generic scenario for service management.

Maintainability: There are often changes in the way services are implemented, which does not only refer to the use of resources, but also to the use of subservices. For example, additional servers being used by a service have to be considered as potential source of problems. The same holds for an application service that is making use of a different connectivity service as before so that the causal relationship in case of faults has changed. In order to achieve an effort reduction using an automated diagnosis, the effort for managing these changes (i.e. for maintaining an information repository for the diagnosis) has to be low.

Modeling: Automated event correlation has to be based on a modeling of the scenario where it is applied to. In this context, it means that a modeling of services, resources, faults, etc is required. This modeling has to be able to represent the information accurately. This accuracy refers e.g. to time conditions and redundancies where the expressiveness of existing modeling solutions is limited. In addition, no assumption that there is only a single root cause at a given point of time should be made. This assumption is a frequent limitation of existing fault management solutions.

Robustness: The correlation should support the fault diagnosis even in situations where its knowledge base may be partially inaccurate. This can easily happen due to the complexity of real-world service implementations where dependencies can be forgotten or modeled inaccurately. In addition, an update of the knowledge base after a failed diagnosis should be supported which can be regarded as a learning capability.

Performance: The performance of the diagnosis has to be sufficient to deliver results in a timely manner. This means that a diagnosis result based on the correlation should be provided in the order of seconds.

III. Related Work

The discussion of related work is focused on a selection of diagnosis techniques whose evaluation with respect to the requirements is summarized in Table I. A good overview of correlation techniques including further methods is given in [3], while our detailed analysis for the needs of service-orientation has been investigated in [2]. It is important to note that in general the approaches presented in the following do not have to be used exclusively.

Model-based reasoning: In model-based reasoning (MBR, [4]) each object is represented by a software agent model with respect to its attributes, behavior, and relation to other models. The correlation is a result of the collaboration of models[1].

For event correlation on the service layer, the approach will require the representation of service behavior as a model which has to be carried out individually for each service provider since service models can hardly be reused in contrast to resource models. While the method is able to model the diagnosis information accurately, it can be difficult to find out what went wrong in a failed correlation. The performance of the approach depends on the implementation which is not explicitly given for MBR.

Rule-based reasoning: In rule-based reasoning (RBR, [4], [5]), a set of rules is used to perform the correlation. A rule has a condition related to received events and the state of the system. The fulfillment of the condition leads to the execution of the rule which triggers actions leading to changes of the system or results in additional events.

The maintenance of the rules is a crucial issue because rule sets can get very large for real world scenarios. This leads to unforeseen interdependencies of rules. Rules are quite flexible in the modeling of knowledge and can represent the information as needed for the service-orientation. However, the approach is going to fail if an unknown situation occurs. Further, in such a case it is easily possible to trace the previous execution of rules which is helpful to identify the modeling inaccuracy. Algorithms like the Rete algorithm [6] exist to perform an effective correlation even for very large rule sets.

An approach from the area of policy-based management uses rules for QoS-related problem determination and directly executes recovery actions [7].

Codebook approach: The codebook approach ([8], [9]) is based on a representation of relationships between events and originating root causes as a matrix, which is the result of a conversion and optimization of an input dependency graph.

Maintenance in this context means to keep the dependency graph up-to-date which is also a crucial issue for this approach. A major drawback are the limitations of the modeling with respect to time constraints, multiple root causes and redundancies which are not covered by the matrix representation and would require extensions. The approach can sometimes deal with unknown situations where the Hamming distance function can be used to find a similar situation. The matching can be done efficiently as a binary vector matching method.

Case-based reasoning: In contrast to the techniques presented before, the case-based reasoning approach (CBR, [10]) needs no prior knowledge about the service implementation. It contains a database of cases which have occurred in the past together with the identified root causes. While the first root causes have to be identified by hand, an automated matching to prior cases is performed at later stages.

The maintenance of the approach has to be judged differently from the other methods because a change in the service implementation does not necessarily require changes in the case database, even though some stored solutions may then not work anymore. The approach is able to model the knowledge as needed and has the capability to learn from previous situations. However, the adaptation of similar solutions has to be performed with some human interaction so that it is less efficient.

Active probing: The methods previously presented are using information gained mainly from the passive reception of events. In contrast, active probing techniques ([11], [12]) use tests to check the performance of components and are using sets of predefined or dynamically composed probes. These techniques can also be combined with the previously presented techniques as carried out for telecommunications services (RBR technique) in [13].

IV. Stepwise Correlation Algorithm Development

The idea of the automated service fault diagnosis is to process formalized user symptom reports and similar information

[1]Please note that this definition is narrower than the one in the often cited article [5]. It has been chosen to allow for classification since a broader definition would also subsume other methods such as RBR and the codebook approach.

Requirement	MBR	RBR	codebook	CBR	probing
Maintenance	-	0	0	+	0
Modeling	+	+	-	+	0
Robustness	0	0	+	+	+
Performance	0	+	+	0	0

TABLE I
METHOD EVALUATION SUMMARY (+: FULFILLED; 0: PARTIALLY FULFILLED; -: MARGINALLY OR NOT FULFILLED)

from the provider's own service monitoring as so called *service events*. For doing so, a hybrid architecture (see Fig. 1) has been designed which combines RBR and CBR, but also integrates active probing techniques (into the RBR module). It primarily uses the rule-based reasoner to traverse the dependencies which are divided into inter-service dependencies, service-resource dependencies, and inter-resource dependencies. The case-based reasoner acts as a backup solution and is becoming active once the rule-based reasoner fails.

