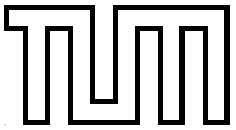


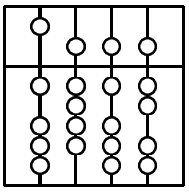
INSTITUT FÜR INFORMATIK

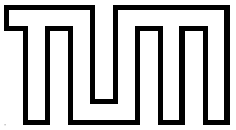
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Service-Oriented Event Correlation

Bearbeiter:	Hans Beyer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Andreas Hanemann David Schmitz





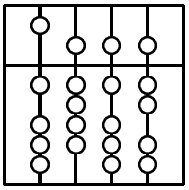
INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit

Service-Oriented Event Correlation

Bearbeiter:	Hans Beyer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Andreas Hanemann David Schmitz
Abgabetermin:	15. Januar 2007



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

München, den 15. Januar 2007

. . . . . .

(Unterschrift des Kandidaten)

Immer mehr Unternehmen lagern IT-Dienste insbesondere aus Kostengründen auf externe Dienstleister aus. Der dabei entstehenden Abhängigkeit zu diesen Providern versuchen die Auftraggeber mit sogenannten *Service Level Agreements* entgegenzutreten, in welchen die Qualität der Dienstleistungen geregelt wird.

Verpflichtet durch SLAs müssen Provider ein gut organisiertes Fehlermanagement seiner angebotenen Dienste aufweisen können, um die Einhaltung vertraglich festgelegter Dienstgüteparameter zu gewährleisten und mögliche Minderleistungen oder gar Schadenersatzforderungen abzuwenden.
Bezogen sich bisherige Fehlermanagementsysteme fast ausschließlich auf die Verwaltung einer Netzinfrastruktur, also der technischen Ressourcen, entstand mit der wachsenden Bedeutung von IT-Dienstleistungen der Bedarf an einer Erweiterung des Wirkungsbereiches dieser Systeme auf die angebotenen Dienstleistungen. Um diesen Bedarf zu decken ist es erforderlich, Fehlermeldungen nicht nur von technischen Komponenten, sondern auch von Benutzern in die Prozesse des Fehlermanagemensystems einzubeziehen.

Im Rahmen dieser Arbeit wird eine allgemeine Vorgehensweise aufgezeigt werden, wie solche Meldungen mit Ereignissen auf Infrastrukturebene verknüpft werden können, um eine gezielte Ursachenermittlung für ein Problem durchführen zu können. Dazu wurde zunächst ein Anforderungskatalog anhand zweier Szenarien am LRZ ausgearbeitet, dessen Inhalte schrittweise erfüllt werden.

Im zweiten Teil werden diese entwickelten Konzepte auf einem kommerziellen Produkt angewendet. Eine Analyse der vorhandenen Möglichkeiten dieses Produkts zeigt, dass diese nicht im gewünschten Umfang für eine dienstorientierte Ereigniskorrelation verwendet werden können. Daher wurde die Software durch Umsetzung der entwickelten allgemeinen Konzepte um neue Funktionalitäten erweitert. Die in praxisnahen Versuchen erreichten Ergebnisse belegen eine erfolgreiche Verknüpfung zwischen abstrakten Ereignissen einen Dienst betreffend und realen Problemen auf Seiten der Ressourceninfrastruktur.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Aufgabenstellung . . . 3
- 1.2 Vorgehensweise und Gliederung . . . 3
- 1.3 Begriffsbestimmung . . . 5

2 Szenario und Anforderungsanalyse **7**
- 2.1 Das Szenario am LRZ . . . 7
  - 2.1.1 E-Mail Dienst am LRZ . . . 7
  - 2.1.2 Webhostingdienst am LRZ . . . 9
  - 2.1.3 Zusammenfassung . . . 10
- 2.2 Verwendung des regelbasierten Ansatzes . . . 10
  - 2.2.1 Regelbasierte Systeme . . . 11
  - 2.2.2 Vorteile des regelbasierten Ansatzes . . . 12
  - 2.2.3 Nachteile des regelbasierten Ansatzes . . . 13
- 2.3 Anforderungskatalog . . . 13
  - 2.3.1 Anforderungen aus der Verwendung eines regelbasierten Ansatzes . . . 13
  - 2.3.2 Anforderungen zur Filterung von Events . . . 14
  - 2.3.3 Anforderungen zur Dienstorientierung . . . 14
  - 2.3.4 Anforderungen an die Handhabung von Ressourcenfehlern . . . 16

3 Vorhandene Ansätze **17**
- 3.1 ITIL . . . 17
- 3.2 MNM Dienstmodell . . . 18
  - 3.2.1 Vorstellung . . . 18
  - 3.2.2 Zusammenfassung . . . 21
- 3.3 Verfeinerung des MNM Dienstmodells - Abhängigkeitenmodellierung . . . 21
  - 3.3.1 Modellierung der Abhängigkeiten . . . 21
  - 3.3.2 Zusammenfassung . . . 23
- 3.4 Dienstorientierte Ereigniskorrelation . . . 23
- 3.5 Produkte . . . 24
  - 3.5.1 HP OpenView . . . 24
  - 3.5.2 IBM Tivoli Enterprise Console . . . 24

4 Vorgehensweise zur dienstorientierten Ereigniskorrelation **29**
- 4.1 Ereigniskorrelation durch Aussagenlogik . . . 30
  - 4.1.1 Elemente der Aussagenlogik . . . 30
  - 4.1.2 Getroffene Annahmen . . . 30
  - 4.1.3 Anwendung auf die Ereigniskorrelation . . . 31
  - 4.1.4 Anwendung am Szenario . . . 33
  - 4.1.5 Zusammenfassung und Bewertung . . . 34
- 4.2 Der Faktor Zeit . . . 36
  - 4.2.1 Correlation Window - Zeitfenster . . . 36
  - 4.2.2 Beispiel . . . 38
  - 4.2.3 Klassendiagramm . . . 38
  - 4.2.4 Zusammenfassung . . . 39
- 4.3 Dienstgüteparameter und Dienstfunktionalitäten . . . 40
  - 4.3.1 QoS - Dienstgüteparameter . . . 40
  - 4.3.2 Dienstfunktionalitäten . . . 40

4.3.3 Erweiterung des Klassendiagramms . . . 41
4.3.4 Beispiel . . . 44
4.3.5 Zusammenfassung und Bewertung . . . 46
4.4 Ereigniskorrelation mit Score . . . 47
4.4.1 Score für Dienste, Ressourcen und Events . . . 47
4.4.2 Erweiterung des Klassendiagramms . . . 52
4.4.3 Zusammenfassung und Bewertung . . . 53
4.5 Korrelationsalgorithmus . . . 54
4.5.1 Anpassung des Korrelationsalgorithmus . . . 54
4.5.2 Prozeduren . . . 57
4.5.3 Zusammenfassung . . . 60

5 Anwendung am Szenario LRZ **61**
5.1 Szenario und Ausgangslage . . . 61
5.1.1 Szenario am LRZ . . . 62
5.1.2 Tivoli Enterprise Console . . . 63
5.2 Implementierung . . . 69
5.2.1 Ereignisdefinitionen . . . 69
5.2.2 Regelerstellung: lrz_correlation Regelsatz . . . 74
5.3 Resultate . . . 80
5.3.1 Versuch 1 . . . 80
5.3.2 Versuch 2 . . . 84
5.4 Zusammenfassung . . . 87

6 Zusammenfassung und Ausblick **89**

A Verwenden der Tivoli Enterprise Console **91**
A.1 Zugriff und Rechte . . . 91
A.2 Arbeiten mit Regelbasen . . . 91
A.3 Arbeiten mit Regelsätzen . . . 92
A.4 Arbeiten mit Regeln . . . 93
A.5 Arbeiten mit Events . . . 93

B Implementierter Regelsatz **94**

C Definition der Eventklassen **107**

Abbildungsverzeichnis

1.1 Ereigniskorrelation im dienstorientierten Fehlermanagement . . . 2
1.2 Vorgehensweise . . . 4

2.1 Ressourcen des E-Mail Dienstes am LRZ (aus [Hane 07]) . . . 8
2.2 Infrastruktur zur Auslieferung von Webseiten am LRZ (aus [Hane 07]) . . . 9
2.3 Korrelation . . . 11
2.4 Elemente regelbasierter Systeme . . . 12
2.5 Dienstinstanzen . . . 15

3.1 Lebenszyklus eines Dienstes nach dem MNM Dienstmodell . . . 19
3.2 Übersicht über das MNM Dienstmodell . . . 20
3.3 Abhängigkeitenmodellierung nach [Marc 06] . . . 22
3.4 Klassendiagramm des Abhängigkeitsmodells . . . 23
3.5 Komponenten der Tivoli Enterprise Console . . . 26
3.6 Der TEC EventViewer. Im Vordergrund eine Balkendarstellung von aktiven Ereignissen. Die Farbe spiegelt dabei den Schweregrad der Events wieder. . . . 28

4.1 Komponenten und Beziehungen . . . 32
4.2 Beteiligte Funktionalitäten beim Webhosting Dienst des LRZ . . . 33
4.3 Zeitfenster für Events . . . 37
4.4 UML Beschreibung von Komponenten . . . 38
4.5 UML Beschreibung von Events . . . 39
4.6 Klassendiagramm der Klasse QoS . . . 41
4.7 Service Event . . . 42
4.8 Die Klasse *ServiceBuildingBlock* aus der Abhängigkeitenmodellierung von [Marc 06] . . . 42
4.9 Modellierung von *Components* und der QoS Klasse . . . 43
4.10 Beispielhafte Modellierung des Zugriffs auf geschützten Bereich . . . 45
4.11 Beispiel für ein Service Event . . . 46
4.12 Score für Dienste, Ressourcen und Events . . . 48
4.13 Automatische Scoreregression durch Zeit . . . 50
4.14 Erweiterung des Components Klassendiagramms . . . 52
4.15 Erweiterung des Events Klassendiagramms . . . 53
4.16 Übersicht über den Korrelationsalgorithmus als UML Aktivität . . . 54
4.17 Hinzufügen von Events als UML Aktivitätsdiagramm dargestellt . . . 55
4.18 Die Ereigniskorrelation als UML Aktivitätsdiagramm dargestellt . . . 56

5.1 Abhängigkeiten des Webhosting Dienstes auf Dienstebene . . . 62
5.2 Abhängigkeiten des Webhosting Dienstes auf Ressourcenebene . . . 63
5.3 Beziehungen zwischen Events wie in Listing 5.1 definiert . . . 64
5.4 Graphischer Regeleditor der TEC . . . 68
5.5 Übersicht über das *lrz_correlation rule set* . . . 75
5.6 Zeitlicher Abfolge der eintreffenden und generierten Events . . . 81
5.7 Zeitlicher Abfolge der eintreffenden und generierten Events . . . 85

Tabellenverzeichnis

2.1 Durchschnittliche Anzahl E-Mails pro Tag am LRZ im Jahresmittel ([LRZ]) 8

4.1 Bewertung nach Anforderungen 35
4.2 Bewertung der Anforderungen nach Einführung des Correlation Window 40
4.3 Beispiele für QoS Parameter 40
4.4 Unterschiedliche Dienstfunktionalitäten des Webhosting Dienstes 44
4.5 Bewertung nach Anforderungen nach Einführung von QoS Parametern und Dienstfunktionalitäten 46
4.6 Bewertung nach Anforderungen nach Einführung der Score 53
4.7 Erfüllte Anforderungen 60

1 Einleitung

Schon seit den 90er Jahren ist auch in der Kommunikations- und Informationstechnologiebranche (IT-Branche) eine Entwicklung zu beobachten, die die Auslagerung von ehemals internen Geschäftsprozessen auf externe Dienstleister vorsieht. Viele Unternehmen erreichen damit eine Reduzierung ihrer Prozesskomplexität, eine Flexibilisierung des Unternehmens sowie eine stärkere Fokussierung auf das Kerngeschäft [Bitk 05].
War es früher für ein Unternehmen noch durchaus üblich, zum Beispiel einen eigenen E-Mail Server zu betreiben, haben heute viele Betriebe solche IT-Dienstleistungen auf externe Dienstleister (engl. *Provider*) ausgelagert. Zur Bereitstellung eines E-Mail Dienstes kommen oftmals weitere Leistungen hinzu, wie

- Breitbandinternetzugang
- die Möglichkeit eigene Inhalte zu veröffentlichen (engl. *Webhosting*),
- virtuelle private Netzwerke (VPN)
- Internettelefonie (VOIP - engl. *Voice over IP*)

Werden diese und weitere Dienste ausgelagert, bedeutet dies eine mehr oder minder starke Abhängigkeit vom externen Dienstleister. Diese Abhängigkeit kann je nach Branche und Art des ausgelagerten Dienstes die Handlungsfähigkeit der Firma unterschiedlich stark beeinflussen. Kommt zum Beispiel ein Handwerksbetrieb mit einem stundenweisen Ausfall seines Internetzugangs aller Voraussicht nach gut zurecht, bedeutet dasselbe Problem bei einer Werbeagentur, die fast ausschließlich online arbeitet und vielleicht sogar auf IP-Telefonie umgestiegen ist, eine nahezu vollständige Stilllegung des Betriebes.

Um diesen Abhängigkeiten wirksam entgegen zu treten, werden bestimmte Verträge zwischen Dienstleister und Kunde geschlossen. Man spricht hier von einer Dienstgütevereinbarung (engl. *Service Level Agreement* - SLA).

Service Level Agreement Eine gute Definition von solchen Verträgen liefert die *IT Infrastructure Library* (ITIL). Die ITIL ist ein Leitfaden zur Unterteilung der Funktionen und Organisation der Prozesse, die im Rahmen des serviceorientierten Betriebs einer IT-Infrastruktur eines Unternehmens entstehen. Hier wird ein SLA als zwischen dem Kunden und dem Dienstleister getroffene Vereinbarung hinsichtlich der zu leistenden Dienste gesehen. Der Vertrag beschreibt die Dienstleistungen in Begriffen ohne technischen Bezug und ist somit auch für technisch unerfahrene Kunden verständlich. Für die Dauer des Kontraktes gilt das SLA in Bezug auf Leistungserbringung und Steuerung des Dienstes [vB 02]. Mit dem Abschluss eines solchen Vertrages werden bestimmte Parameter einzelner Dienstleistungen festgelegt. Diese Dienstgüteparameter oder *Quality of Service Parameter* (QoS) können eine zugesicherte Verbindungsgeschwindigkeit oder auch die maximale Ausfallzeit bestimmter Dienste sein. Das SLA regelt auch, wie diese QoS Parameter objektiv zu messen sind und in welcher Form Supportleistungen erbracht werden müssen [oGC 00].
Die Nichteinhaltung dieser Parameter können zunächst Minderleistung und bei andauernden Vertragsbruch empfindliche Schadenersatzforderungen nach sich ziehen.

Verpflichtet durch das SLA muss der Provider ein gut organisiertes Fehlermanagement aufweisen können, um die Einhaltung der QoS-Parameter zu gewährleisten und die möglichen Folgen abzuwenden. Die Identifizierung und Behebung auftretender Probleme sollte dem Vertrag entsprechend schnell geschehen. Jene Aufgaben werden sogenannten Fehlermanagementsystemen übertragen. Diese setzten sich aus Komponenten zur Fehlersuche und -diagnose zusammen.
Bezogen sich bisherige Fehlermanagementsysteme fast ausschließlich auf die Verwaltung einer Netzinfrastruktur, also der diversen technischen Ressourcen wie Switches, Router oder auch Server, entstand mit der wachsenden Bedeutung von IT-Dienstleistungen der Bedarf an einer Erweiterung des Aufgabenbereichs auf

die angebotenen und im Vertrag festgelegten Dienstleistungen. Diese Erweiterung resultiert auch aus dem steigenden Gewicht, welches diese Dienste in SLAs einnehmen.

Um diesen Bedarf zu decken und möglichen empfindlichen Konsequenzen zu entgehen, ist es erforderlich Fehlermeldungen nicht nur von technischen Komponenten, sondern auch von Benutzern in die Prozesse des Fehlermanagemensystems einzubeziehen. Im Rahmen dieser Arbeit soll eine allgemeine Vorgehensweise aufgezeigt werden, wie solche Meldungen mit Ereignissen auf Infrastrukturebene verknüpft werden können, um eine gezielte Ursachenermittlung für eine Problem durchführen zu können. Diese Korrelation verbindet Unregelmäßigkeiten auf einer „höheren“ Dienstebene mit Fehlern auf Ressourcenebene. Die Bestimmung von Ursachen soll unter Beachtung der Abhängigkeiten zwischen den Diensten und der Infrastruktur erfolgen. Die Funktion einzelner Dienste hängt in der Regel nicht nur von einer Anzahl technischer Komponeten ab. So stützen sich Dienste auf weitere, untergeordnete Dienste, die wiederum mehrere Ressourcen zum Betrieb benötigen können.

Der E-Mail Dienst ist ein gutes Beispiel dafür, dass diese Abhängigkeiten komplexe Strukturen annehmen können: Eine Fehlermeldung eines Benutzers bezüglich seines E-Mail Kontos kann sich auf zahlreiche Subdienste und Hardwarekomponenten beziehen. Neben den Rechnern, die den Versand und den Empfang von Nachrichten durchführen, können neben Konnektivitätsdiensten und -ressourcen auch Namensauflösung oder Verzeichnisdienste beteiligt sein. Die Beschreibung des Szenarios am Leibnitz Rechenzentrum (LRZ) in 2.1 bestätigt diese Aussage.

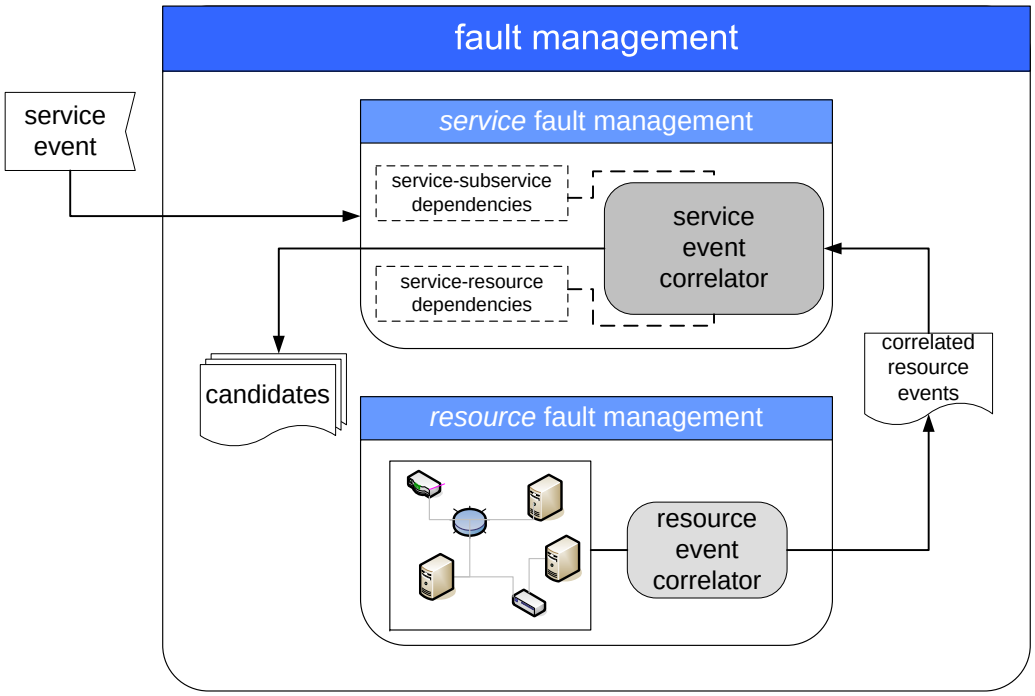


Abbildung 1.1: Ereigniskorrelation im dienstorientierten Fehlermanagement

Die zu entwickelnde Vorgehensweise besteht aus einem regelbasierten Mechanismus. Unter Einbeziehung der das Netz aber auch den jeweiligen Dienst betreffenden Abhängigkeiten und unter Einbezug vorliegender Fehlermeldungen von Benutzern soll eine Liste möglicher Fehlerquellen erstellt werden. Die Nutzung dieser Meldungen setzt die Bildung und das Betreiben einer Schnittstelle zum Kunden voraus. Dies ist nach [HA 05] eine weitere Aufgabe von Fehlermanagementsystemen. Abbildung 1.1 zeigt, wie im dienstorientierten Fehlermanagement mit Hilfe von Ereigniskorrelation Fehlerkandidaten ermittelt werden können.
Zwei wesentliche Komponenten sind hier dargestellt. Das Infrastrukturmanagement (*resource fault manage-*

ment) ist der Teil, der sich ausschließlich auf den Infrastrukturbereich konzentriert. Der *resource event correlator* korreliert Ereignisse, die von der Ressourcenebene stammen. Das Hauptaugenmerk dieser Arbeit liegt jedoch auf der zweiten Komponente, dem Dienstfehlermanagement (engl. *service fault management*), welches unter Berücksichtigung der vom *resource fault management* korrelierten Resource Events eingehende Service Events mit diesen verknüpft. Dazu nutzt der *service event correlator* die vorliegenden Abhängigkeitsstrukturen.
Als Ergebnis dieser Korrelation steht am Ende eine Liste möglicher Fehlerkandidaten (engl. *candidates*).

Provider können diese Erweiterungen nutzen, um eine stärkere Dienstorientierung in ihrem Fehlermanagement zu erreichen. Die Zweiteilung, wie in Abbildung 1.1 skizziert, erlaubt das Nutzen bewährter Systeme zur Ressourcenüberwachung.

1.1 Aufgabenstellung

Das Ziel dieser Diplomarbeit ist es, eine Vorgehensweise zu entwickeln, die beschreibt, wie *Service Events* und *Resource Events* unter Einbeziehung der Abhängigkeiten zwischen Diensten und Subdiensten sowie zwischen Diensten und der Infrastruktur korreliert werden können. Erfüllt werden sollen dabei die praxisbezogenen Anforderungen, welche in Kapitel 2 erarbeitet werden. Als Resultat der Ereigniskorrelation soll das System eine Liste möglicher Kandidaten für Fehlerquellen auf dem *resource level* (*candidates* in Abbildung 1.1) erstellen. Diese Auflistung kann dann benutzt werden, um die Identifizierung des Auslösegrundes, bzw. die fehlerhafte Ressource für das *Service Event* durchzuführen (engl. *root cause analysis*). Liegen für Ressourcen aus dieser Liste bereits Ereignisse vor, kann direkt eine Verknüpfung mit der Angabe der wahrscheinlichen Fehlerursache erzeugt werden.

Die von der Vorgehensweise gewonnenen Erkenntnisse sollen auf das kommerzielle Produkt *Tivoli Enterprise Console* von IBM angewendet werden. Diese Software ist ein typischer Vertreter von regelbasierten Fehlermanagementsystemen für Ressourcen. Eine genaue Betrachtung wird zeigen, dass nur die Implementierung eigener Regeln die Korrelation von *Service Events* möglich macht. Diese Regeln realisieren die oben erwähnte Erweiterung zur Einbeziehung von Diensten in das Fehlermanagement an einem konkretem Beispiel.

1.2 Vorgehensweise und Gliederung

Dieses erste Kapitel gibt eine grundsätzliche Einführung zum Thema. Zusätzlich werden im folgenden Abschnitt die wichtigsten Begriffe, die in der Arbeit verwendet werden, eingeführt und ausführlich erklärt.

Abbildung 1.2 zeigt die verwendete Vorgehensweise bei dieser Arbeit. Die Kästen sind die wichtigsten Arbeitsschritte und spiegeln teilweise Kapitel in diesem Dokument wider. Die Rauten stellen Ergebnisse dar, die in den jeweiligen Phasen entstehen. In den Kreisen werden alle externen Eingaben in die Arbeit dargestellt, wie zum Beispiel verwandte Arbeiten oder auch das Szenario.

Analyse Die Analysephase stellt die Anforderungen an die Arbeit heraus. Hilfsmittel für diese Phase, die in der Anforderungsanalyse in Kapitel 2 mündet, sind das Szenario, welches sich aus dem E-Mail und Webhostingdienst am LRZ zusammensetzt, sowie die Anforderungen, die sich aus dem regelbasierten Ansatz zur Ereigniskorrelation ergeben. Im Kapitel 3 werden die dazu vorhandenen Arbeiten dargestellt und anhand der Anforderungen bewertet. Dabei wird ein besonderes Augenmerk auf das Dienstmodell des *Munich Network Management* (MNM) Teams geworfen. Zusätzlich werden weitere unterschiedliche Ansätze zur Ereigniskorrelation präsentiert und ein kommerzielles Produkt vorgestellt.

Entwurf In der Entwurfsphase in Kapitel 4 soll ein Algorithmus entworfen werden, welcher unter der Berücksichtigung einer Abhängigkeitenmodellierung und mit Hilfe von festzulegenden Ereignisattributen eine *Root Cause Analysis* durchführen kann. Dabei sollen die in der Analysephase aufgestellten Anforderungen erfüllt werden.

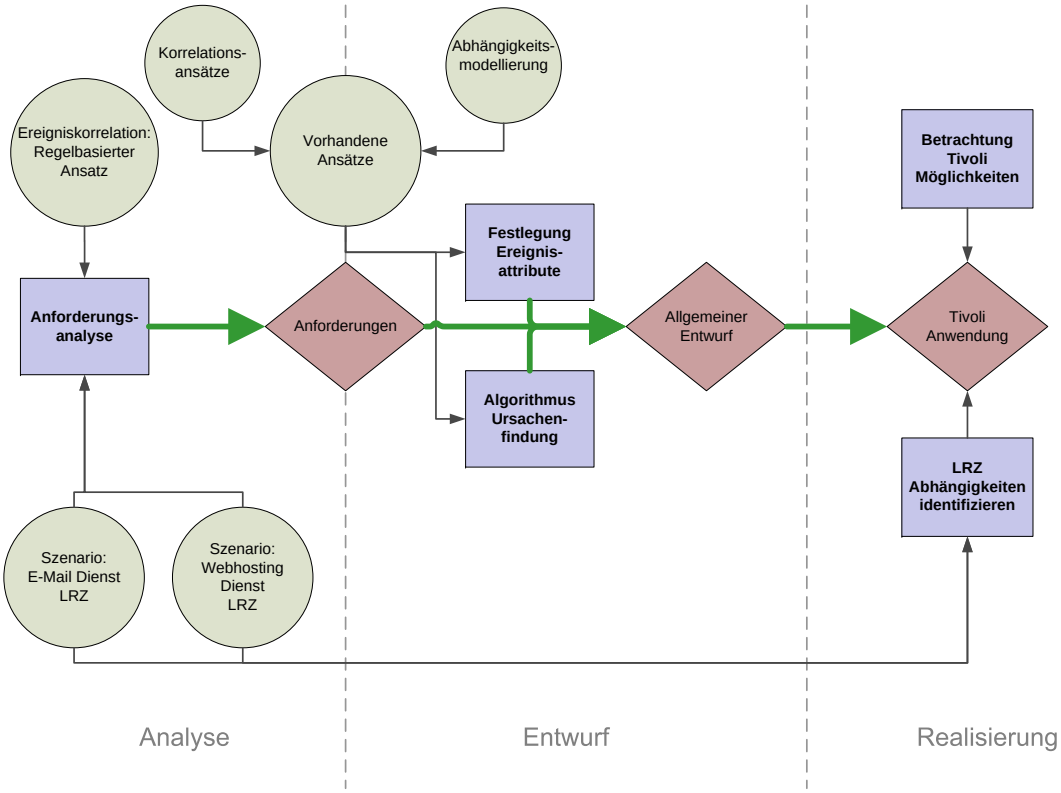


Abbildung 1.2: Vorgehensweise

Realisierung In der Realisierungsphase in Kapitel 5 sollen die in Entwurf und Analyse gewonnenen Erfahrungen genutzt werden, um eine dienstorientierte Ereigniskorrelation am Szenario zu testen. Zur Verwendung kommt ein kommerzielles Produkt, an welchem die Kenntnisse aus dem allgemeinem Entwurf angewendet werden.

Das 6. und letzte Kapitel bietet eine Zusammenfassung der Arbeit und der Ergebnisse, sowie einen Ausblick auf weitere, zukünftige Arbeiten.

1.3 Begriffsbestimmung

Da viele Begriffe Mehrfachbedeutungen besitzen oder aus dem Englischen ins Deutsche übernommen wurden, benötigen diese eine genauere Definition und Erklärung, um ihre Bedeutung im Kontext dieser Arbeit unmissverständlich klar zu stellen. Dabei soll von keinem spezifischen Szenario zum Fehlermanagement ausgegangen werden und eine einheitliche Terminologie innerhalb der Arbeit verwendet werden.

Diese Anforderung erfüllt das MNM-Dienstmodell. Das Modell definiert allgemein gebräuchliche Begriffe und Konzepte. Es kann auf die verschiedensten Szenarien angewendet werden und hilft, die notwendigen Akteure und deren Beziehungen untereinander zu identifizieren. Eine detailliertere Beschreibung findet sich in Abschnitt 3.2. Die folgenden Begriffe wurden soweit möglich aus den Definitionen dieses Modells entnommen.

Dienst (engl. *service*) Der Begriff „Dienst“ besitzt nicht nur im IT-Umfeld vielfältige Bedeutungen. Nach dem MNM-Dienstmodell definiert sich ein Dienst zunächst durch die angebotenen Funktionalitäten. Die konkrete Implementierung eines Dienstes muss nicht weiter betrachtet werden. Der Dienst muss von beiden Rollen (Kunde und Dienstleister) gleich verstanden werden. Nach [Gars 01] setzt sich ein Dienst weiterhin aus zwei Funktionalitätsklassen zusammen: Einer Nutzungs- und einer Managementsfunktionalität.

Service Event Ein *Service Event* ist ein Ereignis, welches den - möglicherweise nur teilweisen - Ausfall oder die Nichteinhaltung von festgelegten Dienstgüteparametern beschreibt. Quelle dieser Ereignisse sind entweder die Dienstnutzer (siehe unten) oder das Fehlermanagementsystem selbst, welches eigene Überprüfungen von Diensten durchführt. Beispiele für solche Ereignisse können sein

- Ausfall oder Nichterreichbarkeit eines Dienstes
- subjektiv und objektiv langsame Verbindungen
- lange Antwortzeiten
- fehlerhafte Darstellung von Webseiten

Bei Meldungen von Benutzern ist es notwendig daraus formalisierte *Service Events* zu erstellen. Das LRZ nutzt hier einen *Intelligent Assistant* (IA am LRZ).