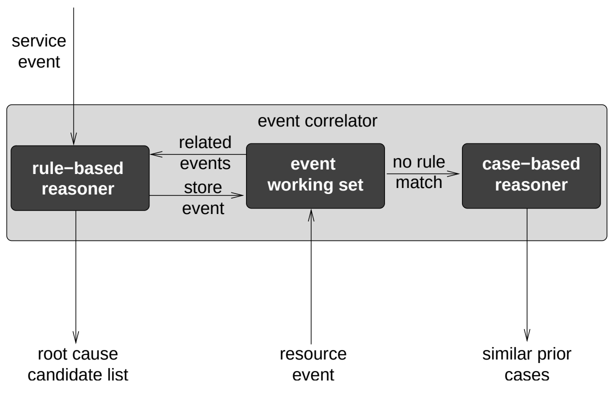


Fig. 1. Hybrid event correlation architecture

This design uses the modeling capabilities of both approaches and the performance of RBR for efficient correlation. The robustness is improved by the application of tests and the CBR learning capability.

In order to cope with the maintenance issue the idea is to generate the rules in a semi-automated way using the service modeling. Since a provider has to maintain the knowledge about the configuration of services and resources as well as their dependencies in any case (in particular to be able to carry out changes in a safe manner), rules can be generated out of this modeling. For instance if a dependency is given between a service and a resource, a rule for matching events related to the service to events related to the resource can be proposed by the system to be approved by a human operator.

A. Assumptions and Basic Algorithm

The algorithm development is carried out in a stepwise manner where several aspects shown in Tab. IV-A are integrated over time. Initially assumptions (A1-11) that these aspects do not exist are made. The basic algorithm is then enhanced when more and more assumptions are removed. The

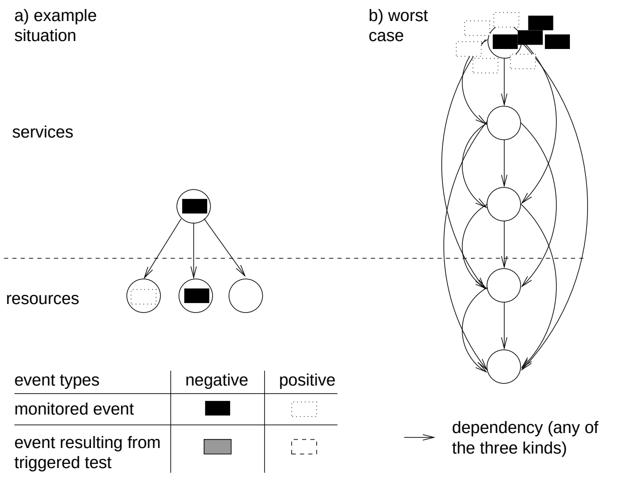


Fig. 2. Events for services and resources in an example situation (a) and in a worst case scenario (b)

last two aspects (O1 and O2) are circumstances that demand an optimization of the algorithm. Please see [2] for the detailed pseudocode representation of the algorithm (both basic and extended versions).

The basic algorithm is quite simple and uses the dependencies to link events related to a service/resource to events for another service/resource if there is a dependency between them. The first kind of elements are called *dependents*, while the second kind are called *antecedents*. An example can be seen in Fig. 2 (a) where a service is a dependent of three (antecedent) resources. This means that a symptom related to the service can be explained by a symptom for one of the resources. The matching is not only done when events for the antecedents indicate symptoms, but also when these events indicate that there are no symptoms. In doing so, the knowledge that these antecedents have been checked is kept. This basic rule is executed for a whole dependency graph so that finally symptoms related to services used by users can be tracked down to resources by following these linkings. These resources are put in a candidate list which is transferred to resource management for detailed checking. Based on the assumptions being made (status known, no provisioning hierarchy, information correct), the algorithm is going to return the correct list of resources.

Even though the algorithm fulfills the aim of the correlation, it does not make use of possibilities for parallel execution which is given by the different kinds of the dependencies. An improved version of the algorithm can therefore correlate inter-service and inter-resource dependencies in parallel at first and then link the resulting events afterwards using the service-resource dependencies. This split-up also has the advantage that organizations can continue using an existing resource event correlation and can add service and aggregate event correlation. Depending on the characteristics of a given scenario, it may be required to further extend this idea to get a correlation hierarchy (compare [14]). For instance, resources within a computing cluster can have their own correlator

New circumstance	Change of algorithm
Starting point.	Basic rule-based algorithm which runs under several assumptions.
A1: There may be services from suppliers so that resources are hidden.	Only the candidate list needs to be changed so that supplier subservices can be put into the candidate list.
A2: There can be maintenance operations affecting the services.	Maintenance information is included in the correlation similar to other events and therefore maintenance can be identified as root cause.
A3: Events may be missing for service and resource status indication.	Active probing is used to trigger tests for services and resources. Consequently, appropriate automatic tests have to be defined.
A4: Time is considered.	The algorithm is split up into different modules which run in parallel. The time conditions make it possible that the event correlation cannot be completed in time. Therefore, the case-based reasoning module is introduced.
A5: There can be redundancies in the service implementation.	It is explained why this does not affect the correlation (in contrast to impact analysis).
A6: There can be multiple events relating to one service or resource.	This information is correlated prior to the main correlation. Time conditions are considered to solve contradictions.
A7: Quality degradations are considered.	While the correlation itself can be left unchanged, additional events have to be introduced for modeling threshold violations. The dependencies also need to be refined for this aspect.
A8: Events can be changed when a mistake in the input has happened.	A procedure for providing input is given. It depends on the progress of the correlation to what extent the correlation can be modified.
A9: Dependencies can change during the correlation.	A validity interval is defined for the dependencies so that only those dependencies that have been present at a certain point in time are considered.
A10: Tests may be missing or inaccurate.	The backup method (CBR) has to deal with the failed rule-based correlation that will occur in this situation.
A11: Dependencies or events may be inaccurately modeled.	Similar to the previous situation, the CBR module will assist to deal with a failed correlation.
O1: Event bursts may occur for service event correlation.	Filtering heuristics may be applied to ensure the stability of the correlation.
O2: Candidate lists should be ordered to give a recommendation which potential causes to examine first.	Failure statistics from the past and deviations from thresholds may be used.

TABLE II
SUMMARY OF ASSUMPTIONS AND ALGORITHM REFINEMENT

and only correlated events from the cluster are sent to other correlators within the organization.

B. Removal of Assumptions

Starting with the removal of the assumptions, the algorithm needs to be changed only slightly for services subscribed from other providers (assumption A1). It needs to be allowed that these subservices are accepted as potential root causes since it is not possible to get information about the subprovider's resources. Maintenance operations (A2) can be included into the algorithm as a special kind of event which can then be used as explanation for detected service or resource events.

While the algorithm has been purely rule-based yet (with rules for executing the described algorithm steps, compare Section V-C), it integrates active probing techniques for removing A3 (missing status information). If information is missing about underlying services or resources, appropriate tests are triggered which then result in events for providing the missing information. This issue is indicated in Fig. 2 (a) where no event is given for the third resource.

The adoption of the algorithm for considering time constrains (arrival times of events, event validities) leads to significant changes of the algorithm (A4). It requires a split-up of the code for different components such as the service event correlation (using inter-service dependencies, depicted in Fig. 3), resource event correlation (inter-resource dependencies), aggregated correlation (service-resource dependencies) and the event working set which is responsible for administrating the assignment of events to the different correlators. This split-up considers the fact that resource events in most cases are received much earlier and can already be correlated prior to the linking to service events. The generation of events via probing also has the consequence that not all events are present at the beginning. It is now also responsible for forwarding events to the case-based reasoner which may be needed to deal with correlations that have not been possible on time.

```
1:  procedure SERVICE EVENT CORRELATION
2:      $serviceEventSet \leftarrow$ null
3:      while true do
4:          add new $service\ events$ to $serviceEventSet$ (received from event working set)
5:          for each $service\ event$ in $serviceEventSet$ do
6:              get antecedents($service$ of the $service\ event$)
7:              if number($antecedent$) = 0 then
8:                  send to subprovider CSM, remove from $serviceEventSet$
9:              else
10:                 for each $antecedent$ in $antecedents$ do
11:                     if $antecedent$ is a service then
12:                         if no event($antecedent$) exists in $serviceEventSet$ then
13:                             if no test($antecedent$) has been triggered yet then
14:                                 trigger test($antecedent$)
15:                             end if
16:                         else
17:                             correlate to previous event
18:                         end if
19:                     else                    ▷ $antecedent$ is a resource
20:                         send $service\ event$ to event working set
21:                     end if
22:                 end for
23:             end if
24:         end for
25:         for each $service\ event$ in $serviceEventSet$ do
26:             if correlation to all antecedents that are services performed then
27:                 if one or more status($antecedent$) = false then
28:                     remove $service\ event$ from $serviceEventSet$
29:                 else
30:                     report $service\ event$ to event working set
31:                     remove $service\ event$ from $serviceEventSet$
32:                 end if
33:             end if
34:             if correlation time slot for $service\ event$ exceeded then
35:                 send $service\ event$ to event working set
36:             end if
37:         end for
38:     end while
39: return
40: end procedure
```

Fig. 3. Procedure for service event correlation

The pseudocode representations for the three correlation modules have structural similarities so that the details are only explained for the service event correlation. The event correlation is running in a loop which is indicated via the *while(true)* statement. In the beginning of each iteration, new service events are received from the event working set which can either be reported from users or the service monitoring as well as be the results of tests. Only negative events are given as input.

A treatment for each event is started in the following: For service events without antecedents (neither resources nor services) the correlation cannot be continued. This situation only exists for events related to services from other providers since internal resources would be existing otherwise. If there are antecedents for the service, a new loop is started (line 10). For subservices it is checked whether events are already existing for the services and if not and no test has been triggered already, a test is requested. If an event exists, the correlation is performed. This means that a link between the current and the antecedent event is denoted. A correlation is performed even if the antecedent event is a positive event in order to save the data that this potential explanation has been excluded. For antecedents that are resources, the service events are sent to the event working set since the correlation to resources is performed in the aggregated correlation.

A second loop for each event is started in line 25 to remove events from the current list of service events. Events for which all antecedents have been checked are no longer relevant for the correlation. Either there are one or more services for which negative events exist (so that the correlation is continued for these events) or there are no symptoms for subservices. In the latter case the event is sent to the event working set where it may be forwarded to the case-based reasoner if the correlation to resources also fails.

The removal of assumption A5 does not require a change in the algorithm because no single root cause assumption has been made. This means that once a match to an event for an antecedent is found the remaining antecedents have still to be checked. The situation is different for impact analysis where the dependencies are traversed in the other direction and require a combined view on the dependencies to decide whether problems of antecedents propagate to dependents.

The removal of the assumption of one representative event per service or resource (A6) makes it necessary to introduce a precorrelation where events related to the same service or resource are matched. For example, a previous event indicating that a service is working should be removed when a service symptom is indicated (more precisely linked to this event to store why it has been regarded to be no longer valid).

The modeling of dependencies has been limited to the binary availability yet, i.e. if services or resources are available or not. For removing A7, this modeling is generalized to allow for dependencies between quality of service (QoS) parameters. Similarly, for resources, the term *quality of resource (QoR)* is introduced to denote a quality feature of a resource. It is then possible to denote, for example, that the delay in the delivery of e-mails is dependent on the mail queue lengths at the mail server.

Some special conditions need to be considered when events or dependencies should be changed during the correlation (A8 and A9). For dependencies additional validity attributes are introduced to check whether a dependency has been relevant at a certain point of time. For dealing with inaccurate information concerning tests, dependencies, and events that causes the rule-based correlation to fail the case-based reasoner is activated as a backup (A10 and A11). Conditions for this are only positive events for antecedents which do not explain a negative event for a dependent as well as event time outs.