Dienstleister (engl. *provider*) Ein Dienstleister bzw. ein Provider bietet in der Regel gegen Entgelt verschiedene technische Leistungen (Dienste, siehe oben) einer Reihe von Kunden an. Der *Provider* ist eine der beiden Rollen im MNM-Dienstmodell. Er erbringt Dienste in einer vertraglich (SLA) festgelegten Qualität. Es obliegt dem Provider diese Qualität zu jeder Zeit aufrecht zu erhalten, was ihn zu einem Management dieser Dienstleistungen verpflichtet.

Kunde (engl. *customer*) Mit Kunde sind sowohl Endverbraucher als auch Unternehmen gemeint, welche nicht in der Lage oder nicht gewillt sind, die Dienste selbst zu realisieren und sie deshalb auf einen externen Dienstleister (Provider) übertragen hat. Der Kunde ist der Empfänger der vom Provider erbrachten Dienstleistungen und eine weitere Rolle im MNM-Dienstmodell. Diese Rolle erfordert eine weitere Differenzierung in einen Kunden, also das Unternehmen, welches einen Dienst in Anspruch nimmt und einem Nutzer, der den Dienst im eigentlichen Sinne benutzt ([Gars 01]). Im Szenario (Kapitel 2.1) ist als beispielhafter Provider das

LRZ gewählt. Als Kunden fungieren unter anderen die Hochschulen LMU (Ludwig-Maximilians-Universität) und TUM (Technische Universität München). Der Nutzerkreis setzt sich aus den Studenten und Mitarbeitern der Hochschulen zusammen.

Nutzer (engl. *user*) Der Nutzer ist der Personenkreis, der einen Dienst tatsächlich in Anspruch nimmt. Bei einem Unternehmen, welches beispielsweise den E-Mail Dienst in die Verantwortung eines Providers gelegt hat, sind die Nutzer die Mitarbeiter, die täglich E-Mails versenden und empfangen.

Ressource Mit Ressource wird in dieser Arbeit jede Komponente bezeichnet, die zum Betrieb oder zur Funktion eines Dienstes notwendig ist. Solche Komponenten müssen nicht unbedingt auf Computer, Router oder Netzwerkleitungen, also der Hardware, beschränkt sein, sondern können durchaus auch Geschäftsprozesse, Programme / Prozesse, oder auch einfach Mitarbeiter sein.

Ressource Event Ein *Resource Event* ist ein Ereignis, welches in der Regel einen Zustandwechsel einer technischen Komponente einer Ressourceninfrastruktur beschreibt. Oft bieten einzelne Komponenten zahlreiche mögliche zutreffende Ereignisse an, die von einem Fehlermanagementsystem ausgewertet werden können. Solche *Resource Events* sind zum Beispiel

- Verbindungsabbrüche
- Erreichbarkeit von ganzen Komponenten oder Teilen (zum Beispiel bei Switches)
- Zustandswechel von Prozessen auf einem Rechner
- Auslastungsänderungen (beispielsweise Prozessorlast, Speicherplatz, Netzwerkauslastung)

Fehlermanagement Wie in der Einleitung dargestellt, kommen Provider nicht ohne Fehlermanagement aus. Damit ist die im Idealfall nahtlose Überwachung von angebotenen Dienstleistungen, entweder automatisiert durch Überwachungssysteme oder auch manuell gemeint. Man unterscheidet prinzipiell zwischen einem Fehlermanagement, welches sich auf Ressourcen konzentriert (engl. *resource level*), und einem, welches sich auf Dienste konzentriert (engl. *service level*).

Abhängigkeit In der Arbeit wird häufig von verschiedenen Abhängigkeiten die Rede sein. Abhängigkeiten können zwischen Diensten und Ressourcen, zwischen Diensten und weiteren Diensten aber auch nur unter Ressourcen auftreten. Eine Abhängigkeit zwischen zwei Komponenten besteht immer dann, wenn eine Komponente nicht oder nur teilweise ihre Funktion ohne die zweite aufrecht erhalten kann. Dabei sind nicht nur Abhängigkeiten möglich, die sich auf zwei Komponenten beschränken, sondern auch solche, die mehrere Komponenten einschließen, oder sogar Redundanzen, die eine komplexere Betrachtung nach sich ziehen.

Symptom Um eine Abgrenzung zu dem sehr allgemeinen Begriff „Fehler“ zu schaffen, wird der Begriff Symptom eingeführt. Ein Symptom ist das Auftreten einer Fehlfunktion eines Dienstes auf der Seite des Kunden bzw. des Nutzers. Ein Symptom ist nicht zwingendermaßen ein Fehler. Zum Beispiel muss eine vom Nutzer subjektiv als langsam empfundene Verbindung zu einem Webserver nicht unbedingt auf einen Fehler im Ressourcenbereich zurückzuführen sein.

Fehler Im Gegensatz zu einem Symptom ist ein Fehler ein tatsächlicher Missstand, der in der Regel auf der Providerseite auf Ressourcen- oder Dienstebene eintritt. Für einen Fehler in diesem Sinne gibt es folgerichtig auch immer einen Kandidaten des Ressourcenmanagements, der für die Ursache verantwortlich ist. Die Aufspürung dieser Ursache führt zu dem Begriff **Root Cause Analysis**.

2 Szenario und Anforderungsanalyse

Inhaltsverzeichnis

2.1 Das Szenario am LRZ 7
2.1.1 E-Mail Dienst am LRZ 7
2.1.2 Webhostingdienst am LRZ 9
2.1.3 Zusammenfassung 10
2.2 Verwendung des regelbasierten Ansatzes 10
2.2.1 Regelbasierte Systeme 11
2.2.2 Vorteile des regelbasierten Ansatzes 12
2.2.3 Nachteile des regelbasierten Ansatzes 13
2.3 Anforderungskatalog 13
2.3.1 Anforderungen aus der Verwendung eines regelbasierten Ansatzes 13
2.3.2 Anforderungen zur Filterung von Events 14
2.3.3 Anforderungen zur Dienstorientierung 14
2.3.4 Anforderungen an die Handhabung von Ressourcenfehlern 16

Die Anforderungsanalyse soll die wesentlichen Ansprüche an ein System zur regelbasierten, dienstorientierten Ereigniskorrelation feststellen. In der Analyse sollen zur besseren Illustration die Anforderungen anhand eines konkreten Szenarios entwickelt werden. Dabei dient die Korrelation von dienstbezogenen Ereignissen, welche den E-Mail und den Webhostingdienst des LRZ betreffen, als beispielhaftes Szenario (Abschnitt 2.1). Diese Analyse mündet in einen Katalog an Forderungen in Abschnitt 2.3. Weitere ergeben sich aus der Struktur regelbasierter Systeme (Abschnitt 2.2.1) und werden in Abschnitt 2.3.1 beschrieben.

2.1 Das Szenario am LRZ

Das Leibniz-Rechenzentrum (LRZ) ist gemeinsames Rechenzentrum der Ludwig Maximilians Universität München, der Technischen Universität München sowie der Bayerischen Akademie der Wissenschaften; es bedient auch die Fachhochschule München und die Fachhochschule Weihenstephan ([LRZ 06c]).
Das LRZ betreibt nicht nur E-Mail Server für die Studenten und Mitarbeiter der Münchner Universitäten, sondern stellt auch zahlreiche Webseiten den universitären Instituten und Forschungseinrichtungen zur Verfügung. Zudem tritt es für das Münchner Wissenschaftsnetz (MWN) als IT-Dienstleiter auf. Das MWN verbindet die Räumlichkeiten der LMU, der TUM, der Fachhochschule München und Weihenstephan, sowie der Bayerischen Akademie der Wissenschaften miteinander und umfasst derzeit ca. *60.000* Computer. Die Dienste des LRZ können so von allen Angehörigen der Münchner Hochschulen (ca. *85.000* Studenten, *7.000* wissenschaftliche und *14.000* sonstige Mitarbeiter) genutzt werden ([LRZ 06a] und [LRZ]).

2.1.1 E-Mail Dienst am LRZ

Der E-Mail Dienst ist eine der Kerndienstleistungen, welche das LRZ zentral für mehr als 100.000 Studenten und den Mitarbeitern der Münchner Universitäten anbietet. Von diesen 100.000 potenziellen Usern haben ca. 85.000 ein E-Mail Konto und 198 E-Mail Domains weisen auf das LRZ. Tabelle 2.1 zeigt einen Überblick über die Dimensionen des E-Mail Verkehrs am LRZ. Während die Studenten und Mitarbeiter der LMU Dienstnutzer sind, übernimmt zum Beispiel die LMU die Kundenrolle.

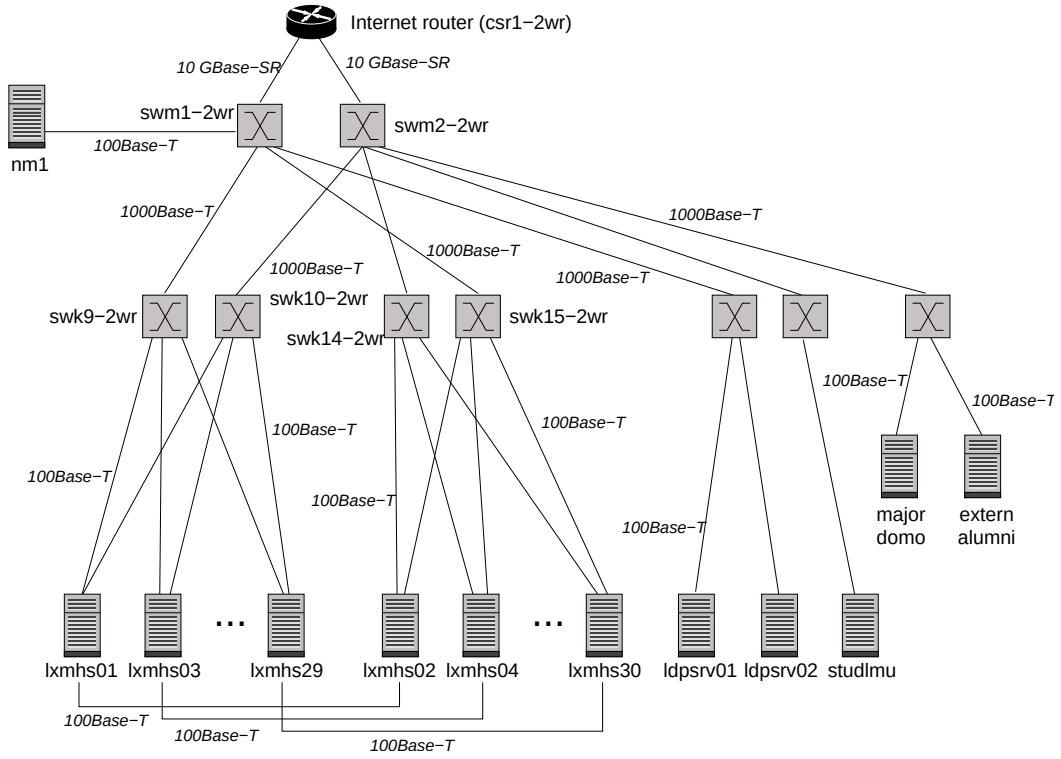


Abbildung 2.1: Ressourcen des E-Mail Dienstes am LRZ (aus [Hane 07])

	Mails gesamt	**akzeptiert**	**verzögert akzeptiert**	**zurückgewiesen**
werktags (außer Sa.)	1.000.000	180.000 (18%)	4.000 (2,2%)	820.000 (82%)
Sa./So./Feiertage	900.000	120.000 (13%)	1.000 (0,8%)	780.000 (87%)

Tabelle 2.1: Durchschnittliche Anzahl E-Mails pro Tag am LRZ im Jahresmittel ([LRZ])

Funktionalitäten und QoS Parameter Die Funktionalität des E-Mail Dienstes setzt sich im Grunde aus dem Empfangen und Weitergeben von E-Mails, die zuvor vom LRZ angenommen wurden, und dem Versenden von Nachrichten über das LRZ zusammen. Bei der E-Mail Funktionalität werden einige Restriktionen vorgenommen, die sich aus Sicherheitsüberlegungen ergeben. So ist zum Beispiel die maximale Größe einer E-Mail auf 30 MB begrenzt ([Stor 06]). Neben der Möglichkeit, Nachrichten über E-Mail Programme am eigenen Computer abzurufen, haben Nutzer auch über eine Webmailschnittstelle per Internetbrowser Zugriff auf ihr Postfach.

Managementfunktionalitäten sind zum Beispiel die Erstellung von neuen Postfächern oder die Konfiguration eines Spamfilters. Als QoS Parameter werden die Auslieferungszeiten für Nachrichten oder die allgemeine Verfügbarkeit angesehen. Zusätzlich wäre die Zeit für Fehlerbehebungen als Parameter mit einzubeziehen. Der Konjunktiv steht deswegen hier, da ein SLA in der weiter oben vorgestellten Form nicht existiert. Dennoch wird vom LRZ ein hohes Maß an Dienstqualität gefordert.

Realisierung Zur Realisierung des E-Mail Dienstes sind mehrere Subdienste nötig. Darunter fallen Speicherdienste zur Speicherung von eingehenden E-Mails und Konnektivitätsdienste für den Zugriff auf die einzelnen Mailserver. Zusätzlich werden DNS und der VPN Dienst, sowie SSH für die sichere Übertragung von E-Mails verwendet. Für den Zugriff auf das Postfach per Webmail Schnittstelle bestehen Abhängigkeiten zum Webhostingdienst, welcher für die Auslieferung der Portalseiten zuständig ist und zu einem Verzeichnisdienst. Eine Übersicht über die Ressourcen für den E-Mail Dienst ist in Abbildung 2.1 gegeben. Auf der Ressourcenebene verwendet das LRZ vier Mailserver, welche die Nachrichten für verschiedene Nutzergruppen emp-

fangen sowie zwei Mail-Relays, die über eine Lastverteilung verbunden sind und das Versenden der E-Mails übernehmen. In die Ressourceninfrastruktur mit einbezogen werden muss auch das Netzwerk, welches diese Komponenten miteinander verbindet.

2.1.2 Webhostingdienst am LRZ

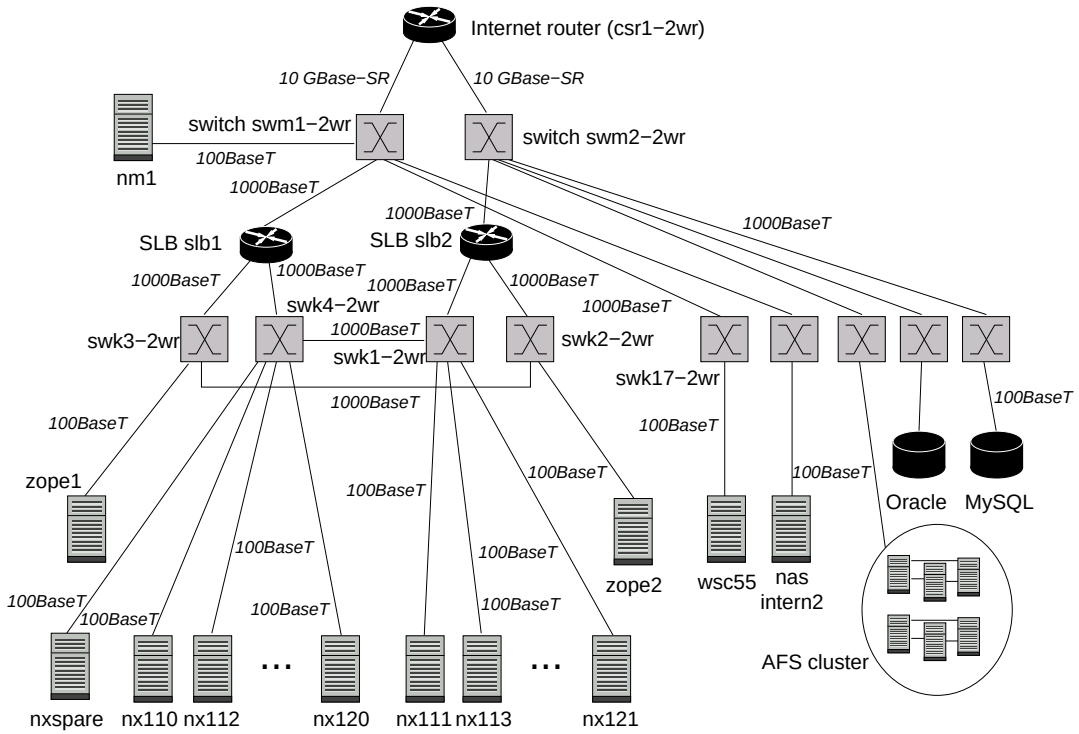


Abbildung 2.2: Infrastruktur zur Auslieferung von Webseiten am LRZ (aus [Hane 07])

Das LRZ hostet die Webseiten zahlreicher Institute, Studenten und Mitarbeiter. Das Bereitstellen von Internetseiten beschränkt sich längst nicht mehr auf das reine Anbieten von über das Internet zugänglichen Speicherplatz. Wie kommerzielle Anbieter auch ermöglicht das LRZ eine Menge an Zusatzoptionen, die mit den Webseiten genutzt werden können.
Der Aufwand, der getrieben werden muss um einen eigenen Rechner für jeden Kunden zu betreiben, ist sehr hoch. Das hat das LRZ veranlasst, sogenannte virtuelle WWW-Server einzurichten.
Diese Server sind insofern virtuell, als dass es sich dabei um eine interne Abbildung handelt, die den vom Kunden gewählten Domainnamen auf das vom LRZ bereit gestellte verteilte Dateisystem abbildet. Für den Benutzer ist diese Abbildung völlig transparent. Der Zugriff auf diese virtuelle Server unterscheidet sich nicht vom Zugriff auf einen realen Server. Derzeit nutzen ca. 350 Institutionen und Forschungsreinrichtungen dieses Angebot ([LRZ 06b]).

Funktionalitäten und QoS Parameter Das LRZ bietet folgende Funktionalitäten auf ihren virtuellen WWW-Servern an und übertrifft damit das Angebot vieler kommerzieller Anbieter:

- Jeder Benutzer hat insgesamt ein Gigabyte Speicherplatz zur Verfügung.
- Der Zugang zum Speicherplatz wird dem Kunden über FTP und SCP gewährt. Diese beiden Dienste garantieren einen sicheren und schnellen Transfer von Dateien vom eigenen Rechner auf den WWW-Server (Managementfunktionalität).

- Vorgefertigte CGI-Skripte, wie Gästebuch, Mini-Datenbank, E-Mail Formular usw. erlauben es dem Benutzer interaktive Elemente auf seiner Seite einzubinden.
- Zur verschlüsselten Übertragung von Inhalten stellt das LRZ eine SSL-Verschlüsselung mit einem LRZ Zertifikat zur Verfügung.
- Zur Auswertung der Besucherstatistiken können Benutzer täglich aktualisierte Zugriffslogs einsehen.
- Das Erstellen von dynamischen Internetseiten ermöglicht die Kombination aus der Skriptsprache PHP und einer MySQL Datenbank.
- Der gesamte Datenbestand wird täglich vom LRZ gesichert.
- Der Nutzer hat die Möglichkeit durch HTTPS oder Passwortschutz geschützte Bereiche (`htaccess`) auf seiner Webseite zu nutzen.

Die Nutzung dieser Dienste ist für wissenschaftliche Zwecke kostenlos. Erst mit der Beantragung einer eigenen Domain fallen Kosten an, die die Kunden tragen müssen. Für die Nutzerperspektive sind wiederum die Verfügbarkeit der Funktionalitäten und die durchschnittliche Ausfallzeit wichtige QoS Parameter, die Bestandteile eines SLA darstellen könnten.

Realisierung Auf der Ressourcenebene zeigt sich, wie in Abb. 2.2 illustriert, wie das LRZ seine Webserver an das Internet anbindet. Zunächst sorgen zwei redundante Load-Balancer, in der Abbildung als Level 4/7 Switches referenziert, für eine Verteilung der Anfragen auf die Webserver. Die Anbindung an das Internet übernimmt ein Internetrouter *csr1-2wr*. Ein Notfallserver *nxspare* kann bei einem schwerwiegenden Ausfall eine Benachrichtigungsseite darstellen. Im Regelbetrieb sind die Webserver über mehrere Switches mit dem Load-Balancer verbunden. Zusätzlich sind auf der Abbildung noch die angeschlossenen Datenbanken, sowie die verteilten Dateisysteme (NFS und AFS) zu sehen. Während statische Seiten nur über das AFS Dateisystem angesprochen werden, wird für die Auslieferung von dynamischen Webseiten der Zugriff auf das NFS Dateisystem und entsprechenden Datenbanken benötigt. Je zwei Webserver, welche über unterschiedliche Switches angebunden sind, bilden ein Paar, auf welchem die gleichen „Dämonen“ laufen. Des Weiteren existieren ein Server für die studentischen Webseiten und zwei Zope Anwendungsserver.

Abhängigkeiten bestehen auch zu den Subdiensten DNS und VPN/Proxy. Weiterhin können die Konnektivität oder auch der Zugriff auf Speicherkapazitäten (AFS / NFS) als Subdienste aufgefasst werden.

2.1.3 Zusammenfassung

Die in den bisherigen Abschnitten aufgeführten Werte zum Beispiel in Tabelle 2.1, die komplexen Strukturen der Ressourcen und die hohe Zahl an angebotenen Funktionnalitäten skizzieren das Szenario am LRZ. Die Anforderungen, die an ein Fehlermanagementsystem, dessen Aufgabe die Überwachung der korrekten und vollständigen Funktion dieser Dienste des Szenarios gestellt werden, lassen sich leicht auf weitere Szenarios, wie sie bei Providern existieren können, übertragen. Das LRZ verwendet nicht nur eine komplexe Hardwarestruktur, sondern auch zum Beispiel virtuelle Rechner zur Dienstrealisierung. Dieses Szenario kommt vielen anderen Unternehmen, die ähnliche Dienstleistungen anbieten, sehr nahe. Somit lassen sich die Schlüsse, die aus dieser Arbeit gezogen wurden, leicht auf deren Gegebenheiten übertragen.

2.2 Verwendung des regelbasierten Ansatzes

Zunächst wird in diesem Abschnitt der regelbasierte Ansatz zur Ereigniskorrelation kurz vorgestellt. Wie im Kapitel 3 über vorhandene Ansätze dargestellt wird, existieren mehrere Korrelationsmethoden, wie sie bereits bei der Korrelation von *Resource Events* nicht nur in kommerziellen Produkten eingesetzt werden. Zunächst erfordert der Begriff *Ereigniskorrelation* eine genauere Erklärung.

Ereigniskorrelation In Fehlermanagementsystemen versteht man unter Korrelation die Zusammenfassung und Bündelung von häufig in großer Anzahl in nur kurzer Zeit eintreffenden Ereignissen. Ein sehr einfaches Beispiel aus dem Ressourcenbereich ist in Abb. 2.3 zu sehen: Drei Server sind über einen Switch an

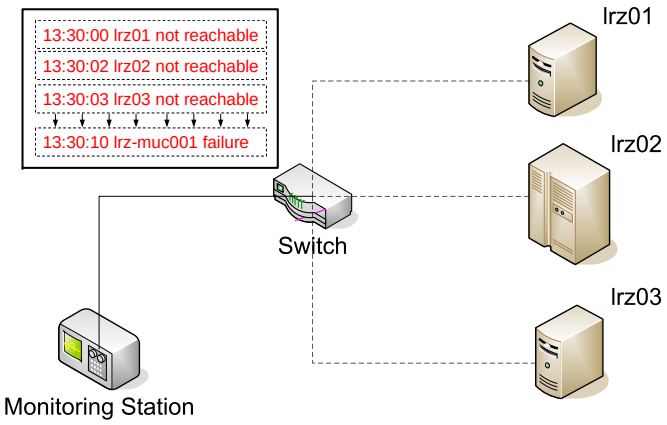


Abbildung 2.3: Korrelation

ein Netzwerk angeschlossen. Fällt nun auf Grund eines Fehlers diese Komponente aus, erkennt eine Überwachungsstation (Monitoring Station) die Nicht-Erreichbarkeit der drei Server. Anstatt jedoch diese drei Ereignisse an ein angeschlossenes Fehlermanagementsystem zu übermitteln, werden die Events korreliert und ein Ausfall des Switches an die angeschlossenen Systeme propagiert.

In der zukünftigen Forschungsarbeit [Hane 07], welche einen Ansatz des MNM Teams am LRZ beschreibt, wird ein Hybridmodell zur dienstorientierten Ereigniskorrelation vorgestellt. Dieses Modell beschreibt eine Lösung, die sich aus einer Kombination eines regelbasierten und eines fallbasierten Systems zusammensetzt.
Die vorliegende Arbeit konzentriert sich auf den regelbasierten Teil des Korrelationssystems. Der regelbasierte Ansatz bietet wie jeder andere Ansatz eine Reihe von Vor- und Nachteilen. Die Anforderungsanalyse soll sicherstellen, dass die Potentiale genutzt werden und dabei die Restriktionen so gut wie möglich ausgeglichen werden. Zusätzlich müssen noch weitere Anforderungen an das System gestellt werden, die teilweise einen direkten Zusammenhang mit der gegenwärtigen Praxis aufweisen können.
Der entscheidende Unterschied zwischen regelbasierten Ansätzen, wie sie in kommerziellen Produkten verwendet werden und dem hier zu entwickelnden Ansatz ist die deutliche Dienstorientierung dieses Ansatzes.

2.2.1 Regelbasierte Systeme

Regelsysteme sind die bekanntesten Vertreter wissensbasierter Systeme ([AJG 93]). Ein solches regelbasiertes System ist durch die grundsätzlichen Elemente wie in Abbildung 2.4 skizziert, gekennzeichnet. Der Informationsfluss zwischen diesen Elementen ist durch Pfeile illustriert.

- Eine Menge von Fakten, auch als **Wissensbasis** referenziert, speichert alle bekannten Informationen über das System. Auf die dienstorientierte Ereigniskorrelation bezogen ist die Wissensbasis das Wissen über alle möglichen Abhängigkeiten, die zwischen Diensten und Subdiensten und zwischen Diensten und Ressourcen bestehen können.

- Regeln bilden die **Interferenzkomponente** des Systems. Diese Regeln liegen in der Form
  `if <condition> then <conclusion>`
  vor, welche auch *Event Condition Action* genannt wird ([Hane 07]). Dabei steht *condition* für die mit logischen Junktoren geformte Bedigung, die erfüllt werden muss und *conclusion* für den Schlussfolgerung, die gezogen wird. Die Regeln steuern die Ereigniskorrelation und stoßen entsprechende Aktionen an. Dieser Algorithmus wird oft als Interferenzkomponente bezeichnet.

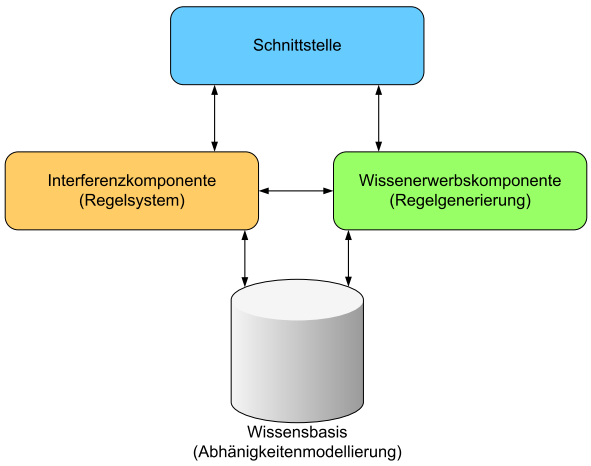


Abbildung 2.4: Elemente regelbasierter Systeme

- Die **Wissenererwbskomponente** ist die Komponente zur Generierung von neuen Regeln. Die Erstellung neuer Fakten kann durch einen Automatismus oder durch Eingabe erfolgen.
- Zur Kommunikation wird eine **Schnittstelle** zu den restlichen Komponenten des Rahmenwerks benötigt.

Im Folgenden werden verschiedene Vor- und Nachteile eines regelbasierten Ansatzes zur Ereigniskorrelation dargestellt und erläutert. Aus diesen Punkten können dann entsprechende Anforderungen an das System abgeleitet werden, die sich entweder aus der Nutzung der angebrachten Vorteile oder aus der Vermeidung der Nachteile ergeben.

2.2.2 Vorteile des regelbasierten Ansatzes

Einfachheit Regelbasierte Systeme sind einfach und intuitiv. Dieser Fakt resultiert aus dem Umstand, dass Regeln immer in der gleichen Form `IF-THEN` vorliegen. Diese Verwandschaft zu natürlicher Sprache erlaubt ein schnelles Verständnis der Regeln und somit auch eine einfachere und schnellere Erstellung dieser. Dabei bleibt die Einfachheit auf die Struktur der Regeln beschränkt: Die Bedingungen für bestimmte Aktionen können ein hohes Maß an Komplexität erreichen.

Hohe Performanz Gerade bei der Ereigniskorrelation kommt es oft zu großen Mengen von Ereignissen innerhalb kürzester Zeit (engl. *event burst*). Dieses Phänomen ist zum Beispiel zu beobachten, wenn eine wichtige Netzverbindung gekappt wird oder ein sehr grundlegender Dienst ausfällt, auf den andere aufsetzen. Innerhalb weniger Sekunden werden Systeme zur Fehlerüberwachung von allen betroffenen Komponenten bzw. Diensten Fehler melden. Ein Korrelationssystem muss diese event bursts möglichst schnell abarbeiten können. Regelbasierte Systeme sind für diesen Zweck gut geeignet ([Hane 06]).
Viele dieser Systeme verwenden einen effizienten Algorithmus, beziehungsweise Varianten davon, wie er von [Forg 82] bereits im Jahre 1982 vorgestellt wurde.