Finally, several methods can be applied to improve the resource candidate list (O2). For performance parameters the method by Agrawal et al. [15] can be used to rank the resources according to the deviation from the expected value.

C. Runtime Considerations

The algorithm has a worst case performance of $O(d + e)$ where d is the number of edges in the dependency tree and e is the number of events. This situation is depicted in Fig. 2 (b) where all services and resources may turn out be affected by symptoms. Therefore, all dependencies are traversed and the events are concentrated on a single service so that a precorrelation involving all events is necessary at first. This consideration is related to the rule-based part only assuming that no additional events apart from the active probing results are received during correlation.

D. Case-based Reasoner

The design of the case-based reasoner is performed according to retrieve, adapt, execute and organize steps from the literature [16]. The retrieval of related cases is realized with a key-term matching methodology making use of several attributes (service, service functionality, QoS parameter, service access point, etc). Other retrieval methods such as ones based on sentence structures or geometric distances are not appropriate since special conditions for the attributes do not apply. For the adaptation step, some methods beyond manual adaptation are possible, in particular parameterized adaptation (e.g. if two systems back up each other and in the past one of the systems has failed and now the other one is broken) and procedural adaptation. However, for the procedural adaptation, it need to be considered that the knowledge related to this adaptation possibility should rather be used to improve the modeling and therefore indirectly the rules. For the execution of the adaptation a semi-automated method is used in order to allow for a save way to restore the services. This means that single recovery steps are automatically executed, but not the adaptation as a whole. The cases are organized with a meshed memory, i.e. they are categorized from top-down according to services, service functionalities, and QoS parameters, but the cases for a specific QoS parameter can be linked to other services and service functionalities if appropriate.

E. Comparison to Existing RBR/CBR Combinations

Hybrid RBR/CBR combinations are used in a variety of domains as shown in the categorization article [17]. Apart from our approach, it only refers to one other approach from a related domain (situation management) [18]. This approach uses a combination where the case-based reasoner has a set of situation templates and permanently interacts with the rule-based reasoner to execute rules which then leads to changes of the situation. Due to a commercial background of the approach, its description in the literature is not very detailed yet.

V. Information Modeling of Objects, Events, Rules and Cases

The implementation of the presented algorithm requires an information modeling as basis which is described in this section. For further details refer to [2].

A. Class Model

Due to identified deficits in the existing modeling of service-related information in the literature (CIM [19], NGOSS SID [20], SNMP MIBs), a modeling of services as an extension to CIM has been performed. Its particular focus is on the modeling of dependencies which are crucial for the traversal of the dependency hierarchy in the algorithm.

The basic class model (i.e. without attributes and operations) is shown in Fig. 4. A Service is mainly characterized by the Service Functionalities and the QoS Parameters.

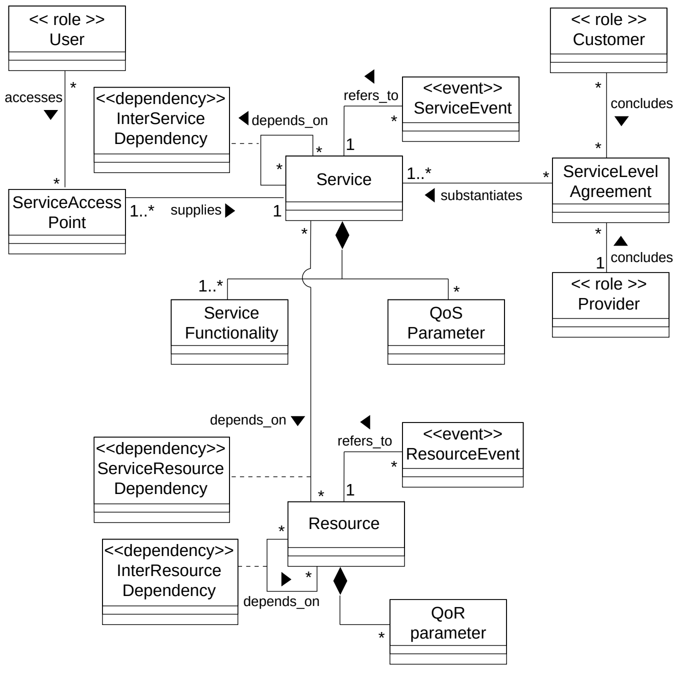


Fig. 4. Basic class model for service fault diagnosis

The idea of the modeling at this point is to allow for different granularities with respect to the Dependencies. Either a Dependency can be tied to the Service as a whole or to a specific Service Functionality. Every Dependency always relates to a QoS Parameter (compare assumption A7). The Service class further requires information about Service Access Points and Service Level Agreements.

For the resources the QoR Parameter class is modeled similar to the QoS Parameter class and the three kinds of dependencies as mentioned for the algorithm are represented accordingly. The details of Events are explained in the next section.

B. Event Modeling

While resource events are usually defined by device vendors, additional considerations are necessary with respect to service events. The idea is to specify them in relation to the SLA term conditions since these are the conditions that a service provider strives to fulfill. The abstract modeling of the events is depicted in Fig. 5 where common attributes for Service and Resource Events are grouped in a Generic Event class.