Fehlerfreiheit Wissensbasierte Systeme arbeiten in hohem Maße zuverlässig. Dieser Vorteil wiegt besonders schwer, wenn man den Vergleich zwischen manuellem Korrelieren von Ereignissen und einem automatisierten Ablauf zieht. Muss zum Beispiel ein Mitarbeiter die Ursache für eine Fehlermeldung finden, sollte er sehr genaue Kenntnisse über die vorhandenen Ressourcen und Dienste und die bestehenden Abhängigkeiten haben, um ein korrektes Ergebnis zu erzielen. Dass er dabei Fehler macht, ist menschlich und wahrscheinlich.

Modularität Jede Regel ist diskret innerhalb einer Wissesbasis und kann daher ohne Berücksichtigung technischer Einzelheiten hinzugefügt oder entfernt werden - natürlich nur solange keine weitere Regel dadurch be-

einflusst wird. Diese Eigenschaft garantiert Flexibilität während der Entwicklungszeit, da eine inkrementelle Erweiterung der Regeln möglich ist.

2.2.3 Nachteile des regelbasierten Ansatzes

Bei der Verwendung von regelbasierten Sytemen zur Ereigniskorrelation müssen auch Nachteile in Kauf genommen werden.

Schwierige Wartbarkeit Kundenwünsche erfordern von Providern oftmals eine sehr dynamische Anpassung ihrer angebotenen Dienstleistungen. Nicht nur Änderungen innerhalb der Dienste, sondern auch an der Konfiguration der zu Grunde liegenden Ressourcen sind die Folge ([Hane 06]). Im oben dargestellten Szenario ist so durchaus möglich, dass zum Beispiel die Einführung eines zentralen Verzeichnisdienstes mitunter gravierende Änderungen an den angebotenen Diensten bewirkt.
Um dieser Dynamik gerecht zu werden, muss ein Korrelationssystem schnell an neue Bedingungen angepasst werden können. Regelbasierten Systemen ist so eine Anpassung nur durch Änderungen am Regelsatz möglich. Sind mehrere Regeln miteinander verknüpft, kann dies zu einem massiven Eingriff führen, der mit einem hohen Zeit-, Arbeits- und Fehleraufwand behaftet ist. Um dem entgegen zu treten, muss das Fehlermanagementsystem unabhängig zu einem Abhängigkeitenmodell operieren können. Eine andere Option wäre die dynamische Erstellung von Regeln, wenn eine Umstrukturierung auf Dienst- oder Ressourcenebene eine Neukonfiguration des Regelsatzes erfordert.

Fehlende Robustheit Mit zunehmender Komplexität der Abhängigkeiten zwischen Diensten, Subdiensten und Ressourcen steigt die Wahrscheinlichkeit, dass Ereignisse auftreten, die nicht in einem Regelsystem erfasst sind und deshalb nicht korreliert werden können. Regelbasierte Systeme stehen unbekannten Situationen, die nicht mit vorhandenen Regeln abgearbeitet werden können, hilflos gegenüber. Ein ähnliches Problem kann auftreten, wenn nur eine der notwendigen Bedingungen für eine Regel nicht erfüllt wird.
Dieser fehlenden Robustheit versucht man mit einem Hybridansatz ([JG 04]) weitestgehend zu kompensieren. In der Praxis müssen solche Vorfälle in der Regel von Mitarbeitern bewältigt werden, was oft mit hohen Zeit- und Kostenaufwand verbunden ist.

2.3 Anforderungskatalog

Der Anforderungskatalog listet alle Anforderungen an ein System zur regelbasierten Ereigniskorrelation auf. Die Anforderungen aus dem regelbasierten Ansatz werden im Folgenden zusammengefasst.
Anschließend werden weitere Anforderungen unter unterschiedlichen Gesichtspunkten gruppiert und aufgelistet.

2.3.1 Anforderungen aus der Verwendung eines regelbasierten Ansatzes

Wie oben aufgezeigt, ist die klare Struktur der Regeln ein wichtiges Merkmal regelbasierter Systeme. Ein System zur Ereigniskorrelation sollte diesen Vorteil zu nutzen wissen, indem diese Struktur so weit wie möglich beibehalten wird. Wie schon erwähnt, können die Bedingungen für Regeln ein hohes Maß an Komplexität erreichen. Eine Anforderung ist, diese Komplexität möglichst zu begrenzen, um diesen Vorteil nicht aufzuheben. Die Schwierigkeit dabei ist, dass sich die Bedingungen oft aus dem Abhängigkeitenmodell heraus ergeben und somit nur wenig Handlungsspielraum bieten. Eingriffsmöglichkeiten sind also nur im Abhängigkeitenmodell gegeben. Nur diejenigen Zusammenhänge sind in die Regel mit einzubeziehen, die auch wirklich nötig sind, um einen Fall richtig abzubilden. Um die Modularität regelbasierter System ist nicht zu relativieren, sollten Regeln nicht zu stark voneinander abhängen.

2.3.2 Anforderungen zur Filterung von Events

In einer komplexen IT-Umgebung generieren zahlreiche Ressourcen, die von einem Managementsystem überwacht werden, eine Vielzahl von diskreten Events. Zum einen kann diese Menge an Events nicht ohne einen vertretbaren Aufwand korreliert werden und zum anderen tragen zum Beispiel mehrere Überwachungsinstanzen auf derselben Ressource zur Generierung von redundanten Events bei. In Produkten zur Ereigniskorrelation, die sich auf die Ressourcenebene beschränken, finden sich ausgeprägte Filtermechanismen. Diese Anforderungen an Filterungsmöglichkeiten muss auch ein System zur dienstorientierten Ereigniskorrelation erfüllen.

Priorisierung von Ereignissen Der Umgang mit Ereignissen, die Dienste, welche von Kunden oder Nutzern verwendet werden, betreffen können, sollte eine sich von der Behandlung ressourcenbezogener Events unterscheidende Vorgehensweise aufweisen. Der Grund liegt im Ursprung des Ereignisses. Werden Ressourcen in der Regel automatisch zum Beispiel mit Hilfe des *Simple Network Management Protocol* (SNMP) überwacht und entsprechend Events automatisiert generiert, kommen Meldungen über fehlerhafte Dienste häufig von den Kunden oder Benutzern. Die Einhaltung von SLAs oder auch die Unternehmenspolitik fordern in diesem Fall in der Regel eine möglichst schnelle Reaktion der Supportabteilung.
Auch wenn das LRZ keinen SLA in diesem Sinne mit den universitären Nutzern eingegangen ist, ist es dennoch anzustreben, Nutzermeldungen möglichst schnell zu bearbeiten, um den reibungslosen Betrieb aufrecht zu erhalten. So behandelt das LRZ solche Meldungen ebenfalls mit weitaus höherer Priorität, als automatisch generierte.

Zeitproblematik Oft ist es bei der Ereignisverarbeitung notwendig, den Faktor Zeit mit einzubeziehen. Für die Korrelation kann es nicht nur entscheidend sein, wann ein Ereignis aufgetreten ist, sondern auch wieviele bestimmte Ereignisse in einem bestimmten Zeitraum eintreffen. Wie auch bei der Korrelation von Ressourcenevents muss das System auf eine Vielzahl von Ereignissen innerhalb kürzester Zeit vorbereitet sein. Zur Filterung kann dann der Zeitstempel von Events mit einbezogen werden.

Eine Situation aus der Praxis ist beispielsweise, dass die Suche nach einer Fehlerursache erst beginnt, wenn mehrere Nutzermeldungen zu einem Dienst eingetroffen sind. Wie aus Gesprächen mit Mitarbeitern hervorgeht, liegt der Grund hierfür darin, dass gerade bei häufig genutzten Diensten auch viele „Fehlalarme“ gemeldet werden, deren Überprüfung einen nicht unerheblichen Aufwand darstellt.

2.3.3 Anforderungen zur Dienstorientierung

Für die Korrelation von dienstorientierten Ereignissen müssen einige gesonderte Anforderungen gestellt werden.

Dynamische Ressourcen- und Dienststruktur Die Ressourcen- und Dienststruktur eines Providers darf in der Regel nicht als statisch angenommen werden, sondern unterliegt einem Änderungsprozess. Dieser Prozess resultiert unter anderen aus Modifikationen an Ressourcen, Redundanzen oder aus der Hinzunahme oder dem Wegfallen von angebotenen Diensten. Diese Veränderungen wirken sich auch auf die Abhängigkeitenmodellierung aus. Sollen Regeln aus dieser Abhängigkeitenmodellierung heraus gewonnen werden, müssen diese automatisch auf Änderungen in der Modellierung reagieren. Die Ableitung dieser Regeln sollte ohne menschliches Zutun von dem System selbstständig erledigt werden.
Am LRZ leitet sich diese Anforderung aus der Tatsache ab, dass immer wieder neue Dienste angeboten werden bzw. andere wegfallen. Zudem finden regelmäßig Anpassungen von Ressourcen statt - beispielsweise durch Austausch von veralteten Komponenten.

Messen der Dienstgüteparameter Wie bereits in der Einleitung erwähnt, spielen *Quality of Service* Parameter eine entscheidende Rolle beim Abschluss von SLAs. Um dem Provider die präzise Einhaltung von diesen Verträgen zu ermöglichen, erfordert dies einerseits eine genaue Modellierung dieser QoS Parameter

in Bezug auf bestimmte, angebotene Dienste und Funktionalitäten. Andererseits muss ein Überwachungssystem die Einhaltung dieser Werte kontrollieren. Dazu ist es auch erforderlich, dass die Information über eine Überschreitung von QoS Parametern dem System bekannt gegeben wird.

Active Probing Unterschiedliche Dienste werden auch von unterschiedlich großen Kundenkreisen genutzt. Wirft man einen Blick auf das LRZ - Szenario, so wird klar, dass die Funktionalität, neue E-Mails vom Mailserver abzuholen, sicherlich eine sehr häufig genutzte Funktion ist. Hingegen wird beispielsweise das Ändern eines Passwortes für eine Mailbox, welches ja auch eine Funktionalität des E-Mail Dienstes ist, nur sporadisch genutzt. Entsprechend unwahrscheinlicher ist, dass für diesen Dienst Kundenmeldungen eintreffen, wenn eine Fehlfunktion oder ein Symptom vorliegt. Um dem entgegen zu wirken, muss es einen Mechanismus geben, der eine aktive Messung der Dienstgüteparameter veranlasst, damit das System Kenntnis über den Dienststatus hat. Ein anderer Grund für dieses *Active Probing* ist, dass für die Korrelation die Status einzelner Dienste bekannt sein müssen, um ein sinnvolles Korrelationsergebnis zu erzielen.
Active Probing darf jedoch nicht auf Dienste beschränkt sein. Ist der Status von Ressourcen unbekannt, so kann eine aktive Anforderung einer Überprüfung einzelner Komponenten hilfreich sein, die Ursache für einen Fehler zu finden. Dies darf jedoch nicht ohne Einschränkung gemacht werden. Umfangreiche, automatische Überprüfungsroutinen können leicht zu künstlich hervorgerufenen Verzögerungen in der gesamten Infrastruktur führen. Möchte man kosteneffektiv handeln, ist es nach [MB 03] notwendig, mit einer kleinen Anzahl an Tests eine hohe Genauigkeit zu erlangen.

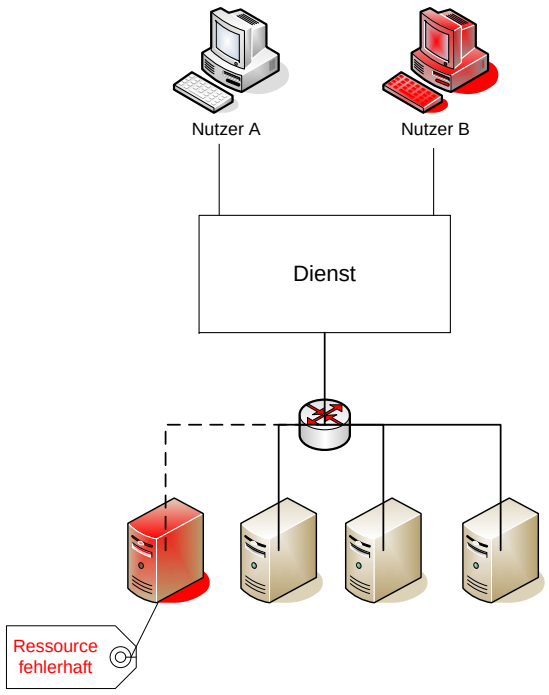


Abbildung 2.5: Dienstinstanzen

Dienstinstanzen Verwenden Dienste mehrere, unter Umständen redundante Ressourcen, so besteht die Möglichkeit, dass unterschiedliche Anwender bei der Nutzung der Dienstfunktionalitäten auf unterschiedliche Ressourcen geleitet werden können (siehe zu dieser Problematik auch Abbildung 2.5). Nutzer A wurde auf eine fehlerhafte Ressource weitergeleitet, während Nutzer B den Dienst fehlerfrei nutzen kann. Meldet der Nutzer A jetzt ein *Service Event* mit diesem Symptom, kann es für das Fehlermanagement schwierig werden, diesen speziellen Fall nachzuvollziehen. Dabei kann dieses Routing auf unterschiedliche Ressourcen für den Nutzer auch völlig transparent von statten gehen. Die Anforderung ist, dass solche Fälle erkannt werden können.

Dienstüberwachung von Drittanbietern Wie bereits weiter oben erwähnt, verlassen sich Dienstanbieter ihrerseits immer wieder auf die Dienste von Drittanbietern. Normalerweise ist es nicht möglich auf die Struktur eines solchen Drittanbieters zuzugreifen und ein Provider ist auf die Funktion des fremden Dienstes angewiesen. Ein System zur Ereigniskorrelation muss auch diesen Umstand bewältigen können und bei einer Fehlfunktion des Dienstes des externen Anbieters diesen als Fehlerursache erkennen könnnen.
Auch das LRZ steht bei einigen fundamentalen Diensten in Abhängigkeit zu Drittanbietern, auf deren Zuverlässigkeit es angewiesen ist. Das LRZ nutzt beispielsweise ein Backbone des kommerziellen Anbieters M-Net ([MN 06]).

2.3.4 Anforderungen an die Handhabung von Ressourcenfehlern

Multiple Ressourcenfehler Bei der dienstorientierten Ereigniskorrelation auf Dienstebene darf nicht von einer singulären Ursache einer Störung ausgegangen werden. Es kann nicht davon ausgegangen werden, dass immer nur eine Ressource zu einer Störung eines Dienstes (also zu einem Symptom) führt. Ein Beispiel ist es, wenn der E-Mail Empfang nur sehr langsam reagiert. Die Ursache dafür könnte das parallele Auftreten einer hohen Auslastung eine Netzwerkverbindung und eines Rechners sein. Nur beide Ereignisse zusammen führen zu dem Problem beim übergeordneten Dienst. Ein weiteres Beispiel für multiple Ressourcenfehler sind Ereignisse, die nach dem Ausfall der Stromversorgung auftreten können. In so einem Fall sind typischerweise zahlreiche Komponenten gleichzeitig betroffen. Ziel muss es sein, solche Mehrfachfehler erkennen zu können.

Ressourcenredundanzen Wie Abbildung 2.2 zeigt, basieren zahlreiche Dienste am LRZ auf einer redundanten Ressourceninfrastruktur. Es wird so versucht, sich unabhängiger von Ausfällen einzelner Ressourcen zu machen. Für die Fehlersuche bedeutet dies, dass unterschiedliche Dienste unterschiedlich viele redundante Ressourcen benötigen, um die Funktion gemäß SLA aufrecht zu erhalten. Ein weiteres Problem kann sich ergeben, wenn Nutzer auf eine (redundant vorhandene) Ressource weitergeleitet werden, die ausgefallen ist, während die anderen Nutzer auf funktionierende Ressourcen geroutet werden. Auch ist es möglich, dass Dienste nach Ausfall einer oder mehrerer redundanter Ressourcen ihre Dienstgüteparameter nicht mehr in vollem Umfang erfüllen können bzw. eine gewisse Mindestanzahl an Ressourcen für den Betrieb benötigen. Ein Korrelationssystem muss auch solche Fehler innerhalb dieser Ressourcen handhaben können.

Erstellung einer Kandidatenliste Zum Ende eines Korrelationsvorgangs sollte eine Liste von möglichen Fehlerursachen vorliegen, die es dem Servicepersonal ermöglicht, gezielt nach einer Störungsursache (oder den Störungsursachen) zu suchen. Diese Liste sogenannter Kandidaten muss sich nicht immer ausschließlich auf Ressourcen beziehen, sondern kann wie im Punkt „Dienstüberwachung von Drittanbietern“ beschrieben auch Dienste beinhalten. Bei der Erstellung dieser Kandidatenliste sollte darauf geachtet werden, dass die Anzahl der möglichen Fehlerursachen überschaubar bleibt, um eine schnelle und gezielte Fehlerbehebung durch das Servicepersonal zu gewährleisten.
Mit Hilfe von Erfahrung und Wissen über die Infrastruktur obliegt am LRZ die Erstellung dieser Liste derzeit den fachkundigen Mitarbeitern. Wurde diese erst einmal erstellt, können die Fachleute dann die einzelnen Ressourcen auf Funktion überprüfen. Ein Korrelationssystem hingegen sollte diese Liste automatisch generieren.

3 Vorhandene Ansätze

Inhaltsverzeichnis

3.1 ITIL 17
3.2 MNM Dienstmodell 18
3.2.1 Vorstellung 18
3.2.2 Zusammenfassung 21
3.3 Verfeinerung des MNM Dienstmodells - Abhängigkeitenmodellierung 21
3.3.1 Modellierung der Abhängigkeiten 21
3.3.2 Zusammenfassung 23
3.4 Dienstorientierte Ereigniskorrelation 23
3.5 Produkte 24
3.5.1 HP OpenView 24
3.5.2 IBM Tivoli Enterprise Console 24
3.5.2.1 Beschreibung 24
3.5.2.2 Konzepte 25
3.5.2.3 Bewertung 27

Dieses Kapitel stellt eine Übersicht über vorhandene Ansätze zum dienstorientiertem Fehlermanagement dar und bewertet diese Arbeiten.

Zunächst werden in 3.1 die Arbeiten der ITIL vorgestellt, die dazu geeignet sind diese Arbeit in einen Gesamtablauf eines Unternehmens einzuordnen, welches IT-Dienste anbietet und ein entsprechendes Management dieser betreiben möchte.
Das MNM-Dienstmodell, entwickelt an den Münchner Hochschulen LMU und TUM, wurde bereits in der Begriffsbestimmung in Abschnitt 1.3 eingeführt und soll nun in 3.2 detaillierter vorgestellt werden. Es hilft dabei, dienstorientiertes Fehlermanagement und die Aufgaben der Ereigniskorrelation einzuordnen.
Eine kurze Vorstellung eines Hybridansatzes zur Ereigniskorrelation wird in Abschnitt 3.4 gegeben.
Eine besondere Stellung nimmt eine Arbeit über Abhängigkeitsmodellierung ein, welche die Grundlage für die Entwicklung eines Regelsystems ist.
Kommerzielle Systeme, welche derzeit auch am LRZ zur Ereigniskorrelation eingesetzt werden, werden am Ende dieses Kapitels in 3.5.2 kurz dargestellt.

3.1 ITIL

Mit Prozessen beschreibt ITIL vor allem die im Rahmen des IT-Service Managements notwendigen Maßnahmen und Aktivitäten für die Lieferung von IT-Services in der gewünschten Qualität. Ursprünglich war ITIL ein Produkt der *Central Computer and Telecommunications Agency* (CCTA). Die CCTA war eine Organisation der britischen Regierung. Seit 2001 ist die CCTA jedoch in das *Office of Government Commerce* (OGC) übergegangen, welches somit neuer Eigentümer der ITIL wurde ([vB 02]).

In sieben Kernpublikationen („Bücher“) beschreibt ITIL wesentliche Prozesse des IT Service Managements. Dazu zählen insbesondere

- Service Support
- Service Delivery
- Service Management

- Infrastructure Management

Diese, auch als Module bezeichneten Komponenten, beschreiben die jeweiligen Prozesse, Rollen und Zuständigkeiten, sowie die notwendigen Datenbanken und Schnittstellen. Dabei erfolgen diese Beschreibungen auf einem hohen Abstraktionsniveau, das keine konkreten Rückschlüsse auf bestimmte Produkte oder Architekturen zulässt. Für diese Arbeit relevant ist insbesondere das Modul *Service Support*, welches die Provider internen Vorgehensweisen bei der Dienstbereitstellung aufführt. Ein Teil dieses Moduls ist auch das Fehlermanagement (*Fault Management*), welches wiederum in den *Incident Management Process* und den *Problem Management Process* zersplittet ist. Während ersterer den Umgang mit Problemmeldungen (*Incidents*) von Nutzern beschreibt, wird in Zweiterem die Suche nach Ursachenproblemen erläutert.

Der *Incident Management Process* stellt den vollständigen Ablauf, den ein *Incident* in diesen Prozessen durchläuft, dar. Unter anderem schlägt ITIL eine Vielzahl von Attributen vor, die dem folgenden *Problem Management Process* helfen sollen, möglichst zeitnah die Ursache für ein Problem zu finden. Nach der initialen Behandlung einer Nutzermeldung, führt dieser unter Zuhilfenahme einer Datenbank und anderen beteiligten Modulen (zum Beispiel dem *Change Management Process*) die Suche nach einer Ursache durch ([oGC 00]).

Diese Publikationen beschreiben auf einem hohem Abstraktionsniveau die notwendigen Schritte des IT-Service- und Fehlermanagements. Fehlende, detaillierte Angaben zu Techniken oder Architekturen werden kompensiert durch einen Maßnahmenkatalog, der sicherstellt, dass keine wichtigen Aspekte unbeachtet bleiben. Dieser Katalog beinhaltet Ziele, die durch die Anwendung der in dieser Arbeit vorgestellten Vorgehensweise zur dienstorientierten Ereigniskorrelation erfüllt werden können. Ebenso wird die Einordnung der Arbeit in ein umfangreicheres System ermöglicht.

3.2 MNM Dienstmodell

3.2.1 Vorstellung

Das MNM-Dienstmodell, wie in [Gars 01] beschrieben, ist ein Dienstmodell, welches gebräuchliche Begriffe, Konzepte und Strukturregeln eindeutig definiert und festlegt, um Probleme beim IT Service Management zu adressieren. Das Modell kann auf eine Vielzahl an möglichen Szenarien angewendet werden und hilft, die notwendigen Akteure und deren Beziehungen untereinander zu analysieren, identifizieren und zu strukturieren. Da es die gesamte Lebensdauer eines Dienstes berücksichtigt und unterstützt, fördert es den Informationsfluss zwischen Organisationen und Geschäftsteilen zu etablieren, verstärken und optimieren. Zusammenfassend erfüllt das MNM Dienstmodell unter anderem diese Anforderungen:

- *Generische und abstrakte Dienstdefinition*
  Eine abstrakte Definition des Dienstbegriffes hilft Dienste allgemein zu beschreiben und von Szenarien unabhängig zu verstehen. Das Modell kann auf alle Arten von Diensten angewendet werden, egal ob Kommunikationsdienste oder auch verteilte Anwendungen. Diese Eigenschaft wurde bereits in der Einleitung für eine einheitliche Begriffsbestimmung genutzt.

- *Trennung von Dienst und Dienstrealisierung*
  Diese Trennung erlaubt es Dienstanbietern, sich an lokale Gegebenheiten anzupassen und sich nicht auf eine bestimmte Implementierung eines Dienstes festlegen zu müssen.

- *Integriertes Management*
  Management ist im Modell ein integraler Bestandteil eines Dienstes. Rollen und Wechselbeziehungen während der gesamten Lebensdauer eines Dienstes sollen die Abdeckung aller funktionaler Managementaspekte garantieren.

Folgende Merkmale kennzeichnen das MNM Dienstmodell:

- *Lebenszyklus*
  Das MNM Dienstmodell unterteilt den Lebenszyklus eines Dienstes in fünf Phasen (siehe Abbildung 3.1).

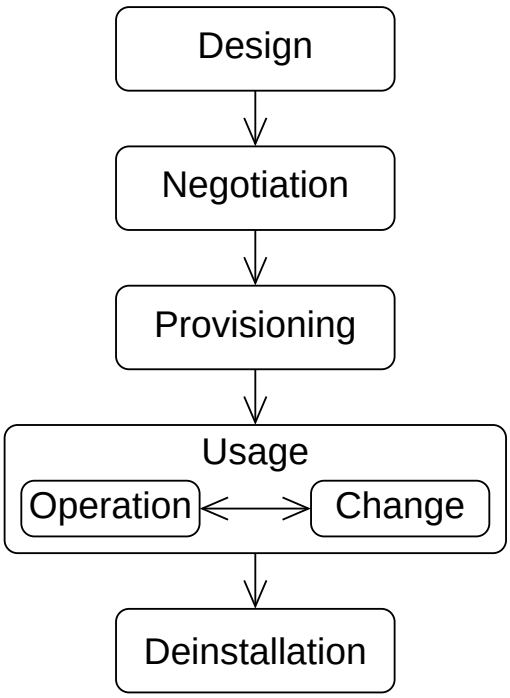


Abbildung 3.1: Lebenszyklus eines Dienstes nach dem MNM Dienstmodell

1. *Design:* Spezifikation und Vereinbarungen über Dienstgüteparameter sind die wichtigsten Interaktionen in dieser Phase.
2. *Negotiation:* Diese Verhandlungsphase ist geprägt durch die Vermittlungen zwischen Kunde und Provider über Tarife, Eskalationsmechanismen, Nutzungsvereinbarungen. Das Ergebnis führt zu einem SLA.
3. *Provisioning:* In dieser Phase kommt es zum Informationsaustausch zwischen Kunde und Provider (zum Beispiel über Nutzerdaten, Netzwerkverbindungen, etc.). Der Provider auf seiner Seite implementiert und testet den Dienst und dessen Managementfunktionen.
4. *Usage:* In dieser Phase wird der Dienst vom Kunden genutzt. Sie enthält zwei untergeordnete Phasen, die in Wechselbeziehung stehen. Während die *operation* Phase den normalen und Störungsbetrieb inklusive Unterstützung und Dienstgüteüberwachung einschließt, werden in der *change* Phase entsprechende Änderungen an der Dienstrealisierung vorgenommen. In dieser Phase wird auch das dienstorientierte Fehlermanagement angesiedelt.
5. *Deinstallation:* Freigabe verwendeter Ressourcen und Deinstallation des Dienstes.

- *Rollen* Wie schon in der Begriffsbestimmung in Abschnitt 1.3 gezeigt, ist die Identifizierung der bei Interaktionen beteiligten Rollen eines der Hauptziele des Modells. Die Kombination von Rollen und Interaktionen führt zu den Objekten, Beziehungen und Schnittstellen, die bei der Dienstbereitstellung involviert sind.
- *Dienstfunktionalitäten und QoS Parameter* Neben Interaktionsklassen lässt sich ein Dienst nach dem MNM Dienstmodell in zwei Funktionlitätsklassen - Nutzung und Management - einteilen. Während die Klasse Nutzung alle Interaktionen des Benutzers, die den eigentlichen Zweck des Dienstes repräsentieren, abdeckt, legt die Managementklasse die Funktionalitäten fest, die notwendig sind, um den Dienst dem Kunden anzupassen und die Qualität der Leistung sicherzustellen. Beide Funktionalitätsklassen müssen die in SLAs festgelegten QoS Parametern erfüllen. Diese Unterteilung ist ein wesentlicher Beitrag zur Möglichkeit der Unterscheidung verschiedener Funktionalitäten, die ein Dienst beinhalten kann.

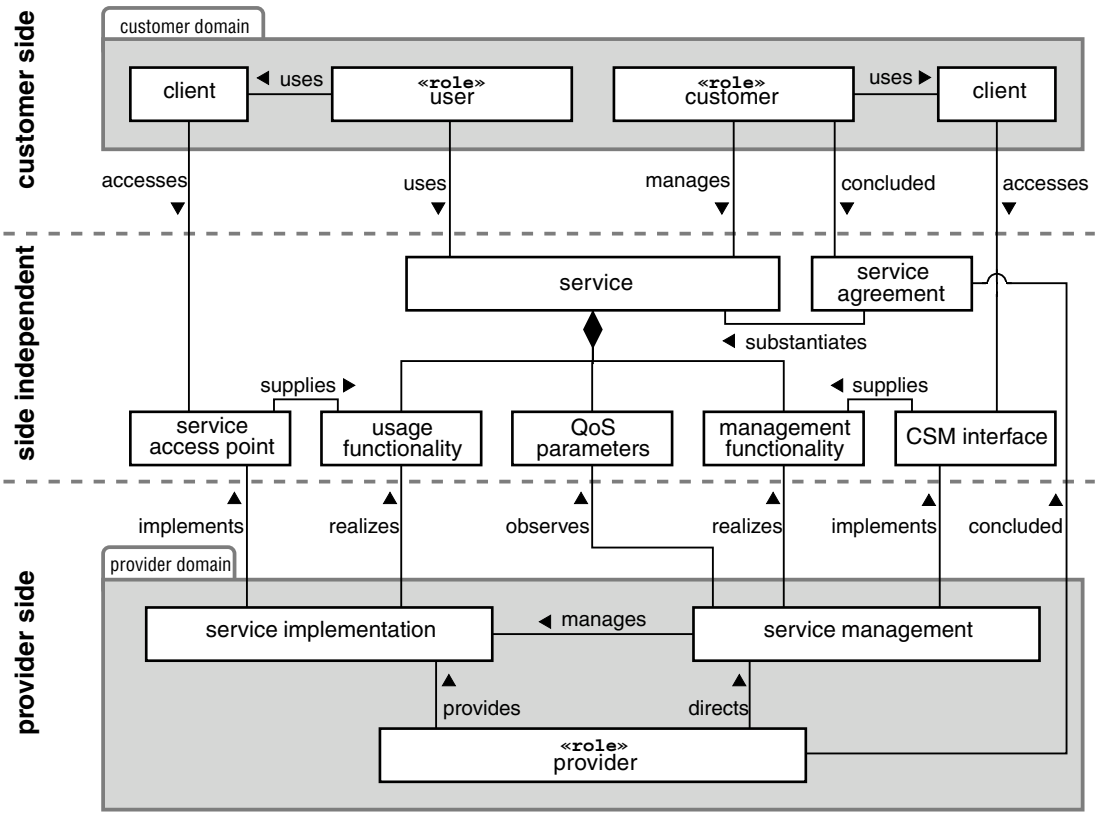


Abbildung 3.2: Übersicht über das MNM Dienstmodell

Dieses Wissen ist für ein Fehlermanagementsystem sehr wichtig, wie sich später in Kapitel 4 zeigen wird.

- *Dienstmodell* Eine Übersicht über das Dienstmodell zeigt Abbildung 3.2. Zunächst erfolgt eine (horizontale) Einteilung in drei Abschnitte. Oben und unten finden sich die zwei Hauptrollen, Kunde und Provider, wieder. In der Mitte werden Aspekte unabhängig von diesen Rollen betrachtet, zum Beispiel ein *Service Access Point*, der den Zugang zum Dienst zur Verfügung stellt. Auch der eigentliche Dienst wird hier aufgeführt. Auf der Kundenseite wird noch einmal die Aufteilung in Kunde und Benutzer dargestellt. Für das dienstorientierte Fehlermanagement ist vor allem die Managementseite interessant. Der Block *Service Management* überwacht den Dienst, realisiert die Managementfunktionalitäten (engl. *management functionality*), implementiert eine Managementschnittstelle für Kunden (engl. *customer service management (CSM)*) und regelt die Dienstrealisierung (engl. *service implementation*).
- *Rekursive Anwendung* Oftmals bieten Provider Dienste an, die sich aus weiteren Diensten zusammensetzen, die wiederum von verschiedenen Sub-Providern angeboten werden. Der Provider wird somit auch zum Kunden. Das MNM Dienstmodell erlaubt hier eine rekursive Anwendung. Die Providerseite wird dann mit Elementen der Kundenseite erweitert.

3.2.2 Zusammenfassung

Das MNM Dienstmodell ist die Basis für das verwendete Abhängigkeitenmodell (siehe Abschnitt 3.3). Es macht sehr generische Aussagen über Begriffe und Zusammenhänge bei Nutzung und Management von Diensten, trifft jedoch keine genaueren Aussagen darüber, wie Managementfunktionen auszusehen haben, oder gar wie sie zu implementieren sind. Es legt jedoch die Ziele und Hauptaufgaben fest. Das macht eine Anwendung auf nahezu beliebige Szenarien möglich.

Möchte man ein Regelsystem verwenden, um Ereigniskorrelation, zu realisieren, benötigt man wie bereits erwähnt genaue Kenntnisse über die Abhängigkeiten der für die Dienstnutzung benötigten Ressourcen beziehungsweise Dienste. Das MNM Dienstmodell alleine lässt dazu keine weiterführenden Rückschlüsse zu, erlaubt allerdings die Erweiterung oder Verfeinerung durch eine Abhängigkeitenmodellierung, wie sie im folgenden Abschnitt vorgestellt wird.

3.3 Verfeinerung des MNM Dienstmodells - Abhängigkeitenmodellierung

In [Marc 06] wird ein „systematischer, strukturierter Ansatz zur Modellierung von Abhängigkeitsbeziehungen“ durch eine Verfeinerung des MNM Dienstmodells entwickelt. Die Verwendung eines Entwurfsmusters aus der Softwareentwicklung ermöglichte die Erfüllung unter anderem dieser Anforderungen:

- Ein Abhängigkeitenmodell muss alle Phasen des Lebenszyklus eines Dienstes einschließen.
- Alle Funktionalitäten eines Dienstes müssen in die Abhängigkeitenmodellierung einbezogen und unterschieden werden können.
- Das Modell ist mit einer notwendigen, guten Erweiterbarkeit um neue Elemente ausgestattet.
- Einfache Definitionen und Strukturen der Elemente des Modells erlauben eine leichtere Anwendbarkeit auf unterschiedliche Szenarien.

3.3.1 Modellierung der Abhängigkeiten

Eine Übersicht über die in diesem Abschnitt vorgestellten Klassen und deren Beziehungen zueinander kann man sich in der Abbildung 3.4 verschaffen.

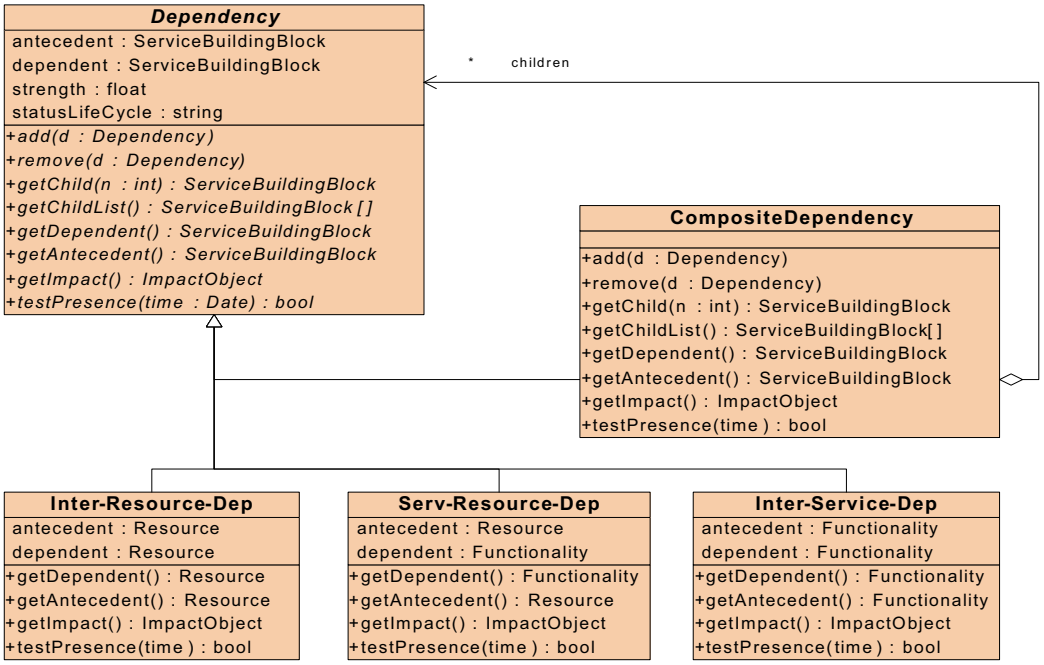


Abbildung 3.3: Abhängigkeitenmodellierung nach [Marc 06]

[Marc 06] verwendet das sogenannte *Composite Pattern* von [EG 95] - ein strukturelles Design Pattern aus der Softwareentwicklung. Der Zweck dieses Patterns ist die Anordnung von einzelnen Objekten in Baumstrukturen, um *Teil-Ganzes*-Hierarchien darzustellen. Das Composite-Pattern ermöglicht es einem Client, sowohl einzelne, als auch zusammengesetzte Objekte einheitlich zu behandeln. Weitere Informationen über dieses Design Pattern finden sich in [EG 95]. Genau diese Funktionalität, die das Pattern liefert, wird benötigt, um Dienste und Ressourcen in ihre Funktionalitäten aufzuteilen, wie beispielsweise im MNM-Dienstmodell, und die Abhängigkeiten untereinander festlegen zu können. Abbildung 3.3 zeigt die Anwendung dieses Patterns.

Der Begriff *ServiceBuildingBlock* repräsentiert dabei im Allgemeinen eine Teilkomponente, die zur Realisierung eines Dienstes beiträgt. Darunter fallen weitere Dienste und/oder Funktionalitäten, die einen Dienst repräsentieren, Sub-Dienste aus der Sicht dieses Dienstes, aber auch Ressourcen. Die ServiceBuildingBlock - Klasse ist eine abstrakte Oberklasse in der Hierarchie und repräsentiert alle Komponenten, die Teil einer Dienstrealisierung sind. Diese Oberklasse hat die Unterklassen *Resource*, welche für alle Ressourcen steht, und *Functionality*, welche eine Verallgemeinerung für Dienst, Dienstfunktionalität, Subdienst, Managementfunktionalität oder Nutzungsfunktionalität darstellt.

Um die Klasse *Functionality* angemessen repräsentieren zu können, wurde der Composite Pattern ein weiteres Mal 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 betrachtet werden kann. Es werden für die Beschreibung der Abhängigkeiten 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.

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.

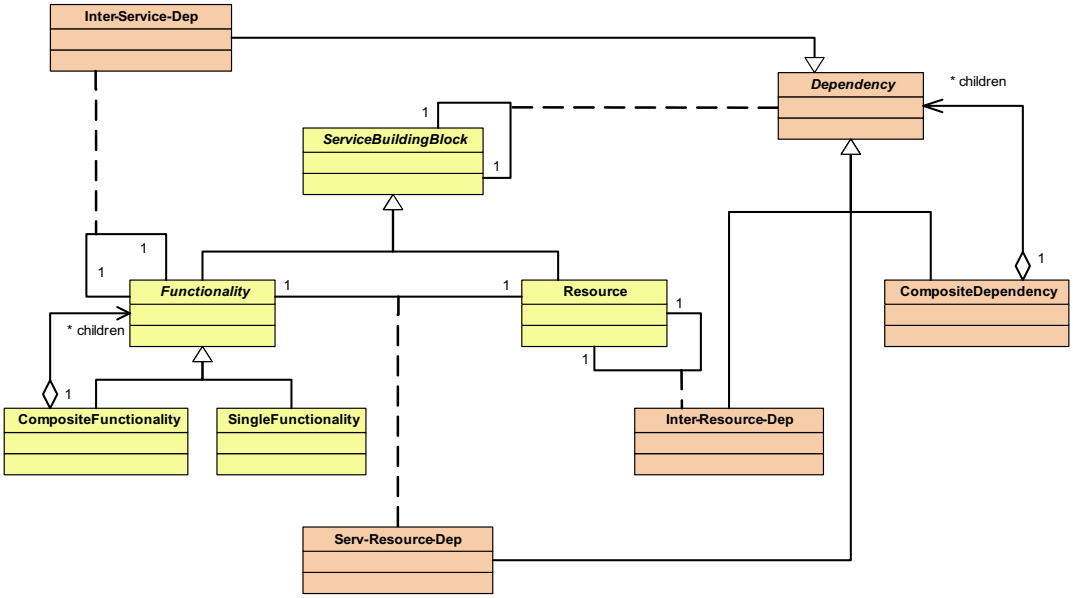


Abbildung 3.4: Klassendiagramm des Abhängigkeitsmodells

3.3.2 Zusammenfassung

Die Abhängigkeitenmodellierung erlaubt eine sehr detaillierte Strukturierung von Abhängigkeiten. Trotzdem ist dieser Detailgrad nicht unbedingt immer notwendig, abhängig von der gewünschten Modellierungstiefe. Die Modellierung kann auch für gröbere Strukturen verwendet werden. Die Verwendung des Design Patterns erlaubt eine problemlose Unterteilung von Diensten und Ressourcen in ihre verschiedenen Funktionalitäten. Trotz dem Gebrauch des MNM-Dienstmodells ist es möglich auch andere Modelle auf die Modellierung der Arbeit von [Marc 06] anzuwenden.

Für das Fehlermanagement ist von [Marc 06] bereits die Methode *getAntecedents* vorgesehen, welche, in der *CompositeDependency* Klasse rekursiv implementiert, alle von einer Komponente abhängigen Kindkomponenten zurück gibt. Wie später in Kapitel 4 dargestellt wird, hilft diese Methode Fehlerursachen aufzudecken. Auch wird erörtert, dass die Tiefe und der Detailgrad der Abhängigkeitenmodellierung entscheidende Auswirkungen auf die Ereigniskorrelation haben wird. Je präziser die Abhängigkeiten modelliert sind, desto präziser werden die Korrelationsergebnisse sein.

3.4 Dienstorientierte Ereigniskorrelation

In [Hane 06] wird ein Hybridmodell zur dienstorientierten Ereigniskorrelation vorgestellt. Nach der Darstellung der Notwendigkeit zur Dienstorientierung, wird bei diesem Ansatz eine Kombination aus einer regelbasierten und einer fallbasierten Korrelationskomponente vorgestellt. Dabei wird hier der sogennante top-down Ansatz angewendet. Dass sich die Vor- und Nachteile des *Case-Based* und eines *Rule-Based Reasoning* gegenseitig aufheben, wird in [Hane 07] dargestellt.

Grundlage für diese hybride Vorgehensweise sind *Service Events*. Ereigniskorrelation kann nur dann sinnvollerweise stattfinden, wenn Ereignisse vorliegen. Das ist erst dann der Fall, wenn ein Fehler oder ein Symptom bereits aufgetreten ist. Man spricht von einem *top-down* Ansatz. Dem gegenüber steht die *Impact* Analyse: Hier versucht man herauszufinden, welche Dienste nach dem Ausfall einer oder mehrere Komponenten betroffen sind oder sein könnten.

Die Ergebnisse der Ereigniskorrelation können wiederum als Eingabe für eine solche Analyse von Auswir-

kungen verwendet werden. Dieser Hybridansatz und weitreichende Aspekte dazu werden ausführlich in der zukünftigen Arbeit [Hane 07] fortgeführt.

3.5 Produkte

Es existiert eine Vielzahl von kommerziellen Systemen, welche im Fehlermanagement eingesetzt werden. Für diese Arbeit sollen nur zwei exemplarisch diskutiert werden, die in der momentanen Praxis am LRZ, wie im Szenario in Abschnitt 2.1 beschrieben, eine Rolle spielen.

Fehlende Projekte im Bereich quelloffener Software verhindern eine Aufnahme in diesen Abschnitt. Einzig das Projekt *SEC - Simple Event Correlator* wie in [Vaar 02b] vorgestellt bietet einen Themenbezug. Die eingeschränkten Möglichkeiten dieser Software verhindern jedoch potenzielle Weiterentwicklungen.

3.5.1 HP OpenView

Das Softwarepaket OpenView aus dem Hause Hewlett Packard wird von großen Unternehmen zur Überwachung und Verwaltung der eigenen IT - Infrastruktur eingesetzt. Für das Produkt sind zahlreiche Einzelmodule erhältlich, deren Einsatz dem jeweiligen Kunden angepasst werden kann. So sind auch Module wie zum Beispiel *Service Desk* verfügbar, die IT-Servicemanagement nach ITIL (siehe 3.1) implementieren.

Das LRZ setzt für das Ressourcenmanagement den *Network Node Manager (NNM)* ein. Dieser dient der Überwachung von Netzwerkkomponenten mit Hilfe des SNMP - Protokolls. Am LRZ werden hier nur Switches und Router verwaltet. Der NNM bietet eine graphische Darstellung der verwalteten Knoten sowie einen Alarmbrowser, in dem Events und Traps angezeigt werden können. Auch ist die automatisierte Ausführung von Skripten möglich.
Ein Addon *HP OpenView Event Correlation Services* erlaubt Korrelationsmöglichkeiten von Ereignissen der Ressourceninfrastruktur, bietet jedoch nur umständlich nutzbare Erweiterungsoptionen.

OpenView ist wie erwähnt ein System zur Verwaltung der Infrastruktur und nur äußerst eingeschränkt für das Management von Diensten einsetzbar. Dies trifft auch auf das folgende Produkt zu, allerdings sind dort weitergehende Erweiterungen in Richtung einer Dienstorientierung leichter anwendbar.

3.5.2 IBM Tivoli Enterprise Console

Die Tivoli Enterprise Console (TEC) wird in Kapitel 5 verwendet, um die in Kapitel 4 entwickelte Vorgehensweise zur dienstorientierten Ereigniskorrelation - zumindest teilweise - umzusetzen. Dazu erfolgt auch in Kapitel 5 eine genaue Betrachtung und Bewertung der Möglichkeiten der TEC, so dass hier zunächst eine allgemeine Beschreibung und die Vorstellung der grundsätzlichen Konzepte stattfindet.

3.5.2.1 Beschreibung

Die TEC von IBM ist eine Softwareanwendung, welche die Verwaltung von Ereignissen aus dem Netzwerk, Hardware und Softwarebereich einer IT-Infrastruktur übernehmen kann. Es bietet Unterstützung bei der Problemdiagnose und -lösung. Folgende Eigenschaften kennzeichnen dieses Produkt ([TB 04]).

- Globale Sicht auf die IT Umgebung
- Filterung, Korrelierung und automatische Reaktion auf bekannte Managementereignisse.
- In zahlreiche Schichten eingeteilte Umgebung zur möglichst quellennahen Behandlung der Ereignisse.
- Kann als zentrale Anlaufstelle für Alarme und Ereignisse von einer Vielzahl von Quellen fungieren. Mögliche Quellen sind andere IBM Tivoli Produkte, Fremdanwendungen, Netzwerkmanagementwerkzeuge sowie relationale Datenbanken.

Die schnelle Bearbeitung von großen Mengen an Ereignissen soll unter anderen durch folgende Punkte garantiert werden:

- Priorisierung von Ereignissen
- Ereigniskorrelation
- Filterung redundanter oder in der Priorität niedrig eingestufter Ereignisse

Verschiedene Adapter erlauben die Einbindung einer Vielzahl an Ereignisquellen. IBM verwendet die *Adapter Configuration Facility*, die als Werkzeug zur zentralisierten Konfiguration von verschiedenen Adaptern zur Verfügung steht. Events verschiedener Quellen können untereinander korreliert werden.

Die Hauptaufgabe der TEC besteht aus dem Verarbeiten einer großen Anzahl von Ereignissen zur Problemdiagnose und -lösung, was in dieser Reihenfolge erledigt wird:

1. Ereignisfilterung direkt am Quellenadapter. Diese herausgefilterten Ereignisse gelangen gar nicht erst an die TEC.
2. In einem Gateway ist eine Zustandsmaschine für die Korrelation von Ereignissen nahe an der Quelle verantwortlich. Eigene Regeln können definiert werden.
3. Der TEC Server korreliert Events aus unterschiedlichen Quellen und fasst sie zu Ereignissen zusammen, die zur Lösung der Fehlerursache beitragen, bzw. Auswirkungen auf Dienste anzeigen.
4. Die Event Console wird zur Ausgabe für Benutzer verwendet. Diese können basierend auf ihren eigenen Erfahrungen oder Unternehmensvorgaben manuell auf die Ereignisse zugreifen und diese bei Bedarf sogar verändern.

3.5.2.2 Konzepte

In Abbildung 3.5 ist ein Überblick über die Komponenten der TEC gegeben.
Dort sind alle Komponenten dargestellt, die die TEC zum Betrieb nutzt. Zentrales Element ist der Ereignisserver, welcher die Ereignisse der unterschiedlichen Quellen (zum Beispiel aus der *Event Integration Facility*) in einer Datenbank speichert und verarbeitet - hier finden auch die Regeln zur Behandlung von Events Platz. Weiterhin sind Komponenten für graphische Oberflächen vorhanden.

Der Begriff *Ereignis* oder engl. *Event* ist ein zentrales Element der TEC und erfordert daher eine genauere Definition. Da Ereignisse von einer Vielzahl von Quellen verarbeitet werden sollen, dienen die schon erwähnten Eventadapter dazu alle Ereignisse in ein einheitliches Format zu bringen, welches die Console versteht und so Eventinformationen verarbeiten kann. Dabei benutzt IBM ein **Klassenkonzept** zur Kategorisierung von Events. Eventklassen definieren eine Menge von Attributen und erlauben auch eine Vererbungshierarchie festzulegen - jedoch keine Polymorphie. So ist es möglich eigene Klassen zu definieren und Events durch Erweiterung der Attributmenge auf eigene Bedürfnisse anzupassen. Events selbst sind also nur Instanzen von Eventklassen. Jede Klasse erbt von der Basisklasse *EVENT*, in der bestimmte Attribute bereits festgelegt sind.

Ein Beispiel für eine Eventklasse zeigt die Definition der Klasse *TEC_Maintenance* in Listening 3.1. In der ersten Zeile steht der Klassenname. Die Attribute werden in den nachfolgenden Zeilen definiert. Dem Namen eines Attributes folgt durch einen Doppelpunkt getrennt, die Typdefinition. Einige Attribute werden in den Klassen mit Standardwerten belegt, welches mit dem Schlüsselwort *default* geschieht. Diese Klasse beschreibt ein Ereignis, welches den Wartungsmodus einer Netzwerkkomponente signalisiert.

Listing 3.1: Eventklassendefinition von *TEC_Maintenance*

```
TEC_CLASS:
  TEC_Maintenance ISA EVENT
    DEFINES {
      current_mode: MODE, default="ON";
      mode_status: STRING;
      command: STRING;
      start_time: INT32;
      max_time: INTEGER;
```

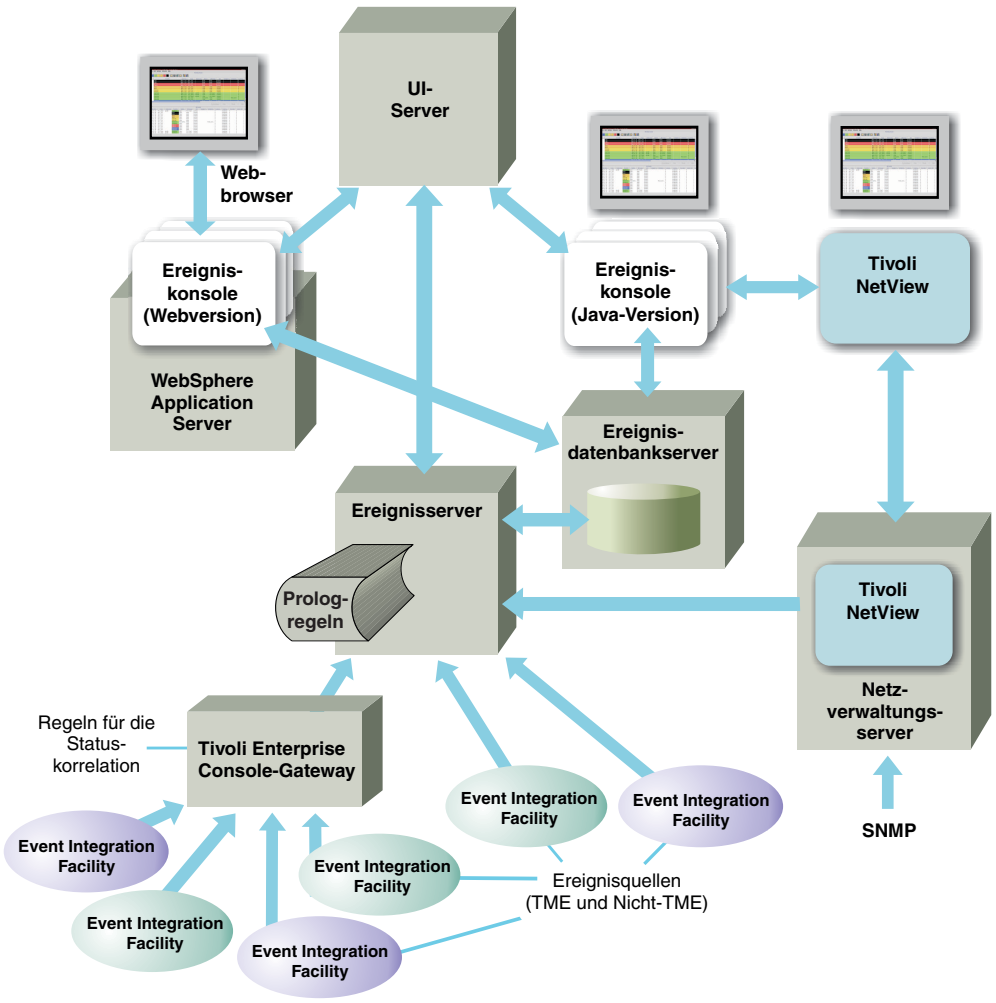


Abbildung 3.5: Komponenten der Tivoli Enterprise Console

```
      owner_info: STRING;
    };
END
```

Wie diese Klasse auch zeigt, beschreibt ein Ereignis bei der TEC nicht immer nur ein Problem. So verarbeitet die Software auch zahlreiche Events, die teils nur informeller Natur sind. Oft werden auch von der Console selbst Events generiert, die zum Beispiel Informationen an den Administrator beinhalten. Weiterhin werden auch sogenannte „positive Ereignisse“ unterstützt. Diese bescheinigen einer Komponente ihre Funktionsfähigkeit und können mit eventuell zuvor eingegangen Problemereignissen korreliert werden.

Alle Events werden von einem **Ereignisserver** verarbeitet und validiert. Validierte Events werden vom Ereignisserver mit einer eindeutigen ID versehen und in einer Ereignisdatenbank gespeichert. Diese Datenbank ist extern angelegt und auf große Ereignismengen ausgelegt. Der Ereignisserver verarbeitet eingehende Ereignisse streng sequentiell und hat die Möglichkeit Events zu puffern, bevor sie in die Datenbank geschrieben werden. Das ist notwendig, wenn zahlreiche Ereignisse in kurzer Zeit eintreffen.

Die konkrete Verarbeitung und Filterung von Ereignissen findet mit Hilfe von sogenannten **Regelsätzen** (engl. *rule set*) statt. Diese sind in einer auf der logischen Programmiersprache Prolog aufsetzenden Regelsprache formuliert. IBM proprietäre Erweiterungen ermöglichen ein großes Spektrum an Filtern oder auch Korrelationsregeln. Diese Regeln können mit einem graphischen Editor oder auch manuell mit einem Texteditor bearbeitet und erstellt werden. IBM stellt dem Benutzer mehrere Regelsätze für unterschiedliche Aufgaben zur Verfügung. Diese Aufgaben umfassen zum Beispiel das Bereinigen der Ereignisdatenbank, die Ereigniskorrelation oder die spezifische Verarbeitung von Events, dievon bestimmten Ereignisadaptern eingespeist werden. Ein Regelsatz ist eine Sammlung von Regeln. Vor dem Einsatz dieser Regelsätze müssen diese kompiliert werden. Eine Zusammenstellung von IBM bereits implementierten Regelsätzen findet sich in der *Rule Set Reference* [IBM d].

Die Kombination aus Regelsätzen und Eventklassendefinitionen bezeichnet IBM als Regelbasis (*rule base*). Mit der Auslieferung des Produkts liegt eine Standardregelbasis mit mehreren Regelsätzen vor. Eine genauere Beschreibung dieser Regelsätze findet sich in Abschnitt 5.1.2. Für einen Eventserver kann immer nur eine Regelbasis aktiv und geladen sein.

Eine Regel wird beim Eintreffen eines Events, für dessen Eventklasse sie definiert wurde aktiv. Die Ausführung ist von den in der Regel festgeleten Bedingungen an das Event abhängig. Die Aktionen, die die Regel durchführt, können zum Beispiel eine automatische Antwort durch Ausführung eines Skriptes, das Verändern von Eventattributen (Erhöhung des Schweregrades eines Events), das Benachrichtigen von Administratoren oder das Löschen dieses Events aus der Ereignisdatenbank sein. Weiterführende Informationen gibt das TEC Benutzerhandbuch [IBM b].

Die Tivoli Enterprise Console bietet zwei mögliche graphische Oberflächen an: Eine Java-basierte und eine Browser-basierte Version. Die Java-Oberfläche stellt dabei einen größeren Funktionsumfang bereit. Hier findet sich auch der graphische Regeleditor. Das Produkt läuft auf zahlreichen Betriebssystemen wie Windows Server 2003, AIX, HP-UX, Linux, Red Hat, SuSE oder Solaris.

3.5.2.3 Bewertung

Die TEC ist ein kommerzielles Produkt der Tivoli Produktlinie von IBM. Dieser Umstand ist auch der Grund, warum der Quelltext einiger Algorithmen zum Beispiel zur Korrelation, nicht einsehbar ist. Ein nicht zwangsweise negativer Aspekt ist, dass dieses Produkt durch die Vielzahl an Komponenten und Funktionen sehr komplex und umständlich anzuwenden ist. Das erklärt sich teilweise durch die Auslegung auf große und eben komplexe Infrastrukturen, wie sie in Unternehmen auftreten können. Der Nutzerkreis konzentriert sich dementsprechend auf größere Unternehmen und deren Systemadministratoren.

Wie viele andere regelbasierte Produkte zur Ereigniskorrelation geht TEC von singulären Ursachenfehlern aus (single root cause). Diese Annahme hilft die Menge an aktiven Events zu reduzieren. Konnte einmal eine Beziehung zwischen einem Event auf höherer Ebene und einem auf darunter liegendem Niveau hergestellt werden, wird der Suchvorgang nach weiteren Ursachen nicht fortgeführt.

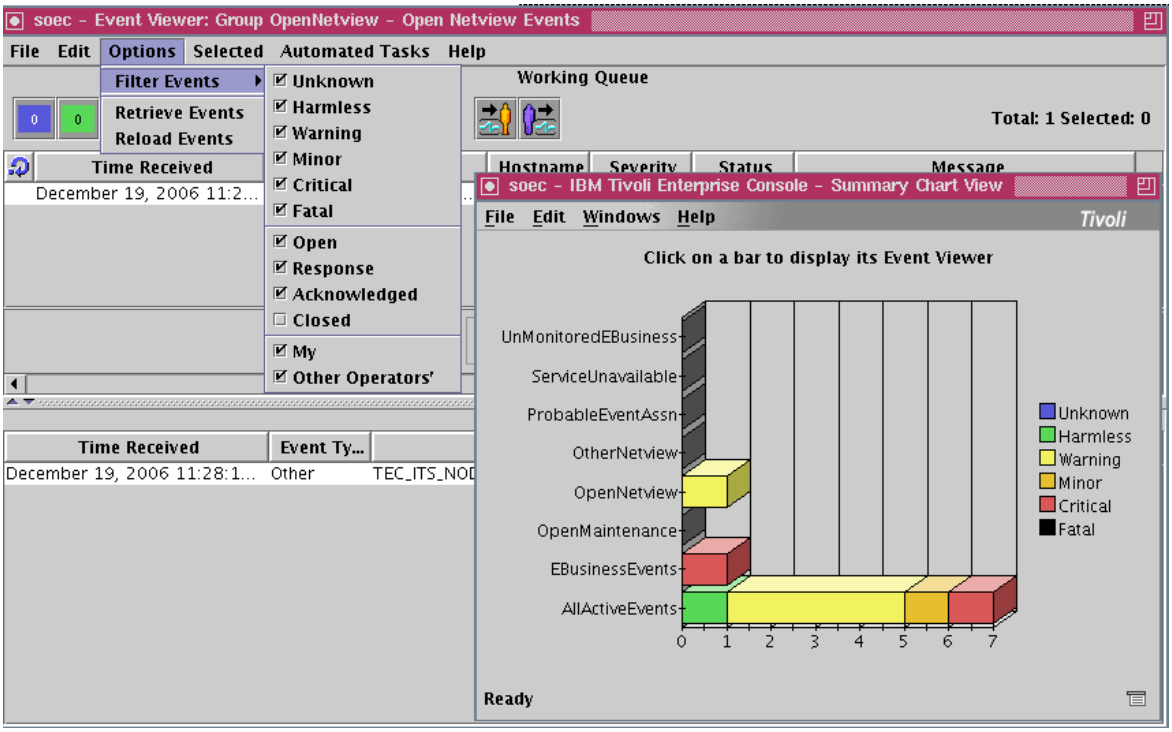


Abbildung 3.6: Der TEC EventViewer. Im Vordergrund eine Balkendarstellung von aktiven Ereignissen. Die Farbe spiegelt dabei den Schweregrad der Events wieder.