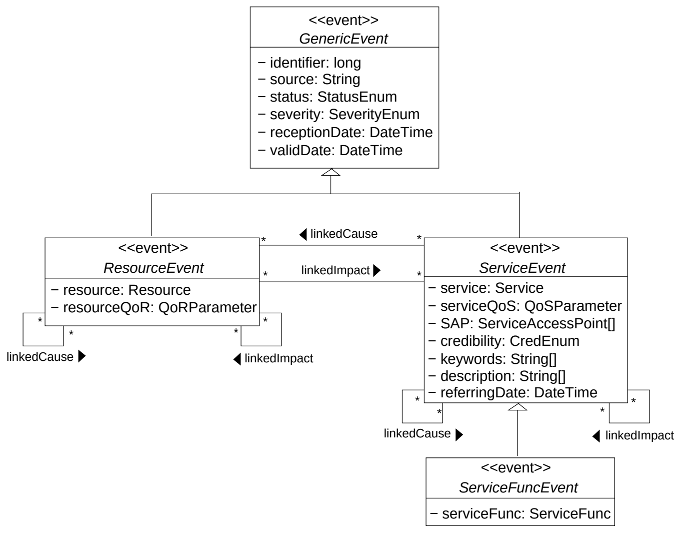


Fig. 5. Event hierarchy classes (abstract classes)

The attributes for Service Events are as follows: Each Service Event is tied to a Service and a QoS Parameter as explained previously so that these pieces of information are recorded. The potential linking to a Service Functionality leads to the derived class Service Functionality Event. The attributes inherited from the generic events comprise an identifier, source (here either users or the service monitoring), status, severity, receptionDate and validDate. For services, specific attributes are required for the Service Access Point, credibility, keywords, description, and referringDate. The credibility attribute is introduced to ensure the accuracy of information since service events directly relate to what has been reported from users (if they are not originated from the service monitoring). The reception of events should therefore try to reproduce the reported symptoms both for gaining additional information and for ensuring the credibility. The keywords field contains a set of keywords which have been acquired in the service event reception. These are usually predefined attributes to allow for case retrieval, but additional user-defined keywords are also recorded. The description is a free text field to enable the manual treatment in the case-based reasoner. The referringDate attribute is used to consider the time gap between the witnessing of a symptom and its reporting as a service event. Such a gap does usually not exist for resource events. The storing of the gap is useful to consider the dependencies that existed at the referringDate.

C. Rules and Cases

The implementation of the algorithm using RBR requires an expression of algorithm statements via rules. For this purpose, rules are denoted in a generic manner with event, condition and action statements.

For example, the precorrelation of events into a representative event has to be expressed by rules (compare assumption A6) for which one of the rules is given in Fig. 6.

event $event1$, $event2$
condition $event1.class$ equals $eventclassA$ (here service class)
and $event2.class$ equals $eventclassB$ (here service class)
and $event1.status$ equals OPEN
and $event2.status$ equals OPEN
and $event1.referringDate$ less_than $event2.referringDate$
and $event1.SAP$ equals $event2.SAP$
action $event1.status$ set CORRELATED
and $event1.linkedCause$ add $event2$

Fig. 6. Example rule representation

Here two currently valid events (status open) are matched if they correspond to two related classes of events (e.g. up and down events for a service) and relate to the same SAP. $Event1$ is older than $event2$ so that the information contained in $event2$ is considered to be more accurate then the previous information. $Event1$ is therefore set to be correlated and linked with $event2$. Therefore, it can be tracked why the event has been set to be correlated.

For the modeling of cases, a generic case template is provided to show which kind of information has to be recorded and processed. The information that is needed is mainly contained in the service event data enriched with partial results from the correlation, e.g. if the correlation has been partially successful. Furthermore, case processing information needs to be recorded, in particular which related cases can be retrieved, when has the processing been started, what has been the real root cause and the solution steps.

VI. Prototypical Implementation at the LRZ

The algorithm has been implemented at the LRZ to introduce an event correlation for its Web Hosting Service and E-Mail Service. These are large scale services which host web sites for more than 350 customer institutions and provide e-mail access for more than 85,000 users, respectively.

A. Implementation Options and Choice of TEC

For the implementation of the rule-based reasoner several tools have been evaluated. General purpose RBR systems such as *JBoss Rules* [21] or *Boeing's NodeBrain* [22] have the advantage that their code is open source so that its details are not hidden and can potentially be modified. However, these systems are not designed for the network and systems management domain per se so that additional interfaces need to be developed. The open source tool *Simple Event Correlator* [23], which has also been integrated into *HP Event Correlation Services* [24], has too few possibilities and is relatively hard to extend. It has then been decided to use the *IBM Tivoli Enterprise Console (TEC)* [25] because it can be adapted for the needs of service-orientation and has already modules to integrate it to the given environment at the LRZ (i.e. *HP OpenView NetworkNodeManager* for network management and *BMC Remedy ARS* as trouble ticket solution).

Nevertheless, the implementation had to overcome some limitations of the TEC standard rule sets. Similar to other tools, the predefined rules are based on a single root cause assumption. Consequently, after the correlation of an event for an antecedent to an event for a dependent the latter event is regarded as fully correlated and closed. The predefined rules are also too limited concerning the modeling of dependencies for which only two rules related to IBM WebSphere exist.

B. Rule Set

The implemented rule set is depicted in Fig. 7. Its first three rules serve administrative purposes for starting and closing the correlation. The rule *duplicate_services* is responsible for precorrelating events related to the same service (compare assumption A6).

The *correlate* rule is the central rule in the rule set and is split into a set of actions. It implements the top-down correlation using the linking and active probing helper rules. The rule is executed for a quality degradation with respect to a service or service functionality. In the *correlate_resources* action, the service-resource dependencies are specified and it can, therefore, determine which antecedent resource QoR parameters are given, for the input QoS parameter. It is then checked whether events are present for the antecedent resources according to these dependencies. For dependencies where events for the antecedents are given the linking is initiated. In the action *check_resource_restlist* active probes are triggered for the remaining dependencies. A similar handling for inter-service dependencies is done in the *correlate_services* and *check_service_restlist* actions where the list of dependencies is used to identify the antecedent services and to search for given events. For missing events, active probing is requested.

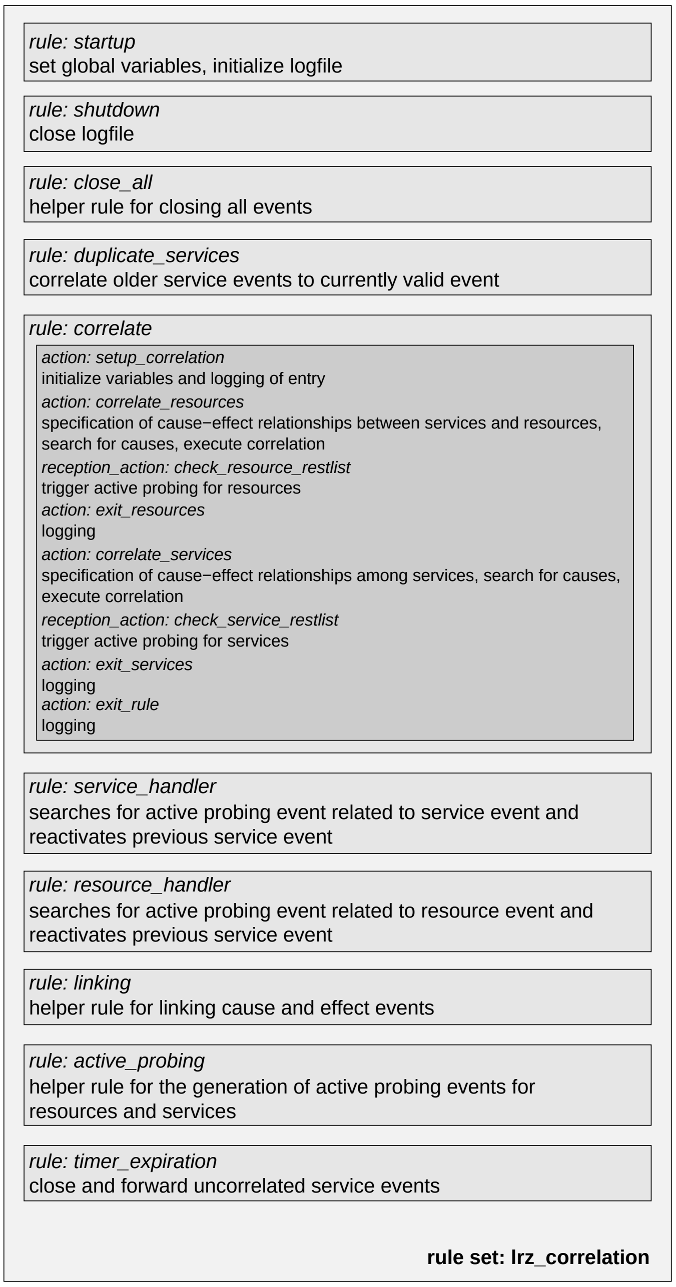
The actions in the correlation rule have been based on the assumption that events for antecedents are already given which may not be the case. Therefore, it is examined in the *service_handler* and *resource_handler* rules whether a current service or resource event is the result of active probing. The service event that triggered active probing is then reactivated and is again input for the correlation rule. The correlation rule is reexecuted for this service event so that a linking between the current event and the previous higher level event can be constructed.

The *linking* rule is a helper rule to perform the linking of events. Another helper rule is the *active_probing* rule which splits an active pro bing request related to a set of services or resources into single probing events.

For service events that have reached the end of their validity, the rule *timer_expiration* is in place to forward them to the case-based reasoner.

C. Correlation Example

In order to illustrate the service-oriented event correlation, an example for the Web Hosting Service is provided in Fig. 8.


rule: startup
set global variables, initialize logfile

rule: shutdown
close logfile

rule: close_all
helper rule for closing all events

rule: duplicate_services
correlate older service events to currently valid event

rule: correlate

action: setup_correlation
initialize variables and logging of entry
action: correlate_resources
specification of cause−effect relationships between services and resources, search for causes, execute correlation
reception_action: check_resource_restlist
trigger active probing for resources
action: exit_resources
logging
action: correlate_services
specification of cause−effect relationships among services, search for causes, execute correlation
reception_action: check_service_restlist
trigger active probing for services
action: exit_services
logging
action: exit_rule
logging

rule: service_handler
searches for active probing event related to service event and reactivates previous service event

rule: resource_handler
searches for active probing event related to resource event and reactivates previous service event

rule: linking
helper rule for linking cause and effect events

rule: active_probing
helper rule for the generation of active probing events for resources and services

rule: timer_expiration
close and forward uncorrelated service events

rule set: lrz_correlation

Fig. 7. Implemented correlation rule set [2]

On the left side events received from the service monitoring are shown (same semantics as in Fig. 2), while helper events are depicted on the right side.

The Web Hosting Service is an offer to smaller research institutions to run their web sites at the LRZ. The service is split into several functionalities such as the retrieval of static web pages and is based on subservices (e.g. Firewall Service) and resources (e.g. web servers). The example relates to the QoS parameter availability (i.e. whether the service is available for users at a certain point in time) for which also similar QoD parameters for the resources exist.