Positiv zu bewerten ist, dass die TEC auf einer Vielzahl von unterschiedlichen Betriebssystemen laufen kann. Mit der *Adapter Configuration Facility* stehen dem Anwender viele Möglichkeiten offen, eigene Ereignisse in das System einzugeben. Filter- und Korrelationsregeln können in einer eigenen Beschreibungssprache definiert werden. Dies zeigt, dass IBM dem Benutzer zahlreiche Eingriffsmöglichkeiten bei der Verarbeitung von Ereignissen zugesteht, und führt letztendlich auch dazu, dass das Produkt für eine Implementierung in Kapitel 5 verwendet wird.

4 Vorgehensweise zur dienstorientierten Ereigniskorrelation

Inhaltsverzeichnis

4.1 Ereigniskorrelation durch Aussagenlogik 30
4.1.1 Elemente der Aussagenlogik 30
4.1.2 Getroffene Annahmen 30
4.1.3 Anwendung auf die Ereigniskorrelation 31
4.1.4 Anwendung am Szenario 33
4.1.5 Zusammenfassung und Bewertung 34
4.2 Der Faktor Zeit 36
4.2.1 Correlation Window - Zeitfenster 36
4.2.2 Beispiel 38
4.2.3 Klassendiagramm 38
4.2.4 Zusammenfassung 39
4.3 Dienstgüteparameter und Dienstfunktionalitäten 40
4.3.1 QoS - Dienstgüteparameter 40
4.3.2 Dienstfunktionalitäten 40
4.3.3 Erweiterung des Klassendiagramms 41
4.3.3.1 Modellierung der QoS Paramter 41
4.3.3.2 Modellierung der Dienstfunktionalitäten 42
4.3.4 Beispiel 44
4.3.5 Zusammenfassung und Bewertung 46
4.4 Ereigniskorrelation mit Score 47
4.4.1 Score für Dienste, Ressourcen und Events 47
4.4.1.1 Grundscore 47
4.4.1.2 Score und QoS 49
4.4.1.3 Scoreregression 49
4.4.1.4 Ressourcenredundanzen 51
4.4.2 Erweiterung des Klassendiagramms 52
4.4.2.1 Components 52
4.4.2.2 Events 53
4.4.3 Zusammenfassung und Bewertung 53
4.5 Korrelationsalgorithmus 54
4.5.1 Anpassung des Korrelationsalgorithmus 54
4.5.2 Prozeduren 57
4.5.3 Zusammenfassung 60

Kapitel 4 beschreibt die in dieser Arbeit entwickelte Vorgehensweise zur dienstorientierten Ereigniskorrelation. Zunächst wird ein grundlegender Algorithmus erstellt, der die Korrelation durch einfache aussagenlogische Formeln vornimmt. Um den Zusammenhang zwischen Diensten, Ressourcen und Events zu verdeutlichen, werden Klassendiagramme verwendet. Die Erweiterung dieser Klassen mit Attributen und Methoden werden schrittweise die in Abschnitt 4.1 getroffenen Annahmen aufheben und die in Kapitel 2 entwickelten Anforderungen erfüllen.

4.1 Ereigniskorrelation durch Aussagenlogik

In diesem Kapitel soll ein Ansatz zur Ereigniskorrelation mittels einfacher aussagenlogischer Regeln vorgestellt werden. Dazu werden zunächst die wichtigsten Elemente der Aussagenlogik vorgestellt. Im nächsten Abschnitt wird gezeigt, wie mit Hilfe eines Abhängigkeitenmodells (siehe Erläuterung in Abschnitt3.3) aussagenlogische Regeln erstellt werden können. In der Zusammenfassung und Bewertung wird anhand des Anforderungskataloges in Abschnitt 2.3 dieser Ansatz zusammenfassend bewertet. Bei der Anfertigung dieser Grundlage der Vorgehensweise wird es eine Reihe von Annahmen geben, die dann in den nächsten Abschnitten schrittweise aufgehoben werden.

4.1.1 Elemente der Aussagenlogik

Die Aussagenlogik ist ein Teilbereich der Logik, die sich mit Beweisen für Formeln einfacher Struktur, in denen nur Terme der Sorte *bool* betrachtet werden, beschäftigt. Grundterme der Sorte *bool* werden als elementare oder atomare Aussagen bezeichnet. Sie können ausschließlich die Wahrheitswerte *true* oder *false* annehmen. In der Logik werden Regelssysteme (Ableitungs- oder Inferenzregeln) angegeben, die es erlauben, aus einer Menge von als wahr angenommenen Formeln (Axiome) weitere Formeln abzuleiten. Die Ableitung einer Formel heißt auch formaler Beweis der Formel. Für die Aussagenlogik gelten die Ableitungsregeln der Aussagenlogik, sowie die Gesetze der Booleschen Algebra. Mit Hilfe dieser Struktur können aus einer gegebenen Menge von Aussagen weitere Aussagen abgeleitet werden [Broy 98]. Man spricht hier auch von Schlussregeln in einem Ableitungssystem. Der Begriff der Ableitung formalisiert das Konzept eines mathematischen Beweises. Für diese klassische Aussagenlogik existieren Kalküle, also formale Systeme, die sowohl korrekt als auch vollständig sind. Ein formales System definiert ein Alphabet der logischen Sprache, bestehend aus atomaren, also nicht mehr ableitbaren, Aussagen und einer Menge von Junktoren und Gleichheitszeichen.

4.1.2 Getroffene Annahmen

Möchte man mit Hilfe einfacher aussagenlogischer Elemente ein System zur Ereigniskorrelation bilden, müssen zunächst einige Annahmen getroffen werden.

Statisches Abhängigkeitenmodell Diese Vorgehensweise geht von einem statischen Abhängigkeitenmodell aus. Ändern sich Abhängigkeiten, können nicht mehr alle, bzw. werden zu viele abhängige Komponenten eines Dienstes berücksichtigt. Falsche Korrelationsergebnisse sind dann die Folge. Wie bereits in der Anforderungsanalyse in 2 beschrieben, existiert auch im Szenario am LRZ eine dynamische Ressourcen- und Dienststruktur, die eine Anpassung der Korrelation erfordert.

Annahmen über Events Es werden *Service Events* und *Resource Events* betrachtet. Die Events haben volle Glaubwürdigkeit, das heißt die Information, die sie tragen, trifft immer zu. Diese Information bezieht sich immer auf einen einzelnen Dienst bzw. auf eine einzelne Ressource. Nach der Korrelation werden die Events verworfen. Weiterhin wird die Quelle der Events nicht weiter bewertet. *Service Events*, die von Kunden bzw. Nutzern kommen, werden mit der gleichen Priorität behandelt wie Events, die automatisch generiert wurden.

Annahmen über den Dienststatus Es existieren ausschließlich binäre Dienstzustände. Ein Dienst hat entweder den Status „funktioniert“ oder „funktioniert nicht“. Es werden keine weiteren Abstufungen bei der Funktionalität getroffen. Die Information, die Events tragen, bezieht sich immer nur auf diesen Status.

Zeitunabhängigkeit Der Faktor Zeit wird in diesem Ansatz vollständig ignoriert. Weder werden Service Events mit Hilfe eines Zeitstempels auf ihre Gültigkeit überprüft, noch die zeitliche Reihenfolge und Anzahl, in denen Events innerhalb eines Zeitraumes eintreffen, beachtet. Es wird also angenommen, dass Events nach ihrem Eintreffen sofort korreliert werden und das Korrelationsergebnis auch sofort vorliegt. Das bedeutet, dass

es während der Korrelation gibt keine weiteren Events gibt. Trifft mehr als ein Event ein, so wird angenommen, dass diese gleichzeitig vorliegen.

Ressourcenredundanzen Hängt ein Dienst von mehreren redundanten Ressourcen ab, reicht eine dieser Ressourcen, um die volle Dienstfunktionalität aufrecht zu erhalten.

Regelabdeckung Es wird zusätzlich angenommen, dass nur Fehlersituationen auftreten, die mit Hilfe dieser Vorgehensweise aufgedeckt werden können. Umgekehrt wird angenommen, dass jede Fehlersituation erkannt wird. Aus dieser Annahme folgt, dass keine weitere Komponente benötigt wird, die aktiv wird, wenn keine Regel greifen kann. Diese Annahme wird im weiteren Verlauf der Arbeit nicht aufgehoben. Ein Lösungsvorschlag ist der in [Hane 07] vorgestellte Hybridansatz, der aus einer regelbasierten und einer fallbasierten Komponente besteht.

4.1.3 Anwendung auf die Ereigniskorrelation

Um die Aussagenlogik für die Zwecke der Ereigniskorrelation nutzen zu können, wird zunächst definiert, welche Art von Aussagen verwendet werden sollen.
Wie oben beschrieben kann die Aussagenlogik nur mit Aussagen arbeiten, die die Wahrheitswerte wahr oder falsch haben. So ist es also notwendig zu definieren, auf was sich diese Aussagen beziehen. Hier kann mit einer Abbildung gearbeitet werden, welche die korrekte Funktion eines Dienstes oder einer Ressource als *wahr*, und das Nichtfunktionieren als *falsch* abbildet. Funktionierende Komponenten werden also als Axiome aufgefasst.

Die Vorgehensweise erfordert neben den Elementen der Aussagenlogik, mit deren Hilfe die eigentliche Korrelation geschieht, auch Komponenten eines wissensbasierten Systems wie in 2.2.1 vorgestellt. Konkret stellt die Anwendung der aussagenlogischen Ableitungen die Interferenzkomponente eines regelbasierten Systems dar. Die Fakten, also die Wissensbasis, werden durch die eben definierten Aussagen über Dienste und Ressourcen definiert. Eine Komponente zur Generierung neuer Regeln fehlt zunächst in diesem Ansatz, da keine neuen Regeln erzeugt werden, sondern die Korrelation nur mit Hilfe aussagenlogischer Vorschriften stattfindet. Ebenso nur skizziert ist eine Schnittstellenkomponente zur Kommunikation mit anderen Teilen. Die Funktion einer Schnittstelle übernehmen einzig die *Service Events*, die von „außen“ eingespeist werden.

Wie also können diese atomaren Aussagen in der Wissensbasis zusammen verknüpft - also korreliert - werden? Interferenzregeln bilden nach [Broy 98] ein Ableitungssystem. Die Anwendung dieser Regeln in einer Folge ist als *logisches Schließen* bekannt. In jedem Schritt dieser Folge entstehen Sätze entweder aus Axiomen oder aus vorangegangenen Sätzen mit Hilfe von Interferenzregeln.

Mit Anwendung der Methode des logischen Schließens auf die Ereigniskorrelation, muss man die Interferenzregeln auf Terme anwenden, die sich aus den bekannten und nicht bekannten Status der Dienstelemente zusammensetzen. Mit einem initialem Satz, der den aktuellen Zustand der Dienste wiedergibt, können jetzt durch logisches Schließen und mit Hilfe des Abhängigkeitenmodells Schlüsse gezogen werden, die Aussagen über die *root cause* eines Problems auf Dienstebene erlauben. Dieser initiale Satz wird nach dem Eintreffen eines oder mehrerer Service Events gebildet. Ein wichtiger Schritt ist dabei die Anwendung des Wissens aus der Abhängigkeitenmodellierung.

Nach [Marc 06] kann die Funktion *getAntecedent()* eines *ServiceBuildingBlock* (siehe auch 3.3), welcher ja Abhängigkeiten repräsentiert, dazu verwendet werden, um alle Komponenten, von denen der Dienst abhängt, herauszufinden. Bei diesem Schritt muss zwischen zusammengesetzten und nicht zusammengesetzten ServiceBuildingBlocks unterschieden werden. Im ersten Fall muss diese Funktion rekursiv implementiert sein, um alle Antecedents herauszufinden. Ein initialer Satz besteht also zunächst aus den mit UND verknüpften Status der Dienste. Vorliegende *Service Events*, zum Beispiel von Benutzern, beinflussen den Dienststatus. Für die Dienste, über deren Dienststatus nichts bekannt ist, weil keine Service Events vorliegen, wird vorerst die einwandfreie Funktion angenommen. Im nächsten Schritt hilft die Funktion *getantecedent()*, die Dienste oder Ressourcen zu ermitteln, von denen die einzelnen Dienste abhängen. Unter Anwendung des Gesetzes von de Morgan findet nun die Ersetzung des Dienstatus mit denen der Antecedents statt. In den restlichen Schritten

werden die Interferenzregeln auf dem entstandenen Term angewendet, bis keine Ableitungen mehr möglich sind.

Für *Resource Events* wendet man eine ähnliche Vorgehensweise an: Die Funktion *getantecedent()* liefert keine Dienste, sondern weitere Ressourcen. Als Initialsatz müssen dementsprechend auch die mit UND verknüpften Status der Ressourcen verwendet werden.

Diese Vorgehensweise lässt sich in einem Algorithmus wie in der Prozedur *Correlate* 1 formalisieren.

Procedure 1 Algorithmus

```
1:  procedure CORRELATE                                   ▷ d_1...d_n nehmen boolsche Werte ein
2:      d_1...d_n
3:                                                        ▷ t ist ein boolscher Term
4:      t = d_1 ∧ ... ∧ d_n
5:                                                        ▷ Einsetzen der Abhängigkeiten
6:      t = Antecedents(d_1) ∧ ... ∧ Antecedents(d_n)
7:                                                        ▷ t' ist der mit Interferenzregeln abgeleitete Term
8:      t ↦ t'
9:                                                        ▷ Die Interferenzregeln werden solange wie möglich angewendet
10:     repeat
11:         t := t'                                       ▷ t erneut ableiten
12:         t ↦ t'
13:     until t ≠ t'
14: end procedure
```

Es entsteht schließlich ein Term, aus dem sich Aussagen über mögliche Fehlerursachen treffen lassen können. Aus den Eingaben, also den *Service Events* der Benutzer, bestimmt der Algorithmus ausschließlich durch logisches Schließen die möglichen Ursachen für den Ausfall eines oder mehrere Dienste auf höherer Ebene. Mit Dienste „höherer Ebene“ sind jene Dienste gemeint, mit denen Nutzer unmittelbar interagieren und daher auch nur für diese Dienste *Service Events* vorliegen können.

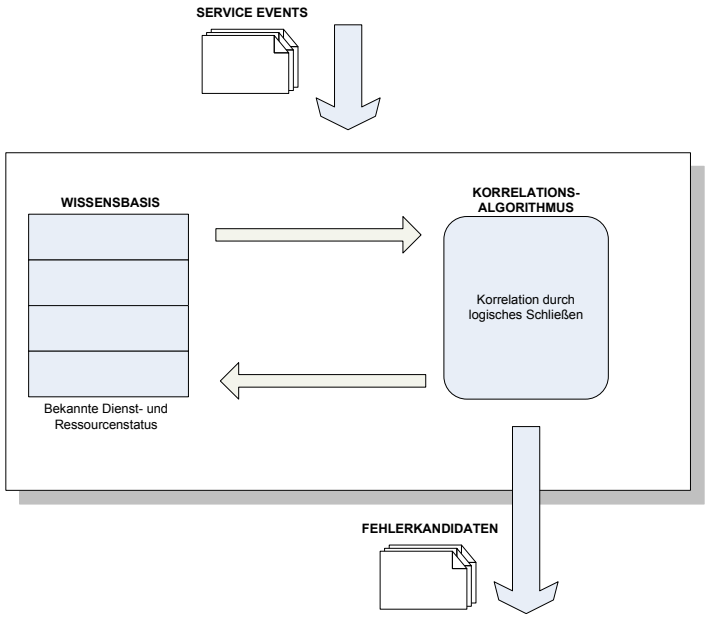


Abbildung 4.1: Komponenten und Beziehungen

Abbildung 4.1 fasst die beteiligten Komponenten und deren Beziehungen zueinander zusammen. Zunächst dienen Service Events als Eingabe. Die Wissensbasis verwaltet die bekannten Dienst- und Ressourcenstatus. Der Korrelationsalgorithmus führt die Korrelation durch logisches Schließen, wie eben beschrieben, durch.

Als Ergebnis wird eine Liste mit Fehlerkandidaten erstellt.

4.1.4 Anwendung am Szenario

Diese erste Vorgehensweise wird auf ein Beispiel aus dem Szenario am LRZ angewendet. Zur Verwendung kommt wieder die Abhängigkeitenmodellierung. Das Beispiel wird aus Komplexitätsgründen auf den Webhosting Dienst und dessen Funktionalitäten „Zugriff auf statische Seiten“, „Zugriff auf dynamische Seiten“ und „Zugriff auf geschützen Bereich“ eingeschränkt. Der Webhostingdienst am LRZ wurde bereits im Szenario in Abschnitt 2.1 vorgestellt und beschrieben.

Die Anwendung des vorgestellten Ansatzes zur Ereigniskorrelation erfordert, wie bereits erwähnt, genaue Kenntnis der beteiligten Ressourcen und Dienste. Nach der Abhängigkeitenmodellierung von [Marc 06] ergibt sich dazu das Bild wie in Abbildung 4.2 skizziert. In dieser Abbildung sind Dienste mit abgerundeten Rechtecken und Ressourcen mit regulären Rechtecken dargestellt.

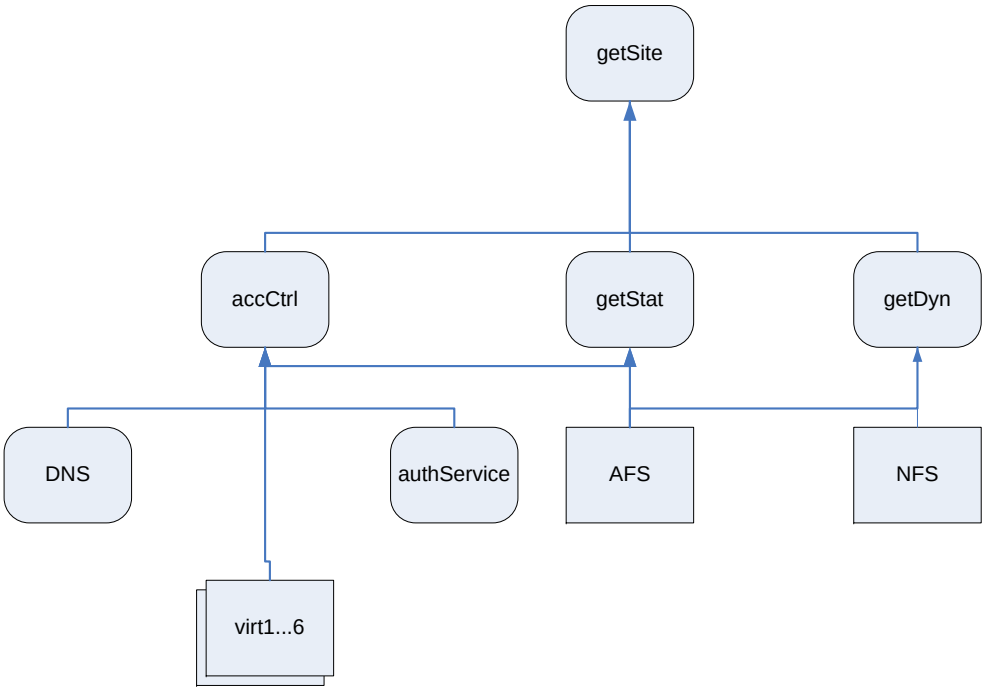


Abbildung 4.2: Beteiligte Funktionalitäten beim Webhosting Dienst des LRZ

Ganz oben in der Hierarchie steht die Funktion *getSite*, die den Zugriff auf beliebige, vom LRZ bereitgestellte, Seiten repräsentiert. Weiterhin lässt sich diese in drei weitere Funktionalitäten aufteilen. Für den Zugriff auf statische Seiten steht die Funktion *getStat*, die auf die redundanten virtuellen Webserver *virt1...6* und das AFS Dateisystem zugreift. Da statische Seiten auch innerhalb eines geschützten Bereichs liegen können, wird auch noch die Funktionalität *accCtrl* benötigt, die eben dies realisiert. *accCtrl* benötigt den DNS Dienst, der sich hier aus dem DNS Dienst des LRZ (*DNSLRZ*) und dem des MWN (*DNSMWN*) zusammensetzt. Zusätzlich werden **htaccess** und *htpasswd* Dateien benötigt. Dynamische Seiten (*getDyn*) können vom Webhostingdienst nur dann geliefert werden, wenn sowohl der Zugriff auf das *AFS* als auch auf das *NFS* Dateisystem möglich ist.

Beispiel 1 Im ersten Beispiel wird angenommen, ein *Service Event* eines Benutzers liegt vor, welches aussagt, dass der Zugriff auf geschützte Bereiche nicht möglich sei. Wir starten mit leerer Wissensbasis, das heißt, es wird angenommen, dass alle Dienste und Ressourcen einwandfrei funktionieren. Mit Hilfe des oben

skizzierten Algorithmus zur Ereigniskorrelation durch logisches Schließen wird jetzt versucht, die Ursache für die Störung des Dienstes zu ergründen.

Zunächst werden die Dienststatus festgelegt. Da nur ein Service Event ansteht, welches den Ausfall von *accCtrl* bescheinigt, werden die anderen Dienststatus als funktionierend angenommen.

$getStat = true$

$getDyn = true$

$accCtrl = false$

Im folgenden Term werden die Status mit UND verknüpft.

$T := getStat \wedge getDyn \wedge \neg accCtrl$

Die nächsten Schritte beschreiben das logische Schließen mit Interferenzregeln.

$T = (AFS \wedge (\neg DNS \vee \neg authService)) \wedge (AFS \wedge NFS) \wedge (\neg DNS \vee \neg authService)$

$T = (AFS) \wedge (AFS \wedge NFS) \wedge (\neg DNS \vee \neg authService)$

$T = AFS \wedge NFS \wedge (\neg DNS \vee \neg authService)$

Interessant in diesem letzten Term sind natürlich die Dienste, die als nicht funktionierend markiert sind. Diese werden jetzt in die Kandidatenliste aufgenommen.

Beispiel 2 In diesem Beispiel wird angenommen, dass der Zugriff auf dynamische Seiten nicht mehr funktioniert. Wieder wird ein Term aus den Dienststatus gebildet, die Abhängigkeiten eingesetzt und die Interferenzregeln angewendet.

$getStat = true$

$getDyn = false$

$accCtrl = true$

$T := getStat \wedge \neg getDyn \wedge accCtrl$

$T = (AFS \wedge accCtrl) \wedge (\neg AFS \vee \neg NFS) \wedge (DNS \wedge authService)$

$T = (AFS \wedge DNS \wedge authService) \wedge (\neg AFS \vee \neg NFS) \wedge (DNS \wedge authService)$

$T = (AFS \wedge (\neg AFS \vee \neg NFS)) \wedge (DNS \wedge authService)$

$T = ((AFS \wedge \neg AFS) \vee (AFS \wedge \neg NFS)) \wedge (DNS \wedge authService)$

$T = (FALSE \vee (AFS \wedge \neg NFS)) \wedge (DNS \wedge authService)$

$T = AFS \wedge \neg NFS \wedge DNS \wedge authService$

An diesem Beispiel ist zu erkennen, wie das logische Schließen die Ressource *AFS* aus der Kandidatenliste ausschließt, obwohl der Dienst *getDyn* davon direkt abhängt. Würde das *AFS* ausgefallen sein, würde ja auch der Zugriff auf statische Seiten nicht mehr funktionieren. Im Beispiel lag dazu jedoch keine Information vor. Aus diesem Grund ist auch die Ressource *NFS* als die mögliche Fehlerursache markiert.

Diese Beispiele zeigen, wie mit Hilfe des relativ simplen Algorithmus dennoch Ergebnisse erzielt werden können, die zumindest teilweise den gestellten Anforderungen gerecht werden. Diese Ergebnisse konnten jedoch nur unter Berücksichtigung der Annahmen weiter oben erreicht werden. Welche Anforderungen genau erfüllt bzw. nicht erfüllt wurden werden im nächsten Abschnitt erläutert.

4.1.5 Zusammenfassung und Bewertung

Zur Bewertung dieses Ansatzes soll die Anforderungsanalyse herangezogen werden. Die folgende Tabelle listet die gestellten Anforderungen auf und zeigt inwieweit der vorgestellte Ansatz sie erfüllt:

Anforderung	erfüllt	Erläuterung
Priorisierung	nicht erfüllt	Die Ereignisquelle wird in diesem Ansatz nicht berücksichtigt. Eine Priorisierung nach dieser Quelle findet also nicht statt.
Zeitproblematik	nicht erfüllt	Aus den Annahmen geht hervor, dass der Faktor Zeit in diesem Ansatz vollständig ignoriert wird.
Ressourcen- und Dienststruktur	teilweise erfüllt	Aus den Annahmen geht hervor, dass von einem statischen Abhängigkeitenmodell ausgegangen wird. Ändert sich jedoch dieses Modell, kann der Ansatz ohne weitere Eingriffe fortbestehen, da die Informationen ausschließlich aus dem Modell stammen. Eine automatische Regelerstellung findet in diesem Ansatz nicht statt. Die Korrelation findet hier nur mit Hilfe aussagenlogischer Vorschriften statt.
Priorisierung von Ereignissen	nicht erfüllt	Eine Priorisierung von Ereignissen findet nicht statt - schon weil die Annahmen über Events eine Beachtung oder Wertung der Quelle der Events ausschließt.
Messen der Dienstgüteparameter	nicht erfüllt	Dienstgüteparameter spielen in diesem Ansatz auch keine Rolle, da Service Events sich immer auf einen einzelnen Dienst beziehen und nur aussagen, ob dieser funktionsfähig ist oder nicht
Active Probing	nicht erfüllt	Es liegt kein Mechanismus vor, der das Anstoßen von aktiven Tests auf Diensten oder Ressourcen regelt.
Dienste von Drittanbietern	erfüllt	Dienste von Drittanbietern, die in der Abhängigkeitenstruktur eingebunden sind, können wie Ressourcen betrachtet werden. Da in der Regel keine Kenntnisse über die detaillierte Abhängigkeitenstruktur dieser Dienste vorliegen, stellt dieser ein Blatt in der Abhängigkeitenmodellierung dar.
Multiple Ressourcenfehler	erfüllt	Der Ansatz kann multiple Ressourcenfehler erkennen und die entsprechenden Fehlerkandidaten in die Ergebnisliste aufnehmen. Betrachtet man Beispiel 2, und nimmt den Ausfall der NFS und der AFS Funktionalität an, so kann auch angenommen werden, dass Service Events sowohl für den getStat als auch für den getDyn Dienst vorliegen würden.
Dienstinstanzen	nicht erfüllt	Dienstinstanzen werden in diesem Ansatz nicht berücksichtigt.
Ressourcenredundanzen	teilweise erfüllt	Fehler innerhalb redundanter Ressourcen werden zwar erkannt, können aber nicht weiter auf eine bestimmte Ressource eingegrenzt werden. Zusätzlich wurde in den Annahmen festgelegt, dass eine einzelne Ressource genügt, um die volle Dienstfunktionalität aufrecht zu erhalten. Das widerspricht den Anforderungen, wo Dienste unterschiedlich viele Ressourcen benötigen können.
Kandidatenliste	erfüllt	Die als nicht funktionierend markierten Ressourcen bzw. Dienste am Ende des Korrelationsalgorithmus stellen mögliche Fehlerursachen dar.

Tabelle 4.1: Bewertung nach Anforderungen

Die Bewertung in obiger Tabelle zeigt, dass die in der Anforderungsanalyse gestellten Forderungen durch den Ansatz nur zu einem geringen Teil erfüllt werden können. Zusätzlich mussten eine ganze Reihe von Annahmen getroffen werden. Damit ist dieser Ansatz für eine Anwendung in der Praxis zum Beispiel am LRZ nicht

geeignet.

Dennoch konnte der einfache Algorithmus bereits einige Punkte aus dem Anforderungskatalog zumindest teilweise erfüllen. Im nächsten Schritt muss jetzt eine Verfeinerung des Ansatzes erfolgen, um weiteren Anforderungen zu genügen.

Betrachtet man Anforderungen wie das Einbeziehen der Dienstgüteparameter in den Korrelationsprozess oder das Problem der Dienstinstanzen, so wird schnell klar, dass diese nicht erfüllt werden können, ohne weitere Informationen mit einzubeziehen. Diese fehlenden Informationen können entweder aus den Events oder aus der Ressourcen- bzw. Dienstmodellierung bezogen werden. Die nächsten Schritte in der Vorgehensweise werden der weiteren Informationsgewinnung dienen.

4.2 Der Faktor Zeit

Als ersten Schritt der Verfeinerung sollen die Annahmen, die über die Zeit bei der Korrelation getroffen wurden, aufgehoben werden. Bisher wurde die Zeit bei der Vorgehensweise nicht berücksichtigt. Die Einbeziehung der Zeit in den Korrelationsvorgang ist eine wesentliche Voraussetzung für sinnvolle Korrelationsergebnisse, damit einem Zeitstempel nicht nur der absolute Zeitpunkt eines Events berücksichtigt wird, sondern auch die zeitliche Eintreffreihenfolge von Events in das System.

4.2.1 Correlation Window - Zeitfenster

Gerade für die Bearbeitung von Events ist ein Zeitstempel äußerst wichtig. Diese Zeitangaben sollte sich dabei nicht auf das Event selbst beziehen, also zum Beispiel auf den Erstellungs- oder Eintreffszeitpunkt, sondern darauf, wann der Fehler, den das Event beschreibt, bestanden hat. Warum ist dies notwendig? Bei *Resource Events* kann es unter Umständen - zum Beispiel durch Netzwerkprobleme, die ja selbst Events auslösen können - zu einem stark verzögerten Eintreffen am Fehlermanagementsystem kommen. Auch bei *Service Events* ist ein verzögertes Eintreffen möglich. Nutzer können und werden nicht immer sofort, nachdem sie einen Fehler bemerken, bei einer Hotline anrufen, welche dann ein *Service Event* generieren. Für die Fehlersuche ist aber nur interessant, wann ein Fehler bestanden hat. Ein Zeitfenster - auch Correlation Window genannt - kann helfen, die Gültigkeit von Events festzulegen. Es hilft immer nur eine Menge aktiver Events zu betrachen, nämlich diese, deren Fehler innerhalb eines Zeitfensters liegt. Dieses Fenster erstreckt über sich einen konstanten Zeitraum in die Vergangenheit. Liegt der Fehler außerhalb dieses Zeitraums, weil er schon zu lange zurückliegt, so kann das System dieses Event einer gesonderten, möglicherweise manuellen, Behandlung zuführen oder in bestimmten Fällen sogar verwerfen. Die Länge eines solchen Zeitfensters bedarf Diskussion. Diese Dauer ist stark abhängig von der Anzahl der Events die zum Beispiel pro Stunde in das System eintreffen und dem Typ der Events:
Resource Events beziehen sich nicht nur auf die reine Verfügbarkeit einer Ressource. Vielmehr beschreiben die Ereignisse, die Komponenten senden, häufig auftretende Vorkommnisse, die oft nur von sehr kurzer Dauer sind. Zum Beispiel sind gängige Events, die eine hohe CPU oder Link Auslastung melden. So wird klar, dass diesen ein wesentlich kürzeres Zeitfenster zugewiesen werden als man es *Service Events* zuspräche. Für erstere sind Zeiträume im Minuten- oder sogar im Sekundenbereich sinnvoll, hingegen sollten letztere eine längere Zeit als aktiv betrachtet werden.

Die Entwurf solcher global gültigen Zeitfenster ist nur eine Möglichkeit. Eine weitere stellt das Vergeben von Gültigkeitszeiträumen für einzelne Events dar. Mögliche Kriterien für die Festlegung dieser individuellen Zeitfenster sind die jeweiligen Zielressourcen, der jeweilige Zieldienst oder die Uhrzeit. Treten für bestimmte Komponenten, insbesondere für bestimmte Ressourcen nur sporadisch Events auf, könnte man für diese Geräte eine kürzere Gültigkeitsdauer festlegen, als man es für *Service Events* von Kunden tun würde. Jene würde man dann mit einer besonders langen Gültigkeitsdauer belegen. Ein großer Nachteil dieser Idee ist, dass hier schnell ein hoher Aufwand entstehen würde, diese Konfigurationen zum einen zu erstellen und zum anderen immer aktuell zu halten.

Weiterhin wurde in [Vaar 02a] ein Event Correlator vorgestellt - ein freies, Plattform unabhängiges Programm, welches Korrelationsmethoden für ein Ressourcenmanagementsystem implementiert. Hier finden sich mehrere

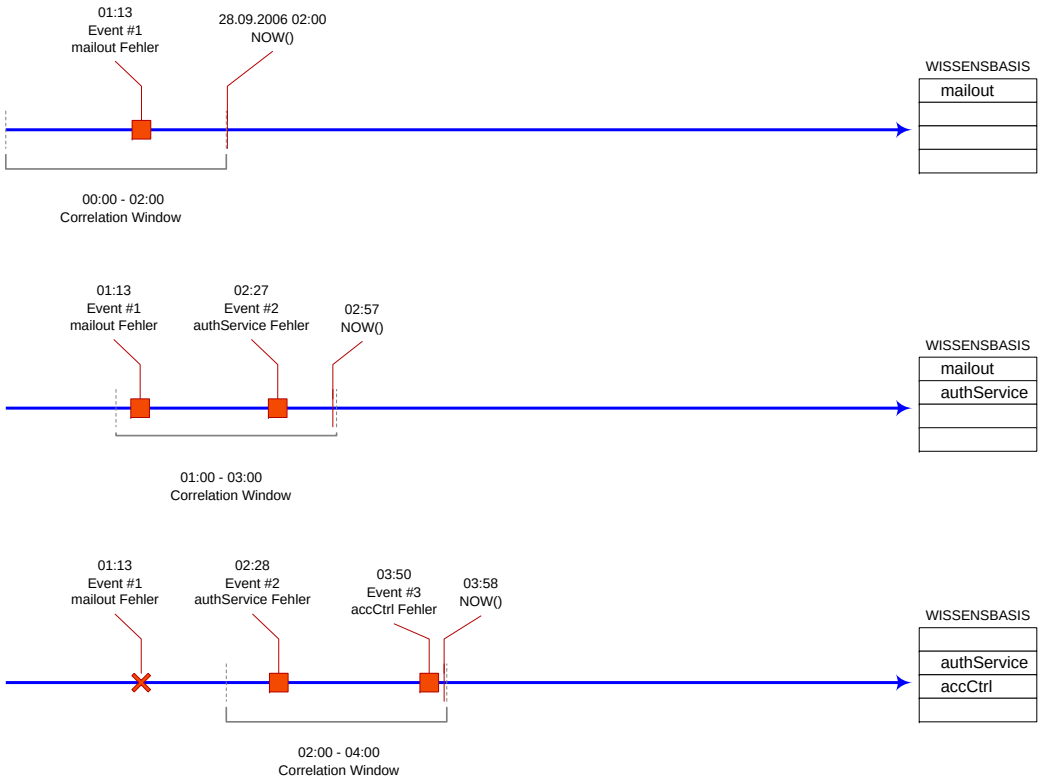


Abbildung 4.3: Zeitfenster für Events

Korrelationsmethoden, wobei einige davon auch auf Zeiträumen basieren. Diese sind Möglichkeiten, die Idee der individuellen Gültigkeitszeiträume umzusetzen und könnten auch mit diesem Ansatz verwendet werden.

4.2.2 Beispiel

Wieder bezogen auf das Beispiel aus dem Abschnitt oben, wird die Modellierung des Webhosting Dienstes am LRZ wie in Abbildung 4.1 auch für dieses Beispiel verwendet. Abbildung 4.3 zeigt drei Zeitstränge mit jeweils unterschiedlichen Zeiträumen. Mit einer Klammer markiert ist jeweils das aktuelle Correlation Window, welches hier auf einen Zeitraum von zwei Stunden festgelegt wurde. Am Ende dieses Zeitfensters ist die aktuelle Uhrzeit zu sehen (*NOW()*). Rechts neben den Zeitsträngen sind die jeweiligen Status der durch Events gemeldeten Dienste oder Ressourcen notiert. Hier wird von einem globalen, statischem Correlation Window, wie oben diskutiert, ausgegangen.

Im ersten Zeitstrang wird angenommen, dass es 02:00 Uhr ist - das Correlation Window reicht also von 00:00 Uhr bis um 02:00 Uhr. Innerhalb dieses Zeitfensters liegt nur ein einzelnes Event: Event #1, ist ein *Service Event* und bezieht sich auf ein Problem beim Senden von E-Mails (*mailout*). Da in diesem Beispiel der E-Mail Zugang nicht modelliert ist, findet hier keine weitere Korrelierung statt. Dieses Event muss hier manuell bearbeitet werden.

Im mittleren Zeitstrang, die Zeit ist derweil um eine Stunde vorgerückt, trat um 02:30 Uhr ein weiteres *Service Event* (Event #2) auf, welches eine Fehlfunktion des *authService* Dienstes bescheinigt. Der Korrelationsalgorithmus, wie oben vorgestellt, kann mit diesem einzelnen Event eine Kandidatenliste mit den möglichen Fehlerquellen *htpasswd* und *htaccess* liefern.

Im letzten, unteren Zeitstrang, beschreibt das Correlation Window den Zeitraum zwischen 02:00 Uhr und 04:00 Uhr. Das *mailout* Event ist bereits aus dem Zeitfenster heraus und erscheint deswegen auch nicht mehr in der Wissensbasis rechts. Neu hinzugekommen ist Event #3, ein *Service Event*, welches den Ausfall des *accCtrl* Dienstes, also den Zugriff auf geschützte Webseiten, beschreibt. Dieses Event wird mit Hilfe des Algorithmus mit dem Event #2 korreliert. Die Kandidatenliste ändert sich nicht.

Das Beispiel zeigt, dass die Reihenfolge, in der Events in das Fehlermanagementsystem eintreffen, eine wichtige Rolle spielt. Wäre Event #3 zuerst eingetroffen, wäre eine längere Kandidatenliste entstanden, da das System zu diesem Zeitpunkt noch keine Informationen über den Status der vom *accCtrl* Dienst abhängigen Ressourcen gehabt hätte.

4.2.3 Klassendiagramm

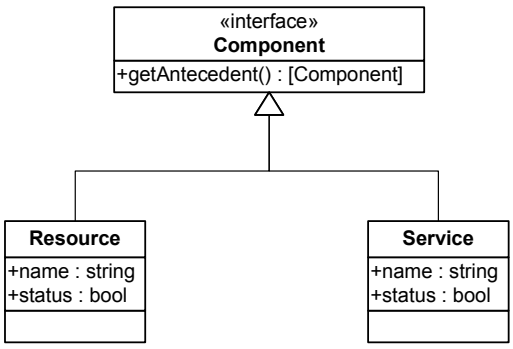


Abbildung 4.4: UML Beschreibung von Komponenten

Um diese Änderungen zusammenzufassen, bietet es sich an, die beteiligten Komponenten, also Dienste, Ressourcen und Events in einem Klassendiagramm nach der *Unified Modeling Language* (UML) darzustellen.

Abbildung 4.4 beschreibt die Attribute und Methoden eines Dienstes oder einer Ressource. Beide Klassen implementieren eine Schnittstelle *Component*, welche die Methode *getAntecedent()* vorschreibt. Diese Methode gibt, in der Klasse *Resource* oder *Service* implementiert, rekursiv alle Komponenten zurück, von denen die Komponente abhängt. Diese Information kommt aus der Abhängigkeitenmodellierung. Die Klassen tragen die Attribute *name*, welches den (eindeutigen) Bezeichner einer Komponente speichert, und *status*, welcher in einer booleschen Variable den aktuellen Status der Komponente widerspiegelt.

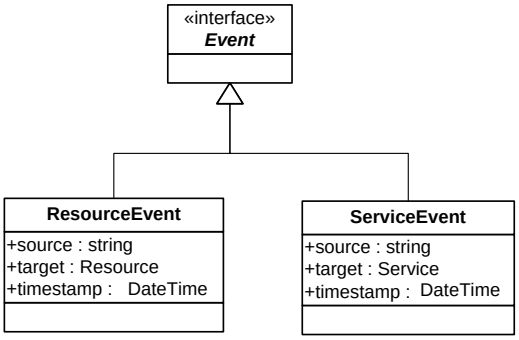


Abbildung 4.5: UML Beschreibung von Events

In der Abbildung 4.5 werden Ressourcen und *Service Events* beschrieben. Die Schnittstelle *Event* dient als Basis für Service und Ressourcen Events. Bis jetzt trägt ein Event, gleich ob Ressourcen- oder Service- nur wenige Informationen. Die *source* bezeichnet die Ereignisquelle. Das kann beispielsweise ein Benutzer sein bei *Service Events* oder ein bestimmtes Gerät bei einem *Ressource Event*. Da diese Quelle in dieser Arbeit nicht näher spezifiziert wird, wurde als Datentyp ein *string* gewählt. Das Attribut *target* speichert die Ressource oder den Dienst, auf den sich das Event bezieht und ist daher entpsrechend vom Typ *Resource* oder *Service*. Letztendlich werden im *timestamp* Attribut die Zeit hinterlegt, auf die sich das Event wie oben beschrieben bezieht.

4.2.4 Zusammenfassung

Die Einführung der Zeit ist ein wichtiger Schritt zur Verbesserung der Korrelationsergebnisse. Wie sich im weiteren Verlauf dieses Kapitels (in 4.4) zeigen wird, können mit der Zeitinformation weitere auftretende Probleme gelöst werden. Weiterhin wurde in diesem Abschnitt eine UML Klassenstruktur begonnen, die mit den folgenden Verfeinerungsschritten der Vorgehensweise noch erweitert werden wird.

Die Annahme der Zeitunabhängigkeit wird mit einem Zeitstempel bei den Eventklassen aufgehoben. Ein sequentielles Auftreten von Ereignissen ist nun möglich, wie im Beispiel gezeigt wurde. Es stehen fast beliebige zeitbasierte Korrelationsmethoden, wie auch in [Vaar 02a] vorgestellt, zur Verfügung. Die Einführung einer globalen Gültigkeitsdauer für Events ist nur eine Möglichkeit.

Ein Fall, der bei *Service Events* auftreten könnte ist, dass Benutzer Fehlverhalten eines Dienstes über einen Zeitraum hinweg melden könnten. *„Der Zugriff auf dynamische Seiten ging gestern den ganzen Abend nicht.“* wäre so eine Aussage. Liegt dieser Zeitraum ausserhalb des aktuellen Correlation Windows, das heißt der betroffene Dienst hat mittlerweile wieder seine Funktionalität erreicht, muss manuell eine Entscheidung gefällt werden, wie mit diesem Ereignis umgegangen werden soll. Liegt zumindest ein Teil des gemeldeten Zeitraumes innerhalb des aktuellen Zeitfensters, so kann der späteste Zeitpunkt, der noch innerhalb des aktuellen Zeitfensters liegt, als Zeitpunkt des Events gewählt und normal weiter behandelt werden. Eine alternative Behandlungsweise, wäre die Verwerfung des Events. Diese ist jedoch nicht zu empfehlen, da unter Umständen später auf Eventinformationen zurückgegriffen werden muss.

Tabelle 4.2 listet die erfüllten Anforderungen ein Ergänzug zu Tabelle 4.1 auf.

Anforderung	erfüllt	Erläuterung
Messen der Dienstgüteparameter	erfüllt	Dienstgüteparameter werden in dieser Erweiterung modelliert. Es können Grenzwerte festgelegt werden. *Service Events* können diese Informationen jetzt weitergeben.
Priorisierung von Ereignissen	teilweise erfüllt	Die individuelle Möglichkeit, die Zeitdauer von Events festzulegen, ermöglicht es, Events höherer Priorität länger zu den aktiven Events zu zählen.

Tabelle 4.2: Bewertung der Anforderungen nach Einführung des Correlation Window

Dienst	Klasse	QoS Parameter
Webhosting	Nutzung	Datendurchsatz beim Zugriff auf den Webspace über FTP (z.B. mindestens 100kb/s)
Webhosting	Management	Maximale Zeit, die das Supportteam braucht, um auf eine E-Mail Anfrage zu reagieren (z.B. in Tagen)
Webhosting	Management	Maximale Zeit, die benötigt wird, um eine Anforderung zur Passwortänderung umzusetzen (z.B. maximal eine Stunde)
E-Mail	Nutzung	Fehlerquote, die ein Spamfilter für eingehende E-Mails aufweist (Fehlerquote: Anzahl der fälschlicherweise als Spam eingestuften E-Mails / Gesamtzahl der E-Mails)
E-Mail	Nutzung	Zeit, die E-Mails benötigen, um vom Mailrelay versendet zu werden (z.B. in Minuten)

Tabelle 4.3: Beispiele für QoS Parameter

4.3 Dienstgüteparameter und Dienstfunktionalitäten

Dieser Abschnitt konzentriert sich stark auf die Dienstorientierung der Vorgehensweise. Ziel ist es die Annahme, dass ein Dienst nur die binären Status „funktioniert“ und „funktioniert nicht“ einnehmen kann, aufzuheben. Die Unterscheidung zwischen einzelnen Dienstfunktionalitäten soll helfen, den Status von Diensten feiner abstufen zu können. Die Grenzwerte von Dienstgüteparametern, die ja auch in SLAs eine wichtige Rolle spielen (siehe Abschnitt 1), werden in diesem Schritt modelliert. Zudem wird festgelegt, wie das Fehlermanagementsystem auf Überschreitungen dieser Grenzwerte reagieren kann.

4.3.1 QoS - Dienstgüteparameter

QoS Parameter sind die messbaren Werte, anhand welcher die Erfüllung von Verträgen bestimmt werden. Der Provider sollte also ein besonderes Augenmerk auf die Einhaltung der jeweiligen Grenzwerte legen. Diese Parameter können sich auf Nutzungs- und Managementfunktionalitäten von Diensten beziehen. Tabelle 4.3 listet einige mögliche Qos Parameter für unterschiedliche Dienste auf.

Es existieren also zahlreiche mögliche QoS Parameter. Damit ein Fehlermanagementsystem diese überwachen kann, müssen sie zunächst modelliert werden. Diese Modellierung erfolgt in zwei unterschiedlichen Komponenten. Zum einen muss festgelegt werden, welcher Dienst welche QoS Parameter einzuhalten hat, und zum anderen müssen die Grenzwerte für jeden Parameter festgelegt werden. Da unterschiedliche QoS Parameter bei jedem Dienst einen anderen Grenzwert erreichen dürfen, müssen diese Werte für jeden Dienst eigens modelliert werden.

4.3.2 Dienstfunktionalitäten

Nach dem MNM Dienstmodell, wie in Abschnitt 3.2 beschrieben, ist vorgesehen, die Funktionalitäten von Diensten in Klassen zu unterteilen. Dabei schlägt das MNM Dienstmodell die beiden Klassen *Nutzung* und

Management vor. Wie schon [Marc 06] in ihrer Arbeit feststellte, ist diese grobe Unterteilung nicht immer ausreichend, obgleich die prinzipielle Unterteilung eines Dienstes in seine einzelnen Funktionalitäten ein wichtiger Schritt für die Modellierung eines Szenarios ist. In der Abhängigkeitenmodellierung von [Marc 06], wie in 3.3 beschrieben, werden die Abhängigkeiten mit einem Design Pattern aus der Softwareentwicklung dargestellt. Diese Modellierung wird auch für diese Arbeit verwendet.

Mit dieser Modellierung ist es möglich, einen Dienst nach der Abhängigkeitenmodellierung in seine Funktionalitäten aufzuteilen. Ein Fehlermanagementssystem kann damit sowohl den Dienst in seiner Gesamtheit, als auch bestimmte näher definierte Funktionalitäten einzeln betrachten. Diese Unterscheidung kann sich auch auf den Status eines Dienstes auswirken, der jetzt eben nicht mehr auf den ganzen Dienst, sondern auf eine spezielle Funktionalität eines Dienstes bezogen werden kann. Auch auf der Ressourcenebene ist dieser Ansatz anwendbar. Ressourcen werden zwar nicht in Funktionalitäten unterteilt, dennoch existieren Abhängigkeitshierarchien auch hier.

Um eine Unterscheidung nach Dienstfunktionalitäten jedoch überhaupt treffen zu können, müssen weitere Informationen von der Quelle der *Service Events* eingeholt werden. Die Quelle, also in der Regel ein Nutzer, muss genauere Angaben über die vom Fehler betroffene Funktionalität machen - die Angabe des betroffenen Dienstes alleine reicht also nicht immer aus. Diese Präzisierung muss jedoch optional bleiben, da Nutzer oft keine ausreichenden Detailkenntnisse über den Dienst, den sie nutzen, verfügen. Das *Customer Service Management* muss versuchen, diese Informationen durch systematisierte Fragestellung zu ermitteln. Ein interessanter Ansatz dazu ist der von [RGD 98] vorgestellte Intelligent Assistant.

Eine weitere entscheidende Voraussetzung, um einen Nutzen aus den gewonnenen Informationen ziehen zu können, ist die präzise Modellierung der Abhängigkeiten. Je detaillierter das Modell ist, um so feiner können Dienste in Funktionalitäten unterteilt werden. Das verwendete Design Pattern erlaubt eine beliebige Verschachtelung. Wie tief diese Verschachtelung ist, liegt ganz in der Hand des Providers. Er muss auch die Abwägung zwischen Mehraufwand bei der Modellierung und besseren Korrelationsergebnissen durchführen.

4.3.3 Erweiterung des Klassendiagramms

4.3.3.1 Modellierung der QoS Paramter

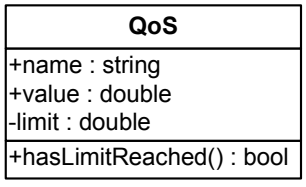


Abbildung 4.6: Klassendiagramm der Klasse QoS

Um eine Modellierung wie oben vorgestellt möglich zu machen, muss die Klassenstruktur aus Abbildung 4.4 erweitert werden. Die Beschreibung von Events bleibt von dieser Erweiterung unberührt und wird in späteren Abschnitten nachgereicht.

In Abbildung 4.6 wird eine neue Klasse zur Abstraktion der QoS Parameter eingeführt. Die Klasse *QoS* repräsentiert einen bestimmten QoS - Parameter und wird später direkt bei der Klasse Service gespeichert werden. Das Attribut *limit* legt dabei den speziellen Grenzwert fest, den der Parameter im Bezug auf diesen Dienst erreichen darf. Es wurde als *private* deklariert, da diese Informationen nach außen hin nicht sichtbar sein muss, um die Korrelation durchführen zu können. Der *value* Wert speichert den jeweils aktuellen Wert des

Parameters, *name* identifiziert diesen Parameter. Mit der Methode *hasLimitReached()* kann das Fehlermanagementsystem feststellen, ob der aktuelle Wert des Parameters den Grenzwert überschritten hat und somit entsprechend den Status des Dienstes.

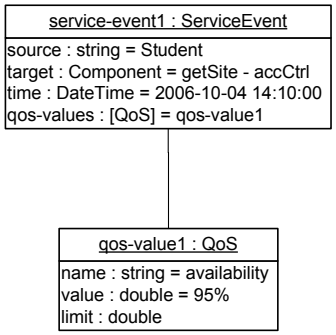


Abbildung 4.7: Service Event

Auch bei *Service Events* muss eine Erweiterung stattfinden, um die Modellierung der QoS Parameter nutzen zu können. Diese Events müssen dahingehend erweitert werden, dass sie auch Informationen über aktuelle QoS Parameter tragen können. Abbbildung 4.7 zeigt diese Erweiterung. Das neue Attribut *qos-values* ist ein Array gefüllt mit Objekten der Klasse QoS (siehe Abbildung 4.6). Dabei sind jedoch nur die *name* und *value* Attribute belegt. Ist dieses Array gefüllt, muss das Korrelationssystem selbst mit der Methode *hasLimitReached()* entscheiden, ob ein Fehler vorliegt bzw. ob der Status der jeweiligen in *target* angegebenen Funktionalität verändert werden muss.

4.3.3.2 Modellierung der Dienstfunktionalitäten

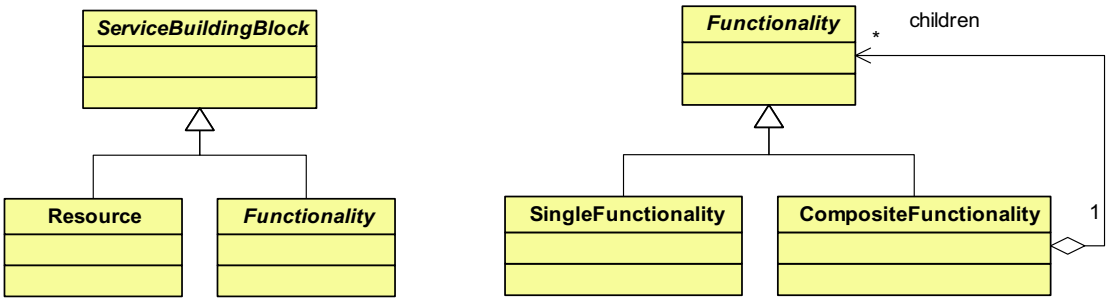


Abbildung 4.8: Die Klasse *ServiceBuildingBlock* aus der Abhängigkeitenmodellierung von [Marc 06]

Wie bereits weiter oben beschrieben wird in der Abhängigkeitenmodellierung von [Marc 06] das Composite Design Pattern aus der Softwareentwicklung verwendet. Nach [EG 95] kann dieses Pattern Teil-Ganzes Hierarchien darstellen. Die Einführung dieses Entwurfsmusters erfordert eine Umgestaltung der Modellierung der oben als Komponenten bezeichneten Struktur der Ressourcen und Dienste.

Wie in 3.3 beschrieben, spielt die Klasse *ServiceBuildingBlock* bei der Abhängigkeitenmodellierung eine besondere Rolle. Sie repräsentiert sowohl Dienste und deren Funktionalitäten, als auch beteiligte Ressourcen. Abbildung 4.8 zeigt die Definition aus [Marc 06]. Die abstrakte Klasse *Component* aus Abbildung 4.4 kommt der Idee der ServiceBuildingBlock Klasse relativ nahe. Diese muss mit den bereits eingeführten Attributen und Methoden zu erweitert werden. Dies ist notwendig, um der Anforderung Dienste in Funktionalitäten unterteilen zu können, und um die Vorteile, die die Abhängigkeitenmodellierung bietet, nutzen zu können.

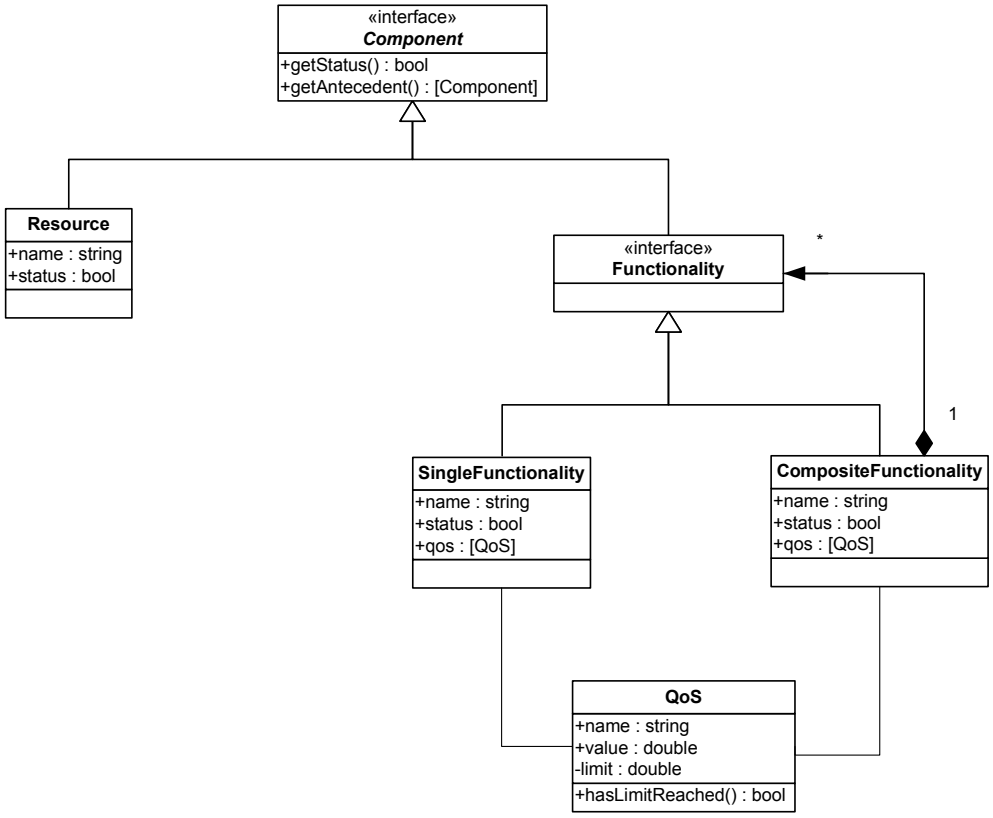


Abbildung 4.9: Modellierung von *Components* und der QoS Klasse

Abbildung 4.9 zeigt diese Erweiterung. Zur besseren Übersichtlichkeit wurden nur die Attribute und Methoden eingefügt, die für eine Korrelation unbedingt notwendig sind. So wurden auch die entsprechenden Eigenschaften, die das Entwurfsmuster mit sich bringt, weggelassen. Die bereits oben vorgestellten Attribute tauchen zum einen in der soweit unveränderten Klasse *Resource* und zum anderen in der Klasse *SingleFunctionality* auf. Diese Klasse erhält zusätzlich ein Feld mit Elementen der oben vorgestellten Klasse *QoS*. Hier werden die für diese Funktionalität spezifischen QoS Parameter und deren Grenzwerte festgehalten. Die Schnittstelle *Component* trägt die neue Methode *getStatus()*, die in den Klassen Resource und SingleFunctionality implementiert einfach nur den Status der Ressource zurückgibt, während sie in der Klasse CompositeFunctionality den logisch mit „UND“ verknüpften Status aller untergeordneten Funktionalitäten zurück gibt. Um auch für zusammengesetzte Dienste QoS Parameter angeben zu können, findet auch hier ein Feld mit *QoS* Elementen Platz.

4.3.4 Beispiel

Ein Beispiel, wie ein Dienst in mehrere Funktionalitäten aufgeteilt werden kann, ist in Tabelle 4.4 zu sehen. Zur Demonstration wird wie in den Beispielen oben der Dienst „Webhosting“, wie ihn das LRZ anbietet, zunächst in die Funktionalitätsklassen Management und Nutzung und schließlich in einzelne Funktionalitäten unterteilt. Tabelle 4.4 erhebt keinen Anspruch auf Vollständigkeit.