The example starts with an unavailability event for a re-

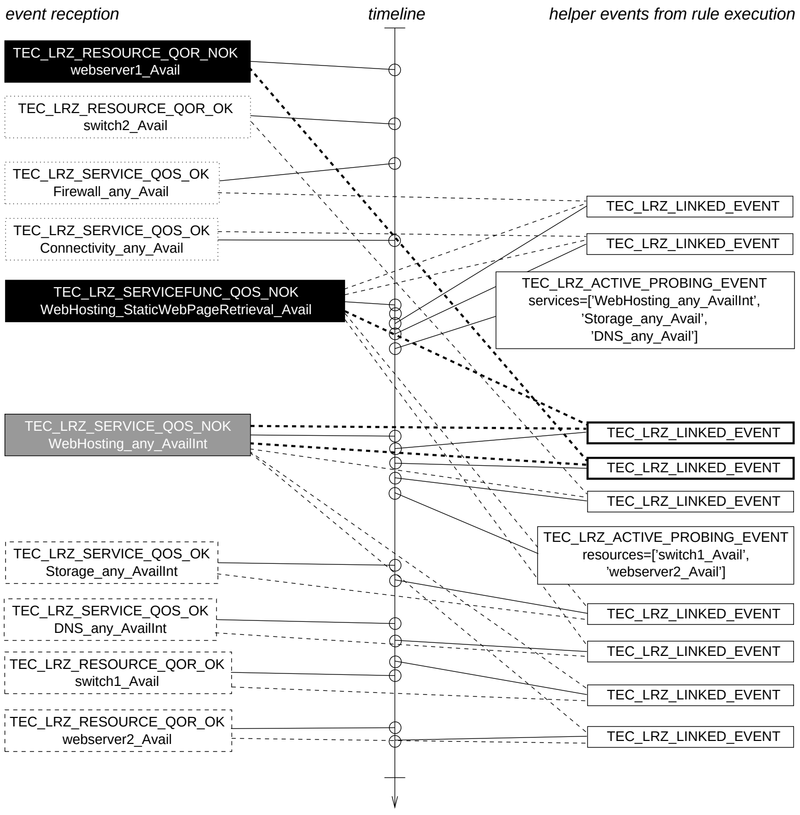

event reception
timeline
helper events from rule execution
TEC_LRZ_RESOURCE_QOR_NOK webserver1_Avail
TEC_LRZ_RESOURCE_QOR_OK switch2_Avail
TEC_LRZ_SERVICE_QOS_OK Firewall_any_Avail
TEC_LRZ_SERVICE_QOS_OK Connectivity_any_Avail
TEC_LRZ_SERVICEFUNC_QOS_NOK WebHosting_StaticWebPageRetrieval_Avail
TEC_LRZ_SERVICE_QOS_NOK WebHosting_any_AvailInt
TEC_LRZ_SERVICE_QOS_OK Storage_any_AvailInt
TEC_LRZ_SERVICE_QOS_OK DNS_any_AvailInt
TEC_LRZ_RESOURCE_QOR_OK switch1_Avail
TEC_LRZ_RESOURCE_QOR_OK webserver2_Avail
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT
TEC_LRZ_ACTIVE_PROBING_EVENT services=['WebHosting_any_AvailInt', 'Storage_any_Avail', 'DNS_any_Avail']
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT
TEC_LRZ_ACTIVE_PROBING_EVENT resources=['switch1_Avail', 'webserver2_Avail']
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT
TEC_LRZ_LINKED_EVENT

Fig. 8. Example correlation for the Web Hosting Service (abstracted from [2])

source (webserver1) which does not lead to further correlation actions, but can be a root cause candidate already. Please note that due to the pure top-down approach affected services, etc. are not determined proactively. The usual monitoring of services and resources does also result in positive events such as for the availability of switch2, Firewall Service, and Connectivity Service.

Then, a service event is received that the retrieval of static web pages for the Web Hosting Service is not available. In the correlation routine, it is at first checked whether valid events for subservices or resources of this service functionality are available which applies to the two service events previously witnessed (Firewall Service and Connectivity Service). An active probing event is therefore issued for the remaining subservices, which results in the reporting of results for the performed tests.

The first result shows that the internal availability (service-internal QoS parameter summarizing the performance of the resources) of the service is violated which is linked to the reported unavailability of the static web page retrieval functionality (highlighted with thicker lines in the figure). This failure has to be further investigated again using the correlation rule so that the two resource events witnessed at the start of the example can now be linked to this internal unavailability including the negative event for webserver1 (linking also highlighted). Furthermore, the correlation results in an active probing event for the remaining resources. At the end, further test results for the first and second active probing are reported and are matched to the originating negative events.

In summary, the unavailability of the static web page retrieval functionality of the Web Hosting Service can be explained by the unavailability of webserver1.

D. Case-based Reasoner Implementation

Similar to the RBR module, the possibilities for tool support of the case-based reasoner have been examined. Only few open source and commercial CBR tools related to network and systems management exist so that often solutions based on trouble ticket systems are used. A general open source CBR system is *jColibri* [26], while *Empolis Orenge* [27] is a mighty commercial tool. The tool *Weka* [28] is suitable for the retrieval step, but similar to the others it has to be adapted for the network and service management domain.

For the LRZ the way to implement the CBR module is to extent its BMC Remedy ARS [29] installation. The function of BMC Remedy ARS to retrieve related trouble tickets manually with a search function is going to be enhanced with a key-term matching function.

VII. Conclusions and Future Work

In this paper the idea of the hybrid architecture for performing service-oriented event correlation has been motivated by analyzing the features of event correlation methods. The development of the correlation algorithm has been explained by detailing the steps for its refinement. The implementation of the rule-based reasoning part extending TEC has been outlined containing a simplified correlation example.

Concerning future work, two main directions can be followed, i.e. improving the event correlation and service-related developments. For the event correlation, a more active role for the case-based reasoner can be examined, in particular by specifying cases that represent the overall situation in contrast to representing a single event. Since faults and security-related incidents (e.g. denial of service attacks) can have similar consequences for users, it should be examined how to integrate security-related events into the correlation. For the service domain a refinement for Web services and Grid services seems to be promising.

In addition, there is currently some uncertainty in organizations (compare [30]) concerning the ways event correlation in larger enterprises where the correlation efforts have to be distributed. It is, for instance, not obvious how to specify the tasks of a local correlation (e.g. for a computing cluster) and how to define and generate aggregated events that represent the state of a set of resources. Therefore, best practice guidelines should be provided to figure out how to analyze and improve the situation in a given organization.