Klasse	**Funktionalität**
Nutzung	Statische bzw. dynamische Seiten abrufen
Nutzung	Zugriff auf einen geschützten Bereich
Nutzung	Zugriff auf eine Datenbank
Nutzung	Zugriffsrechte auf bestimmte Ordner
Nutzung	Webmail Zugang
Management	Kennwort des FTP-Zugangs ändern / festlegen
Management	Order- oder Dateizugriffsrechte ändern / setzen
Management	Administration einer Datenbank
Management	(Sub-) Domains verwalten

Tabelle 4.4: Unterschiedliche Dienstfunktionalitäten des Webhosting Dienstes

Zur Demonstration soll die in Abbildung 4.2 dargestellte Modellierung des Webhosting Dienstes um weitere Funktionalitäten erweitert bzw. unterteilt werden. Zur einfacheren Darstellung und der besseren Übersicht wegen wird das Beispiel auf den Zugriff auf einen geschützten Bereich (*getSite - accCtrl*) beschränkt. Abbildung 4.10 zeigt diese beispielhafte Modellierung. Der DNS Dienst, jetzt als CompositeFunctionality dargestellt, ist in zwei Unterfunktionalitäten aufgeteilt: *DNSMWN*, die die DNS Funktionalität des Münchner Wissenschaftsnetztes und *DNSLRZ*, die die DNS Funktionalität des LRZ repräsentieren soll. Diese beiden Objekte besitzen jeweils QoS Objekte: *dnsmwn-1* und *dnslrz-1* stehen für die Zeit, die der Resolver des DNS Dienstes benötigt bzw. benötigen darf. Hier werden nur Beispielwerte verwendet.

Die zwei virtuellen Server *virt1* und *virt2* stellen zwei Ressourcen dar, welche in dem Beispiel die durch Passwort geschützten Seiten zur Verfügung stellen. Für eine dieser Ressourcen hat ein Ressourcenüberwachungsprogramm ein *Resource Event* generiert, da die Ressource *virt1* ausgefallen ist. Dieses Event ist ebenfalls in Abbildung 4.10 zu sehen.

Die zweite zusammengesetzte Funktionalität ist *accCtrlApache*, welche für die Zugriffsschutzfunktionen eines Apache Webservers (vgl. [Apac06]) steht. Hier wird zwischen der *htaccess* und der *htpasswd* Funktion unterschieden. Für diese beiden Objekte sind keine QoS Parameter definiert.

Ein *Service Event*, welches angenommen einer QoS Überwachung am LRZ entstammt, beschreibt eine Überschreitung eines QoS Parameters, nämlich der minimalen Verfügbarkeit für den Zugriff auf einen geschützten Bereich des Webhostingdienstes. Eine Darstellung dieses Events zeigt Abbildung 4.11. Diese Verzögerung bei der Namensauflösung könnte insgesamt zu Problemen bei dem Zugriff auf einen geschützten Bereich führen.

Da in der bisherigen Modellierung immer mit der Annahme, dass ein Ressourcen- oder Dienststatus nur „funktioniert“ oder „funktioniert nicht“ annehmen kann, führt dieses Service Event zu einer Änderung des Status

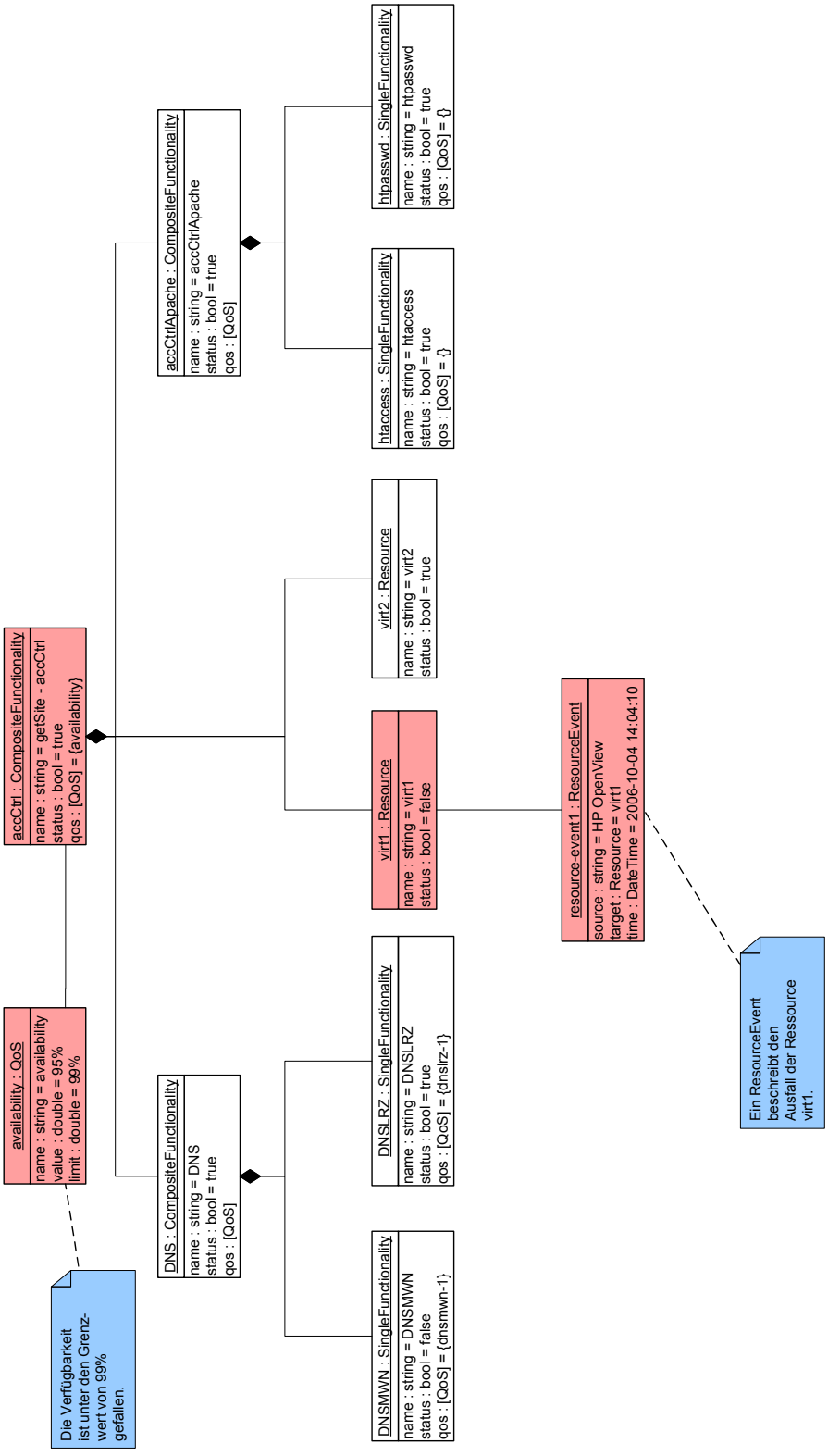


Abbildung 4.10: Beispielhafte Modellierung des Zugriffs auf geschützten Bereich

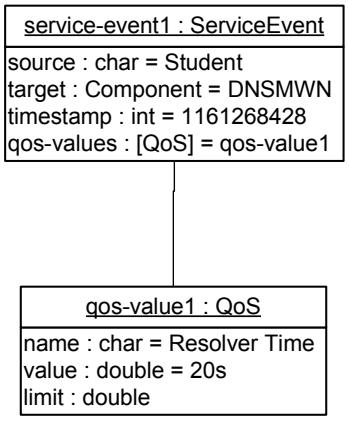


Abbildung 4.11: Beispiel für ein Service Event

Anforderung	**erfüllt**	**Erläuterung**
Ressourcen- und Dienststruktur	erfüllt	Ein vorhandenes Abhängigkeitenmodell kann jetzt durch die neue Modellierung der Dienste und Ressourcen präziser angewendet werden.
Messen der Dienstgüteparameter	erfüllt	Dienstgüteparameter werden in dieser Erweiterung modelliert. Nicht nur Dienste oder Funktionalitäten, sondern auch Events tragen Informationen über QoS Parameter.

Tabelle 4.5: Bewertung nach Anforderungen nach Einführung von QoS Parametern und Dienstfunktionalitäten

der *getSite - accCtrl* Funktionalität.

Erreichte das System ein Resource Event, welches die angesprochenen Probleme beim Nutzen des *accCtrl* Dienstes beschriebe, könnte das System dieses neue Event mit dem vorhergehenden korrelieren. Der Vorteil dieser Verfeinerung liegt auf der Hand: Die wahrscheinliche Fehlerursache für dieses neue Event wäre schnell gefunden.

4.3.5 Zusammenfassung und Bewertung

Die Unterteilung von Diensten in ihre (Sub-) Funktionalitäten erlaubt eine weitaus differenziertere Modellierung eines Szenarios, als es mit dem vorherigen Ansatz möglich war. Dabei ist die Verfeinerung immer optional. Das verwendete Entwurfsmuster erlaubt eine sehr grobe, wie auch eine sehr feine Modellierung. Dienste können sowohl als zusammengesetzte Komponenten, als auch einzeln betrachtet werden.

Die Einführung von QoS Parametern bei Diensten ist enorm wichtig, wenn es darum geht vertraglich festgelegte (SLA) Dienstparameter zu überwachen. Die Einführung dieser Parameter auch bei *Service Events* erfordert, dass von den Nutzern / Kunden weitere Informationen abgefragt werden müssen.

Mit diesen Erweiterungen werden die Anforderungen in Tabelle 4.5 in Ergänzung zu Tabelle 4.1 erfüllt.

Weiterhin wurde die Annahme über Events, dass diese sich immer nur auf einen einzelnen Dienst oder eine einzelne Ressource beziehen, aufgehoben. Durch die feinere Modellierungsmöglichkeiten können sich Ereignisse auf gesamte Dienste oder deren (Sub-)Funktionalitäten beziehen. Damit ist ein wichtiger Schritt getan,

da nun sowohl abstrakte Events einen gesamten Dienst betreffend als auch Events, die sehr spezifisch eine Teilfunktionalität betreffen, modelliert werden können.

4.4 Ereigniskorrelation mit Score

In diesem Schritt der Verfeinerung des Ansatzes soll zunächst die Annahme, dass Ressourcen oder Dienstfunktionalitäten nur einen binären Status einnehmen können, aufgehoben werden. Um dies zu erreichen wird eine zusätzliche Variable eingeführt. Diese Variable kann Werte zwischen 0 und 1 annehmen und wird im Folgenden als *Score* bezeichnet.

4.4.1 Score für Dienste, Ressourcen und Events

Diese „Score“ ersetzt also den neuen Status einer Komponente. Die maximale Score, die eine Komponente erreichen kann ist 1, was gleich zu setzen ist mit dem alten Status *false*. Diese Komponente wird also sicher nicht funktionieren. Der Scorewert ist also eine Abbildung des Verfügbarkeitsstatus auf einen Wertebereich $[0...1]$.
Betrachtet man zusammengesetzte Funktionalitäten, bedeutet eine hohe Score einen Ausfall vieler Subfunktionalitäten. Um diese Informationen auch in das System zu bringen, ist es nötig, die Scoreinformation auch für Events einzuführen. Bei der betroffenen Zielkomponente werden die Score Werte aus den Events bis zum Maximum 1 akkumuliert. Eine entsprechende Wahl der Score erlaubt hier sehr präzise Abstufung, da keine Beschränkung auf diskrete Zwischenwerte besteht.

Stattet man Events mit einem weiteren Scorewert aus, bietet sich eine Möglichkeit der Priorisierung von Ereignisquellen. Eine Quelle mit hoher Priorität, zum Beispiel *Service Events* von Nutzern, treffen mit einer hohen Score in das System ein, während zum Beispiel automatisch generierte Meldungen von Ressourcenüberwachungssystemen eine entsprechend niedrigere Score tragen.

Mit Einführung dieses Wertes kann nun eine Anstoßung des Korrelationsvorganges gesteuert werden. Die Akkumulierung der Score bei den jeweiligen Diensten oder Ressourcen ermöglicht es, einen Grenzwert einzuführen, dessen Überschreitung die Korrelation und damit die Fehlersuche auslöst. Ein guter Wert ist die maximale Score, die ein Element erreichen kann - also 1. So lösen *Service Events* direkt von Benutzern stammend, die entsprechend einen Score von 1 tragen, auch sofort eine Korrelation aus. Es obliegt dem Dienstprovider, welchen Service oder Resource Events er welchen Score zuweist. Möglich wäre beispielsweise die Score $\frac{1}{n}$, wenn man erst nach n Events eine Fehlersuche beginnen möchte.

Zur Steigerung der Performanz des Fehlermanagmentsystems können Events, die sich auf Komponenten beziehen, deren Score bereits den Grenzwert überschritten haben, verworfen oder aber ohne Korrelation abgelegt werden.

Abbildung 4.12 veranschaulicht das eben Genannte. In einer Wissensbasis sind Dienste und Ressourcen gespeichert. Events mit entsprechenden Scorewerten treffen auf das System. In der Wissensbasis werden die Scorewerte bis zum Maximum von 1 akkumuliert. Ist dieses Maximum erreicht, wird der Korrelationsvorgang ausgelöst und eine Liste mit Fehlerkandidaten erstellt. Treffen weiterhin Events für eine Komponente ein, dessen Score Wert bereits das Maxiumum erreicht hat, müssen diese nicht mehr vom System bearbeitet werden. In der Abbildung werden die Scorewerte durch Kreise, Ereignisse durch Kästen mit abgeflachtenEcken dargestellt.

4.4.1.1 Grundscore

Die Speicherung einer Score für jeden Dienst und jede Ressource führt zu einer weiteren Verfeinerung. Provider können bestimmten Komponenten eine „Grundscore“ zuweisen. Standardmäßig hat jede Komponente die Grundscore 0. Mit einer Vorbelegung des Score Wertes größer als 0 lassen sich so Dienste oder insbesondere

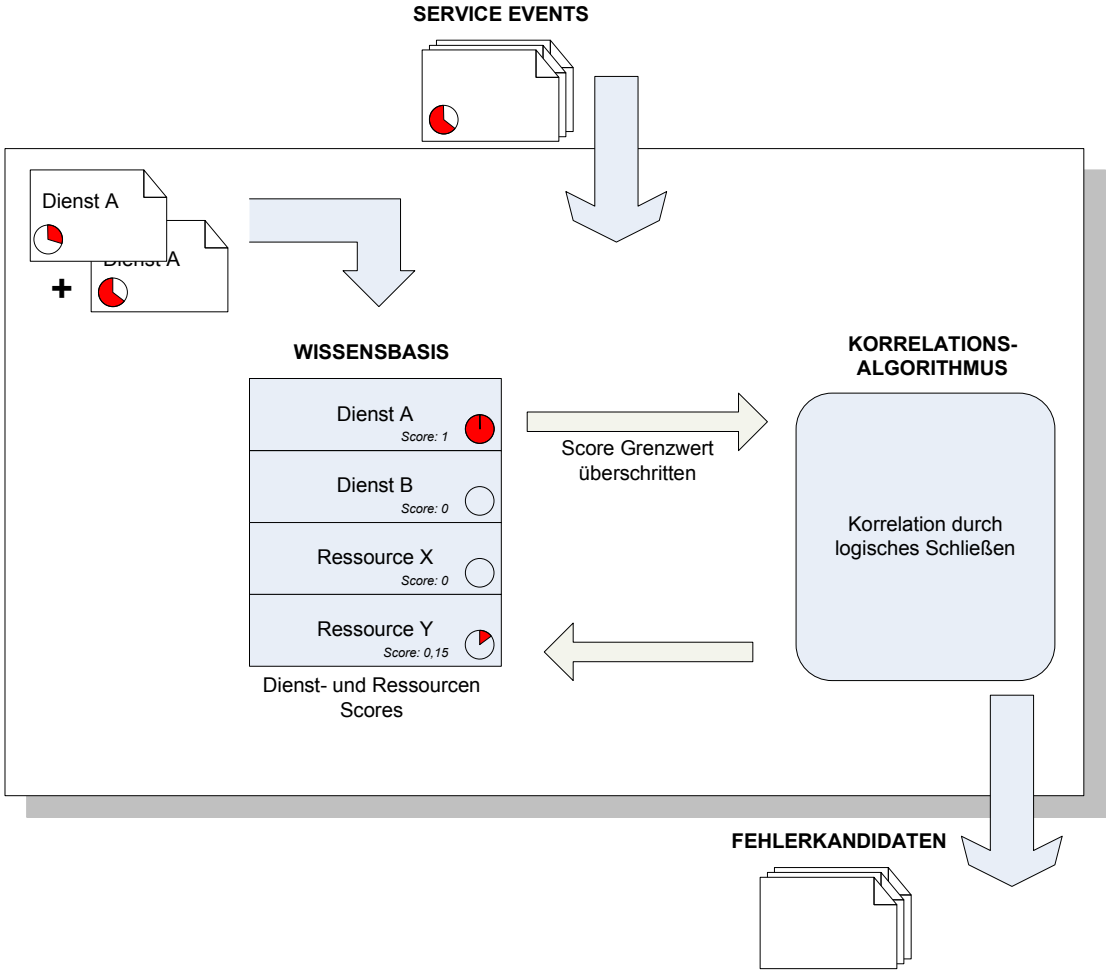


Abbildung 4.12: Score für Dienste, Ressourcen und Events

Ressourcen, die eine erfahrungsgemäß hohe Fehlerquote aufweisen, von anderen Komponenten unterscheiden. Der Grenzwert, der eine Korrelation anstößt, wird hier schneller erreicht. Das bedeutet auch, dass weniger Events notwendig sind, um diesen Wert zu erreichen. Die Zuweisung dieser Grundscore stellt nur eine Option dar und ist nicht zwingend notwendig. Eine Verwendung setzt detaillierte Kenntnisse der Infrastruktur des Providers voraus, da Änderungen an der Grundscore einen wesentlichen Eingriff in die Vorgehensweise der Fehlersuche bedeutet.

Beispiel Ein gutes Beispiel am LRZ ist, wenn es bei den Verantwortlichen am LRZ bekannt wäre, dass der NFS Dienst im Vergleich zu den anderen Diensten unzuverlässig ist. Die Zuweisung einer Grundscore von beispielsweise 0.5 würde schon nach dem Eintreffen eines Resource Events (bei entsprechender Vorbelegung von Resource Events) eine Meldung des Fehlermanagementsystems anstoßen. Modelliert man die Ressourceninfrastruktur so detailliert, dass sich die Abhängigkeitenmodellierung auch auf die (Hardware-) Komponenten einzelner Rechner bezieht, könnte man besonders anfälligen, überwachten Komponenten von Rechnern, wie Festplatten oder Lüftern, eine erhöhte Grundscore zuweisen.

4.4.1.2 Score und QoS

Um die Auswirkungen, die eine Überschreitung eines Grenzwerts eines Dienstegüteparameters bewirken soll, modellieren zu können, sollte die Score des betroffenen Dienstes erhöht werden. Je nach Parameter, abgeschlossenem SLA, Wert des Parameters und natürlich Zielkomponente, muss der Provider entscheiden, wie die Score anzuheben ist. Gelten für einen bestimmten Parameter bezogen auf einen bestimmten Dienst mehrere verschiedene SLAs, so müssen vom Provider mehrere Aspekte berücksichtigt werden.

- Eine Lösung wäre es, immer die Grenzwerte des „schärfsten“ Vertrages gelten zu lassen. Diese Vorgehensweise garantiert dem Provider eine sichere Einhaltung aller abgeschlossenen SLAs. Jedoch birgt sie auch Nachteile: Hat der Provider nur sehr wenige Kunden mit einem scharfen SLA im Vergleich zu anderen Kunden, muss er einen unverhältnismässigen Aufwand betreiben, die Verträge zu erfüllen.
- Eine alternative Möglichkeit wäre, die Grenzwerte aus der Gesamtheit der Verträge zu mitteln und diesen Mittelwert für die Überwachung zu verwenden. Dabei muss aber in Kauf genommen werden, dass unter bestimmten Umständen Vertragsverletzungen in Kauf genommen werden müssen.
- Letztendlich bleibt dem Provider noch die Möglichkeit für einzelne Komponenten individuelle Festlegungen zu treffen, je nach dem wie er die Wichtigkeit der Komponente einstuft.

Zusammenfassend bleibt es eine Angelegenheit der Unternehmenspolitik des Providers, welche Prioritäten er setzt oder wie stark er versucht alle Verträge einzuhalten.

Im Gegensatz zu Events, die *keinen* Bezug zu QoS Parametern aufweisen, wird die Erhöhung der Score der Zielkomponente bei Ereignissen, die einen QoS Paramter tragen (und natürlich auch deren Zielkomponente) ausschliesslich durch den Wert des Parameters bestimmt. Welche Auswirkungen dies auf die Modellierung der Komponenten und Events hat wird weiter unten betrachtet.

Beispiel Welche Auswirkungen die Verbindung zwischen Score und QoS Paramter mit sich bringt, soll dieses kurze Beispiel demonstrieren. Moniert ein Kunde beispielsweise einen zu niedrigen / zu hohen Wert eines bestimmten QoS Parameters bei Nutzung eines bestimmten Dienstes, so muss hier nicht unbedingt eine Fehlersituation vorliegen. Erst nach Abgleich mit dem gültigen SLA kann festgestellt werden, ob hier wirklich ein Fehler vorliegt, oder aber ob der Kunde die Situation nur subjektiv als Fehler (Symptom) empfunden hat.

4.4.1.3 Scoreregression

Akkumuliert man die Score einer Komponente, so benötigt man eine Möglichkeit den Scorewert reduzieren zu können, da sonst eine einmal erhöhte Score nicht wieder auf ihren Ursprungswert zurück gesetzt werden würde, obwohl der Grund für diese Erhöhung nicht mehr vorliegt.

Eine Möglichkeit ist ein manuelles Zurücksetzen der Score einer betroffenen Komponente, wenn der gemeldete Fehler nicht - oder nicht mehr - vorliegt. Um wiederholtes Warten auf Benutzereingaben zu vermeiden, ist diesem manuellen Eingriff automatisierten Varianten der Scoreregression vorzuziehen. Eine davon ist die bereits im Abschnitt 4.2 vorgestellte Gültigkeitsdauer von Events. Läuft dieser Zeitraum, der sowohl durch ein Correlation Window, oder durch individuelle Festlegung bestimmt wurde, ab, so reduziert man die Score der Komponente wieder auf ihre Grundscore.

Eine andere Variante ist die Einbeziehung von positiven Events. Solche Events bescheinigen einer Komponente wieder ihre Funktion und bieten eine Möglichkeit die Score wieder zu regressieren. Anwendung findet dieses Konzept zum Beispiel in der Tivoli Enterprise Console (siehe Abschnitt 3.5.2.2).

Die Kombination von Zeit und Score kann noch in einem anderen Zusammenhang angewendet werden: Insbesondere bei Ressourcen ist es interessant zu wissen, wenn bestimmte Events innerhalb eines Zeitraumes sehr häufig auftreten, während ein nur sporadisches Auftreten nicht zu beachten ist. Das würde man erreichen durch die Konfiguration eines kurzen Gültigkeitszeitraumes. Die Score dieser Events setzt man entsprechend auf $\frac{1}{n}$, wenn man erst nach n Events innerhalb eines Zeitraumes ein Problem vorliegt. Eine Anwendung sind Events, die zum Beispiel die hohe Auslastung eines Links anzeigen. Eine sporadische Erhöhung der Auslastung stellt zunächst kein Problem dar, jedoch eine dauerhafte Überlastung kann zu Problemen in Diensten führen, die von diesem Link abhängen.

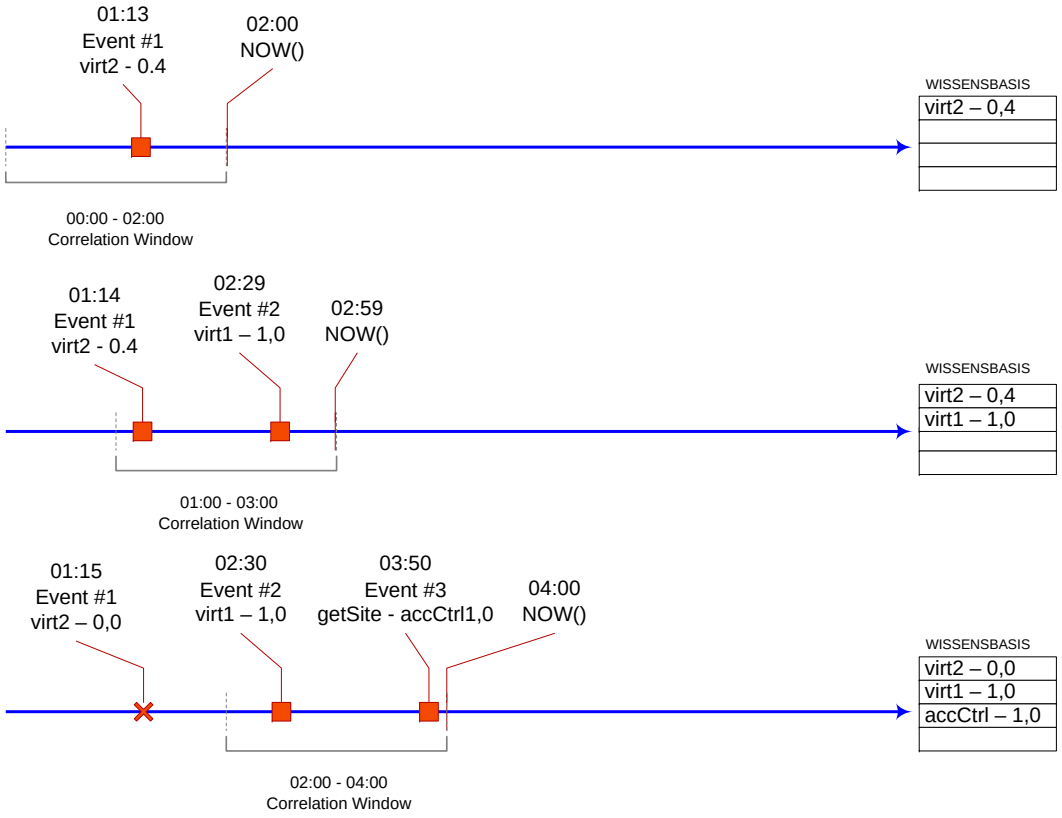


Abbildung 4.13: Automatische Scoreregression durch Zeit

Beispiel Es wird wieder die Modellierung aus Abbildung 4.10 aufgegriffen. Der Dienst *getSite - accCtrl* ist wie dort angegeben modelliert. Abbildung 4.13 zeigt den Sinn einer zeitlich gesteuerten Scoreregression. Im ersten Zeitstrang trifft ein Resource Event für die Ressource *virt2*. Da dieses wohl einem automatischen Überwachungstool entstammt, wurde ihm nur die Score $0,4$ zugewiesen. Im zweiten Zeitstrang trifft ein weiteres Event ein, welches hier angenommen den Score der Ressource *virt1* auf $1,0$ erhöht. In der untersten Zeitschiene ist das Event für virt2 bereits aus dem Correlation Window wieder herausgefallen. Das System hat den Score der Komponente, für deren Fehlfunktion keine weiteren Meldungen eingegangen sind, automatisch

wieder regressiert. Zusätzlich ist ein Service Event für den Dienst *getSite - accCtrl* eingetroffen. Eine Korrelation dieser Events würde *virt1* korrekterweise als möglichen Fehlerkandidaten angeben. Die Regression der Score hat hier den wesentlichen Vorteil, dass keine überflüssigen Komponenten in den Fehlersuchprozess eingebunden werden.

4.4.1.4 Ressourcenredundanzen

Oft versuchen Provider besonders kritische Dienste durch redundante Komponenten zu realisieren. Eine gängige Redundanz auf Ressourcenebene ist zum Beispiel das Verwenden von Festplattenarrays in RAID Systemen. Fällt eine Festplatte aus, kann diese bei aktuellen Systemen einfach im laufenden Betrieb ausgetauscht werden, ohne dass der Datenbestand gefährdet ist. Auch auf höheren Ebenen sind Redundanzen üblich. Einen Blick in die Abbildung 2.2 aus dem Szenario in 2.1 zeigt, wie zum Beispiel die Webserver *nx112* usw. redundant ausgelegt sind. Solche Redundanzen erfordern eine besondere Behandlung beim Fehlermanagement. Im Gegensatz zur Annahme aus 4.1.2, ist es in der Praxis nicht immer gewährleistet, dass ein Dienst, der von mehreren redundanten Ressourcen abhängt, mit nur einer dieser Komponenten voll funktionsfähig ist. Die Dienstfunktionalität ist oft schon beim Ausfall einer Ressource stark beeinträchtigt, zum Beispiel wenn nach Ausfall einer redundanten Ressource eine frequentierte Nutzung zu einer Überlastung der übrigen, noch funktionierenden Ressourcen führt.

Ein Fehlermanagementsystem muss in der Lage sein, solche „Sonderfälle“ handzuhaben. Diese Anforderung kann auch über die Score erfüllt werden. Drei unterschiedliche Fälle müssen hier unterschieden werden:

1. Es liegen nur *Resource Events* vor, die Probleme bei bestimmten redundanten Ressourcen beschreiben. Im ersten Fall ist keine Korrelation mit anderen Events nötig und möglich. Erreicht die Score einer redundanten Ressource den maximalen Wert, ist klar, dass ein Fehler vorliegt und die Fehlerbehebung kann beginnen. Der übergeordnete Dienst sollte an seiner Score erkennen lassen, dass in seiner Ressourcenstruktur ein Problem aufgetreten ist. Dies wird mit einer Methode *getScore()* erreicht, die weiter unten beschrieben wird.

2. Eine oder mehrere redundante Ressourcen einer Funktionalität sind ausgefallen und es liegt ein Ereignis für den Dienst vor und Ressourcen Events für die betroffenen Ressourcen.
   Liegen sowohl Service wie auch Ressourcen Events vor, können diese korreliert werden. Mit einem *Service Event* erreicht die Score der betroffenen Funktionalität noch schneller das Maximum und zeigt somit ein Problem an.

3. Eine oder mehrere redundante Ressourcen einer Funktionalität sind ausgefallen und es liegt ein *Service Event* für den Dienst vor, jedoch (noch) keine Events für die betroffenen Ressourcen.