Acknowledgments

The authors wish to thank the members of the MNM Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team, directed by Prof. Dr. Heinz-Gerd Hegering, is a group of researchers of the Munich Universities, the University of Federal Armed Forces in Munich and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its webserver is located at `http://www.mnm-team.org`.

References

[1] A. Hanemann, "A Hybrid Rule-Based/Case-Based Reasoning Approach for Service Fault Diagnosis," in *Proceedings of 20th International Conference on Advanced Information Networking and Application (AINA2006); includes proceedings of International Symposium on Frontiers in Networking with Applications (FINA 2006)*. Vienna, Austria: IEEE press, April 2006, pp. 734–738.

[2] A. Hanemann, "Automated IT Service Fault Management Based on Event Correlation Techniques," PhD thesis, University of Munich, Department of Computer Science, Munich, Germany, July 2007.

[3] M. Steinder and A. Sethi, "A Survey of Fault Localization Techniques in Computer Networks," *Science of Computer Programming, Elsevier*, vol. 53, no. 1, pp. 165–194, 2004.

[4] L. Lewis, *Service Level Management for Enterprise Networks*. Artech House, Inc., 1999.

[5] G. Jakobson and M. Weissman, "Alarm Correlation," *IEEE Network*, vol. 7, no. 6, November 1993.

[6] C. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artifical Intelligence Journal*, vol. 19, no. 1, pp. 17–37, 1982.

[7] G. Molenkamp, H. Lutfiyya, M. Katchabaw, and M. Bauer, "Diagnosing Quality of Service Faults in Distributed Applications," in *Proceedings of the 20th IEEE International Performance, Computing, and Communications Conference*, Phoenix, Arizona, USA, April 2002, pp. 375–382.

[8] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A Coding Approach to Event Correlation," in *Proceedings of the Fourth IFIP/IEEE International Symposium on Integrated Network Management*, Santa Barbara, California, USA, May 1995, pp. 266–277.

[9] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High Speed and Robust Event Correlation," *IEEE Communiations Magazine*, vol. 34, no. 5, May 1996.

[10] L. Lewis, *Managing Computer Networks - A Case-Based Reasoning Approach*. Artech House, Inc., 1995.

[11] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik, "Real-time Problem Determination in Distributed Systems Using Active Probing," in *Proceedings of the 9th IFIP/IEEE International Network Management and Operations Symposium (NOMS 2004)*, Seoul, Korea, April 2004, pp. 133–146.

[12] N. Odintsova, I. Rish, and S. Ma, "Multi-fault Diagnosis in Dynamic Systems," in *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005, Poster-CD)*, Nice, France, May 2005.

[13] S. Shankar and O. Satyanarayanan, "An Automated System fo Analyzing Impact of Faults in IP Telephony Networks," in *Proceedings of the 10th IFIP/IEEE International Network Management and Operations Symposium (NOMS 2006)*, Vancouver, British Columbia, Canada, April 2006.

[14] J. Martin-Flatin, "Distributed Event Correlation and Self-Managed Systems," in *Proceedings of the 1st International Workshop on Self-* Properties in Complex Information Systems (Self-Star 2004)*, Bertinoro, Italy, May 2004, pp. 61–64.

[15] M. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer, "Problem Determination Using Dependency Graphs and Run-Time Behavior Models," in *Proceedings of the 15th IFIP International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, Davis, California, USA, November 2004, pp. 171–182.

[16] G. Jakobson, L. Lewis, and J. Buford, "An Approach to Integrated Cognitive Fusion," in *Proceedings of the 7th ISIF International Conference on Information Fusion (FUSION2004)*, Stockholm, Sweden, June 2004, pp. 1210–1217.

[17] I. Hatzilygeroudis and J. Prentzas, "Categorizing Approaches Combining Rule-Based and Case-Based Reasoning," *Journal of Expert Systems, Blackwell Publishing*, vol. 24, no. 2, pp. 97–122, May 2007.

[18] G. Jakobson, J. Buford, and L. Lewis, "Towards an Architecture for Reasoning about Complex Event-Based Dynamic Situations," in *Proceedings of the Third International Workshop on Distributed Event Based Systems (DEBS 2004)*, Edinburgh, Scotland, May 2004.

[19] "CIM Standards, Version 2.13, Distributed Management Task Force," http://www.dmtf.org/standards/cim, Sep. 2006.

[20] "Shared Information/Data (SID) Model, Addendum 4S0 - Service Overview Business Entity Definitions," TeleManagement Forum, NGOSS Release 4.0, Aug. 2004.

[21] "JBoss Rules, JBoss - a division of RedHat Linux," http://-labs.jboss.com/portal/jbossrules/.

[22] "NodeBrain - An Open Source Agent for Event Monitoring Applications, The Boeing Company," http://www.nodebrain.org.

[23] R. Vaarandi, "Platform Independent Event Correlation Tool for Network Management," in *Proceedings of the 8th IFIP/IEEE International Network and Operations Management Symposium (NOMS 2002)*, Florence, Italy, April 2002, pp. 907–909.

[24] "HP OpenView Event Correlation Services, Hewlett Packard Corporation," http://www.managementsoftware.hp.com/products/ecs/.

[25] "IBM Tivoli Enterprise Console, International Business Machines Corporation," http://www-306.ibm.com/software/tivoli/products/enterprise-console/.

[26] "jColibri - Case Based Reasoning framework," http://gaia.fdi.ucm.es/-projects/jcolibri/.

[27] "Empolis Orenge," http://www.empolis.de.

[28] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java, University of Waitako," http://www.cs.waikato.ac.nz/˜ml/weka/.

[29] "BMC Remedy Action Request System, BMC Remedy Corporation," http://www.bmc.com/remedy/.

[30] J. Martin-Flatin, G. Jakobson, and L. Lewis, "Event Correlation in Integrated Management: Lessons Learned and Outlook," *Journal of Network and Systems Management*, vol. 17, no. 4, December 2007.