Der schwierigste Fall ist der dritte. Ohne Ereignisse auf der Ressourcen Ebene ist es schwer die verantwortliche redundante Komponente zu bestimmen, die zu einem Problem auf Dienstebene führt. Der Provider hat hier die Möglichkeit durch geschickte Wahl der Grundscore anfälligere Ressourcen hervorzuheben.
Ein effektiverer Weg der Ursachenbestimmung kann gegangen werden, wenn es für den Nutzer des Dienstes in irgendeiner Form ersichtlich ist, welche redundante Ressource er gerade nutzt. Dies setzt natürlich voraus, dass der entsprechende Dienst dies unterstützt. Mit dieser Information über eine bestimmte Dienstinstanz in einem *Service Event* kann dann gezielt die Score der betroffenen Ressource angehoben werden.

Ein gutes Beispiel für Punkt 3 ist, wenn der Nutzer eines Webhostingdienstes eine bestimmte WWW Seite eines Internetangebotes aufrufen möchte. Nach der Eingabe der Startadresse, z.B. `http://www.dell.de` ist für den Nutzer eine Auswahlseite in Abhängigkeit des Ursprungs seiner IP Adresse zu sehen. Für einen Besucher aus Deutschland öffnet sich so eine Seite mit einer URL, die mit `http://www1.euro.dell.com` beginnt. Die Information welche URL der Nutzer angezeigt bekommt kann in ein Ereignis integriert werden und dem Fehlermanagement weitergeleitet werden. Der Betreiber der Seite kann so bei einer redundant angelegten Infrastruktur erkennen, auf welchen Rechner der Nutzer weitergeleitet wurde.

4.4.2 Erweiterung des Klassendiagramms

Dieser Abschnitt beschreibt die Änderungen, die am Klassendiagramm von *Components* und Events vorgenommen werden müssen, um den oben beschrieben Veränderungen gerecht zu werden.

4.4.2.1 Components

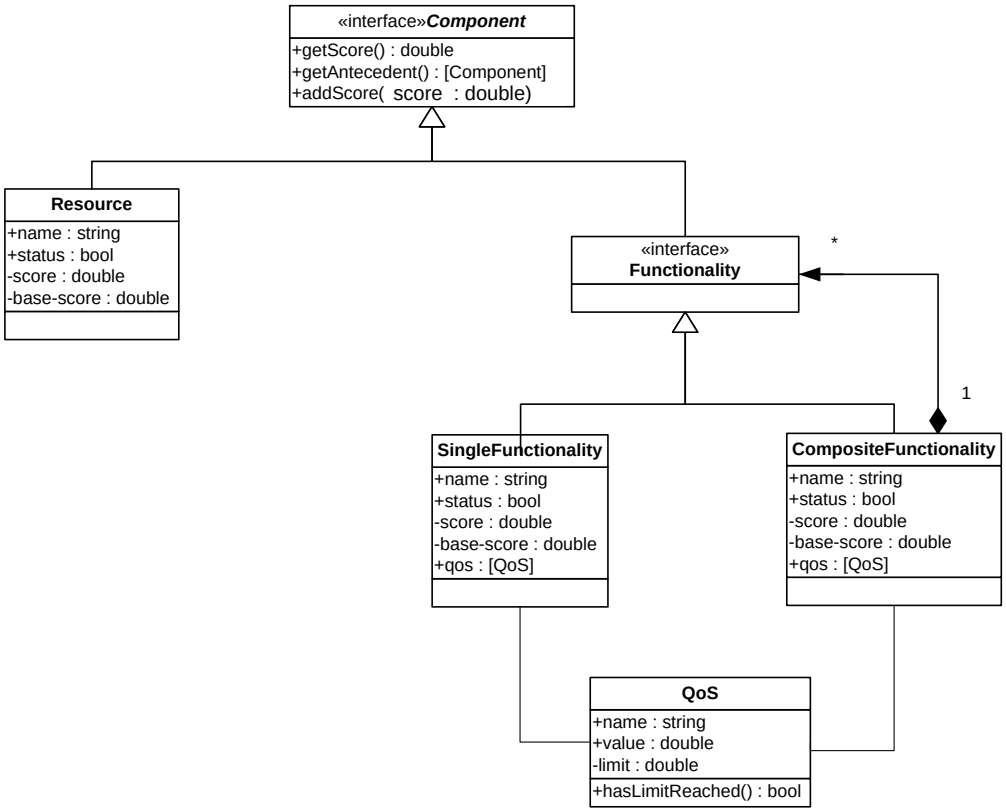


Abbildung 4.14: Erweiterung des Components Klassendiagramms

Abbildung 4.14 zeigt das veränderte UML Diagramm, welches die *Components* beschreibt.

Zunächst die Klasse durch das neue Attribut *score* erweitert. In diesem wird die Score einer Komponente akkumuliert. Der Zugriff auf diesen Wert wird ausschließlich über die neue Methode *getScore()* bereitgestellt. Der Grund dafür liegt darin, dass die Score sich nicht alleine über den *score* Wert definiert. Vielmehr muss auch eine Grundscore, im Diagramm mit *base-score* modelliert, in die Berechnung einbezogen werden. Wie oben bereits erwähnt, kann sich die Score auch individuell je nach Komponente berechnen, wenn QoS Parameter angegeben sind. Die Methode *getScore()* wird also für jede Komponente eigens implementiert. Natürlich kann auch eine Standardimplementierung angegeben werden, die eine statische Berechnung der Score vornimmt. Bei zusammengesetzten Funktionalitäten returniert die Methode die addierten Scores der untergeordneten Funktionalitäten.

Die Attribute *score* und *base-score* werden in der *Resource*, *SingleFunctionality* und in der *CompositeFunctionality* benötigt.

Mit der Methode *addScore(score : double)* besteht die Möglichkeit die Score zu erhöhen - zum Beispiel durch ein Event. Der Parameter *score* ist dabei der Wert, um den die Score der Komponente sich zu erhöhen hat.

4.4.2.2 Events

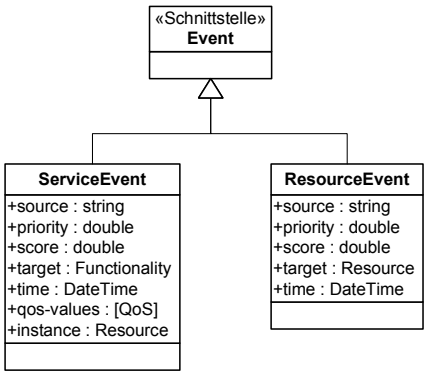


Abbildung 4.15: Erweiterung des Events Klassendiagramms

Abbildung 4.15 zeigt die Veränderungen im Klassendiagramm von *Service-* und *Resource Events*. Neu hinzugekommen ist das Attribut *priority*, womit eine Priorisierung von Events vorgenommen werden kann. Auch hier steht der Wert 1 für maximale Priorität. In *score* steht die Score, um die der Scorewert der betroffenen, in *target* spezifizierten Komponente, erhöht werden muss.

Das Feld *qos-values* vom Typ *QoS* speichert eventuelle, zum Event gehörige Informationen über QoS Parameter. In *instance* kann bei Funktionalitäten mit redundanter Ressourcenstruktur die betroffene Ressource angegeben werden, wenn diese Information bekannt ist.

4.4.3 Zusammenfassung und Bewertung

Mit der Einführung einer Score bei den beteiligten Komponenten, werden mehrere Anforderungen in Ergänzung zu Tabelle 4.1 erfüllt:

Anforderung	**erfüllt**	**Erläuterung**
Priorisierung von Ereignissen	erfüllt	Der Wert für Priorität bei Events erlaubt eine Priorisierung entweder einzelner Events oder Gruppen von Events nach Quelle sortiert.
Dienstinstanzen	erfüllt	Werden Benutzer von Funktionalitäten mit redundanter Ressourcenstruktur auf fehlerhafte Ressourcen geleitet, so besteht jetzt die Möglichkeit die Information über die fehlerhafte Ressource - sofern sie vorliegt - mit in ein *Service Event* zu integrieren.
Ressourcenredundanzen	erfüllt	Mit einer Erweiterung von *Service Events* können unterschiedliche Dienstinstanzen erfasst werden.

Tabelle 4.6: Bewertung nach Anforderungen nach Einführung der Score

Die Erweiterung hebt zudem einige in Abschnitt 4.1.2 getroffenen Annahmen auf. So kann auf die gemach-

ten Annahmen über Events vollständig verzichtet werden. Neben einer Priorisierung ist auch die Möglichkeit modelliert, unterschiedliche Glaubwürdigkeiten festzulegen, um unnötigen Korrelationsaufwand durch ungesicherte Ereignisquellen zu vermeiden.

Auch an den Annahmen über den Dienststatus muss nicht mehr festgehalten werden. Der Dienststatus bezieht sich nicht mehr länger auf eine boolesche Variable, vielmehr können mit Hilfe der Score sehr feine Abstufungen der Status vorgenommen werden.

Der Status zusammengesetzter Funktionalitäten kann ebenso berechnet werden, wie der einzelner Funktionalitäten.

4.5 Korrelationsalgorithmus

In diesem Abschnitt soll die in Kapitel 4 bisher vorgestellte Vorgehensweise zusammengefasst werden. Dazu ist es auf Grund der Änderungen an der Modellierung der Komponenten und Events notwendig, den Korrelationsalgorithmus anzupassen.

Kann für ein Symptom oder einen Fehler auf Dienstebene nicht unmittelbar eine Ursache gefunden werden, ist es notwendig gezielt Komponenten auf ihre Funktion zu überprüfen. Die Komponenten, die hier in Frage kommen lassen sich zum einen auf Grund der Abhängigkeitenmodellierung bestimmen und zum anderen mit Hilfe der Scoreinformation eingrenzen.

4.5.1 Anpassung des Korrelationsalgorithmus

Zur Übersicht werden zunächst alle Schritte, die nur Generierung einer Kandidatenliste und zur Anforderung von gezieltem Active Probing notwendig sind, in einer graphischen Übersicht als UML Aktivität dargestellt. Abbildung 4.16 zeigt diese.

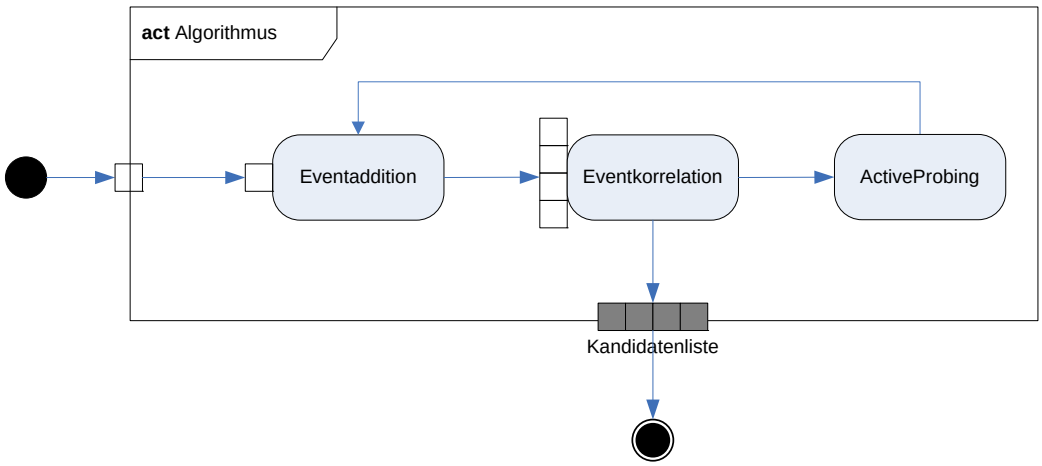


Abbildung 4.16: Übersicht über den Korrelationsalgorithmus als UML Aktivität

Nach dem Hinzufügen von entweder *Resource* oder *Service Events* durch das entsprechende Ressourcen- oder Servicemanagement, welche im Diagramm als Akteure auftreten, wird zunächst eine Eventfilterung angewendet. Anschließend wird bei der Ereigniskorrelation versucht Events zu verknüpfen. Auch hierbei werden Filterbedinungen angewendet. Der Korrelation folgt die Erstellung einer Kandidatenliste. Die Active Probing

Komponente fordert beim Service- oder Ressourcenmanagement wiederum Events an, die dann den Algorithmus durchlaufen.
Die einzelnen Teilschritte dieser Übersicht werden im Folgenden erklärt.

Eventaddition

Das Hinzufügen von Events wird in Abbildung 4.17 als UML Aktivitätsdiagramm illustriert.

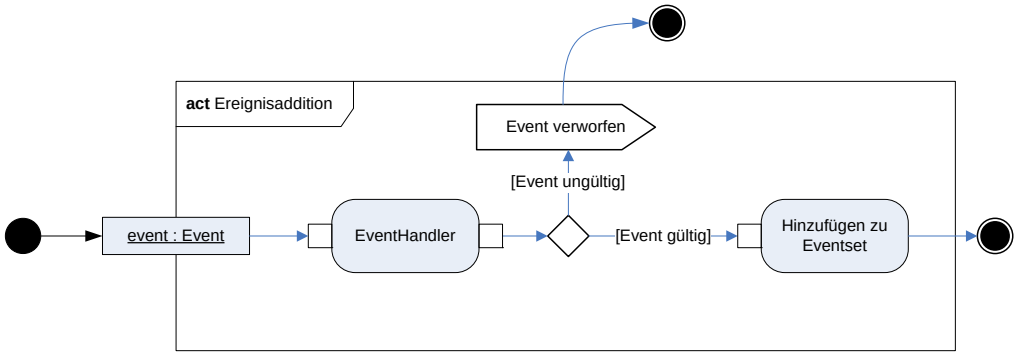


Abbildung 4.17: Hinzufügen von Events als UML Aktivitätsdiagramm dargestellt

Im Folgenden wird davon ausgegangen, dass Service Events von einer Schnittstelle zum Kunden und Resource Events von einem Ressourcenüberwachungssystem bereits vorliegen. Diese Events dienen als Eingabe (siehe Abbildung 4.16). Die Aktivität *Ereignisaddition* verarbeitet diese Events und fügt sie in ein *EventSet* ein. Dieses *Eventset* könnte zum Beispiel mit Hilfe einer Datenbank realisiert sein. Es speichert die jeweils aktiven Eventinformationen. Die Prozedur *Eventhandler* dieser Aktivität wird im Pseudocode 2 beschrieben.
Als Parameter erwartet die Prozedur das einzufügende Event. Zuvor prüft sie mit Hilfe der im Kapitel weiter oben beschriebenen Attribute das Event auf Gültigkeit und nimmt eventuelle Filterungen vor. Der Aufruf der Aktivität *Ereignisaddition* erfolgt immer dann, wenn ein neues Event generiert wird und in das System eintrifft. Hier kann ohne Einschränkung angenommen werden, dass beispielsweise mit Hilfe eines Pufferspeichers sichergestellt wird, dass Events seriell übergeben werden.

Ereigniskorrelation

Die Aktivität *Ereigniskorrelation* ist der komplexeste Teil des Ablaufes und in Abbildung 4.18 als Aktivitätsdiagramm dargestellt.

Hier findet die eigentliche Korrelation von Events statt. Die Aktion *Zugriff auf Eventset* ermöglicht den Zugriff auf das Eventset und wertet die Gültigkeit der gespeicherten Ereignisse aus. Diese Prüfung kann auch durch eine Zeitbedingung angefordert werden. Die zeitlichen Einschränkungen eines Correlation Windows können so realisiert werden. Die Aktion *Service-Correlator*, die in prozeduraler Schreibweise in 3 beschrieben wird, führt eine Korrelation von Events ausschließlich auf Dienstebene durch. Parallel dazu verfährt die Aktion *Resource-Correlator*, die das Gleiche auf Ressourcenebene durchführt und in Algorithmus 4 niedergeschrieben ist. Ein expliziter Aufruf dieser Funktionen ist nicht notwendig, da sie fortwährend in einer Endlosschleife laufen. Bei jedem Durchlauf werden die noch nicht korrelierten Events mit Hilfe der Aktion *Zugriff auf Eventset* geholt und verarbeitet.
Beide Prozeduren arbeiten mit den Events, die zuvor in das EventSet eingefügt wurden. Durch die regelmässige Überprüfung des Inhalts der Datenbank wird sichergestellt, dass keine überflüssigen Events im EventSet verbleiben, die das Korrelationsergebnis möglicherweise verfälschen können.

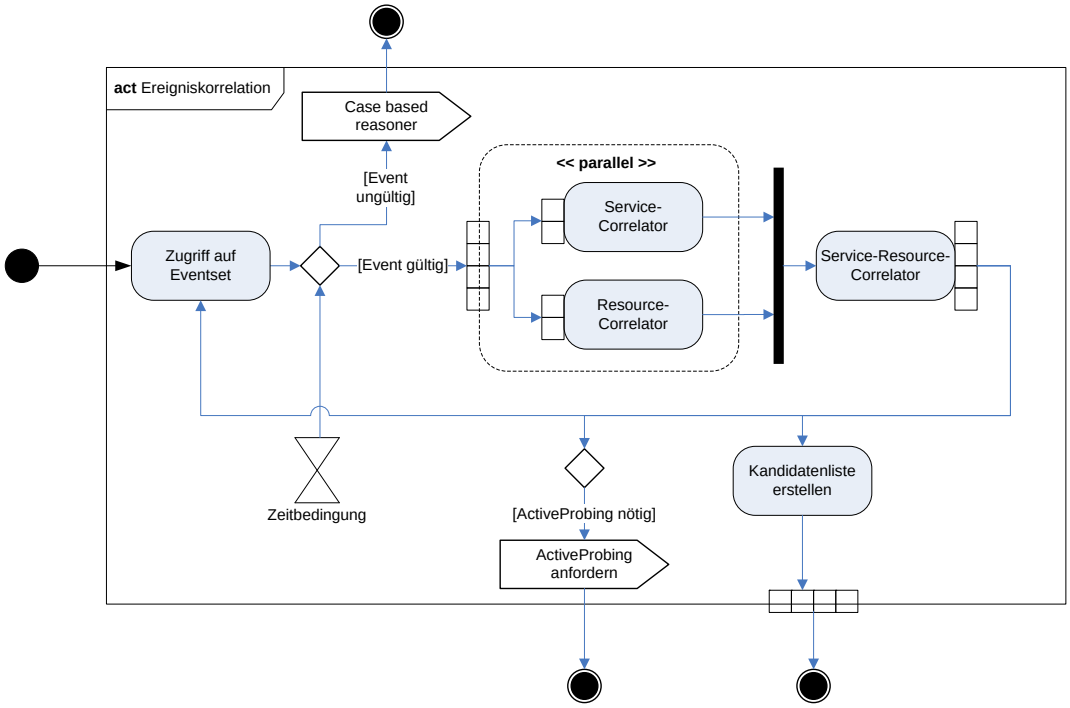


Abbildung 4.18: Die Ereigniskorrelation als UML Aktivitätsdiagramm dargestellt

Die Hauptaufgabe der *Ereigniskorrelation* Aktivität ist die Korrelation von abstrakten Service Events mit Resource Events. Diesen Schritt übernimmt die Prozedur *Service-Resource-Correlator*. Diese Aktion arbeitet mit den korrelierten Service und Resource Events aus den beiden zuvor erläuterten Aktionen. Sie korreliert diese Events soweit wie möglich, und kann so als ein wichtiges Korrelationsergebnis eine Liste mit fehlerhaften Diensten erstellen, derein Einträge auf mögliche Ursachen verweisen. Dieser Vorgang wird im Diagramm als *Kandidatenliste erstellen* referenziert. Diese Aktion erzeugt ein Feld als Ausgabe.

Active Probing

Können durch die oben angegebenen Prozeduren keine Verknüpfungen zwischen Events gefunden werden, verbleiben diese unkorreliert im *Eventset* vorhanden. Besonderes Interesse durch die Dienstorientierung begründet fällt dabei den *Service Events* zu, die nicht zu *Resource Events* korreliert werden konnten. Um auch für diese Fälle eine Fehlerursache bestimmen zu können, ist es erforderlich, aktive Tests auf ausgewählten Komponenten durchzuführen. Dieser Vorgang wird mit dem Signal *ActiveProbing anfordern* angezeigt.

Active Probing für Service Events

Zur Ursachenbestimmung bei *Service Events* ist es zunächst notwendig, einen Schritt tiefer in der Vererbungshierarchie nach fehlerhaften Komponenten zu suchen. In dieser Ebene der Hierarchie kann noch kein Event vorliegen, wären diese ja sonst in den Schritten zuvor bereits korreliert worden. Eine Möglichkeit zum weiteren Vorgehen wäre, jetzt einfach alle diese Komponenten einem aktiven Test zu unterziehen, was jedoch einen nicht unerheblichen Aufwand und Belastung für die Infrastruktur darstellen kann. Während eine simple Überprüfung der Erreichbarkeit schon mit einer PING-Anfrage erledigt werden kann, erfordern machne Komponenten aufwendigere Testroutinen.
Ein differenzierterer Weg ist, zunächst eine Sortierung der betroffenen Komponenten nach ihrer *base-score* durchzuführen. So wird erreicht, dass immer nur die Komponenten getestet werden, die als wahrscheinlichste Fehlerursache in Frage kommen. Liegt die fehlerhafte Komponente (oder die Komponenten) jedoch erst am Ende dieser sortierten Liste, ist der Korrelationsprozess mit einem erweiterten Zeitaufwand verbunden.

Active Probing für Resource Events

Ein verändertes Vorgehen erfordert der Fall, wenn Resource Events vorliegen, die noch nicht korreliert werden konnten. Die Aufgabe besteht darin, den nächsten, darüber liegenden Dienst / Funktionalität auf Funktion zu prüfen. Liegt nach der Abhängigkeitenmodellierung mehr als ein Dienst darüber, kann wie beim Active Probing für Service Events vorgegangen und eine sortierte Liste der in Frage kommenden Dienste erstellt werden. Dieser Vorgang wird jedoch in dieser Arbeit nicht weiter verfolgt, da dies eher einer Impactanalyse entspräche (siehe dazu auch Abschnitt 3.4).

Das aktive Überprüfen einer Komponente wird hier nicht weiter erläutert. Es kann jedoch angenommen werden, dass dieser Vorgang die Erstellung von Events provoziert, die dann wiederum in den Korrelationsprozess, wie in *Eventaddition* beschrieben, eingespeist werden. Dieser Vorgang erlaubt die weitergehende Korrelation.

4.5.2 Prozeduren

In diesem Abschnitt sollen die einzelnen Prozeduren, wie sie in der Ablaufbeschreibung in den oberen Abschnitten schon kurz beschrieben wurden, präzisiert werden und mit Hilfe einer Pseudocodevariante dargestellt werden.

EventHandler

Die komplexen Möglichkeiten zur Filterung von Events, die in den vorhergehenden Abschnitten beschrieben wurden, erfordern eine Routine, die das Hinzufügen und die Filterungen von Events beschreibt. Die Prozedur *EventHandler* wie in Prozedur 2 in Pseudocode skizziert, beschreibt diesen Vorgang. Als Parameter erwartet die Funktion ein Event, wie es von einer Schnittstelle zum Kunden, oder von Ressourcenüberwachungssystemen stammt.

Procedure 2 EventHandler

```
1:  procedure EVENTHANDLER(event)  ▷ Nutzung eines Correlations Windows (Alternativen möglich wie
    in 4.2 diskutiert)
2:      t_cw ← 1 GETCORRELATIONWINDOW(event)
3:      if event.time < (NOW( ) −t_cw) then
4:          RETURN NULL                                   ▷ Das Event wird nicht weiter korreliert.
5:      end if ▷ Hier weitere Filterungen möglich - zum Beispiel Filterung nach Quelle bei Wartungsarbeiten
6:      if INLIST(event.source, blockedSources) then
7:          RETURN NULL
8:      end if
9:      if NOT ISEMPTY(event.qos) then
10:         for all event.qos do
11:             {Setze die QoS Werte des event.target auf die QoS Werte im Event}
12:         end for
13:     end if
14:
15:     if NOT ISEMPTY(event.instance) then     ▷ Für ServiceEvents mit redundanter Ressourcenstruktur
16:         event.instance.ADDSCORE(event.score)          ▷ Score der redundanten Ressource erhöhen
17:     end if
18:     event.target.ADDSCORE(event.score)                ▷ Das Event wird seinem target zugeordnet
19:
20:     ADDTOEVENTSET(event)                    ▷ Das Event wird für den aktiven Events hinzugefügt
21: end procedure
```

In den Zeilen 2 - 4 wird eine mögliche Filterung mit Hilfe eines Correlation Windows dargestellt. In Zeile 6 wird das Event verworfen, jedoch können auch andere Möglichkeiten, wie in Abschnitt 4.2 beschrieben, verwendet werden. Weitere mögliche Filterungen können im Anschluss daran getroffen werden. Zeilen 6 - 8 beschreiben eine Filterung nach der Eventquelle. Dazu wird eine hier nicht näher spezifizierte Liste

$blockedSources$ verwendet, die jene Eventquellen beinhaltet, deren Events nicht weiter beachtet werden sollen. Ein Grund, eine Komponente in diese Liste aufzunehmen, könnten zum Beispiel Wartungsarbeiten an der Komponente sein. In so einem Fall macht es keinen Sinn Events für diese Komponente weiter zu verarbeiten. Die Prozedur zeigt auch den Umgang mit Diensten, deren Funktion sich auf mehrere redundante Ressourcen stützt. Die Zeilen 15 - 18 zeigen, wie Service Events, die sich auf einen solchen Dienst beziehen und einen Verweis auf eine fehlerhafte (redundante) Ressource beinhalten, behandelt werden. Die Score der betroffenen Ressource wird mit der Funktion *addScore* erhöht.

QoS Parameter, die das Event eventuell trägt, werden in den Zeilen 9 - 13 behandelt. Die Parameter werden dem Dienst hinzugefügt - die Erhöhung der Score erfolgt beim Dienst selbst, da dort die entsprechenden Grenzwerte für einzelne Parameter gespeichert sind. Die Methode *getScore()* bei einer Komponente berechnet die Score unter Berücksichtigung diese QoS Werte. In Zeile 11 wird die Score, die das Event trägt, der betroffenen Komponente (*target*) zugewiesen. Die letzte Zeile der Prozedur beschreibt, wie das Event zum aktuellen Eventset hinzugefügt wird.

Service-Correlator

Der *Service-Correlator*, in Pseudocode in Prozedur 3 nimmt den ersten Teil der Korrelation auf Dienstebene vor. Solange Service Events vorliegen, die noch nicht korreliert wurden, werden die in der Abhängigkeitenhierarchie „darunter“ liegenden Dienste oder Funktionalitäten geprüft, ob für diese ebenso aktuelle Events vorliegen. Die Methode **GetAntecedentServiceEvents()** bestimmt diese Events. Als Parameter erwartet sie diejenige Zielkomponente, für den das aktuelle Service Event vorliegt ($se.target$). Das Ergebnis ist eine Liste mit Service Events, die in $lower\_ServiceEvents$ zwischengespeichert werden.
Die Events in dieser Liste können jetzt verknüpft werden. Diese Korrelation wird mit einem Aufruf der Methode **CORRELATE** erledigt. Diese Verknüpfung könnte beispielsweise durch einen Verweis auf das andere Service Event realisiert werden. Als Parameter erwartet diese Methode die zu korrelierenden Events. Es kann davon ausgegangen werden, dass diese Methode die korrelierten Events mit einer Markierung versieht, um eine doppelte Korrelation zu vermeiden. Wichtigstes Ergebnis dieses Aufrufs ist jedoch, die Zusammenführung des Events se mit dem die „niedrigere“ Komponente betreffenden Event $lower\_serviceEvent$.

Die **while** Schleife garantiert eine ständige Wiederholung dieser Prozedur. Die Ergebnisse eines Schleifendurchlaufes sind die soweit wie möglich korrelierten Service Events, die sich nach dem Aufruf von **Service-Correlator** in einem Feld $correlated\_ServiceEvents$ befinden.

Procedure 3 Korrelation von Service Events

```
 1: procedure SERVICE-CORRELATOR( )
 2:     while true do
 3:         while {Korrelation von Service Events möglich} do
 4:             se ← ServiceEvent
 5:             lower_serviceEvents ← GETANTECEDENTSERVICEEVENTS(se.target)
 6:             for all lower_serviceEvents do
 7:                 CORRELATE(lower_serviceEvent, se)
 8:             end for
 9:         end while
10:     end while
11: end procedure
```

Resource-Correlator

Der *Resource-Correlator*, in Pseudocode in Prozedur 4 korreliert Events, die sich auf Ressourcen beziehen. Für jedes Ressourcenevent werden die in Events, die sich auf in der Abhängigkeitenhierarchie „darunter“ liegenden Ressourcen beziehen, gesucht. Die Methode **GetAntecedentResourceEvents()** in Zeile 5 bestimmt diese und speichert sie in einer Liste $upper\_resourceEvents$. Die Korrelation wird wiederum mit einem Aufruf der Methode **CORRELATE** erledigt.

Die **while** Schleife garantiert wiederum eine ständige Wiederholung dieser Prozedur. Die Ergebnisse eines Schleifendurchlaufes sind die soweit wie möglich korrelierten Resource Events, die sich nach dem Aufruf von **Service-Correlator** in einem Feld $correlated\_ResourceEvents$ befinden.

Procedure 4 Korrelation von Resource Events

```
1:  procedure RESOURCE-CORRELATOR( )
2:      while true do
3:          while {Korrelation von Resource Events möglich} do
4:              re ← ResourceEvent
5:              lower_resourceEvents ← GETANTECEDENTRESOURCEEVENTS(re.target)
6:              for all lower_resourceEvents do
7:                  CORRELATE(lower_resourceEvent, re)
8:              end for
9:          end while
10:     end while
11: end procedure
```

Service-Resource-Correlator

In diesem letzten Korrelationsschritt findet die Korrelation zwischen Resource und Service Events statt. Als Eingabe verwendet diese Prozedur nicht das EventSet, sondern die Ergebnisfelder der oben beschriebenen Prozeduren **Service-Correlator** und **Resource-Correlator**.

Die Korrelation zwischen den Resource und Service Events ist in Prozedur 5 beschrieben.

Procedure 5 Korrelation von Service und Resource Events

```
1:  procedure SERVICE-RESOURCE-CORRELATOR( )
2:      for all serviceEvents do
3:          se ← serviceEvent
4:          lower_resourceEvents ← GETANTECEDENTRESOURCEEVENTS(se.target)
5:          for all lower_resourceEvents do
6:              CORRELATE(upper_resourceEvent, re)
7:          end for
8:      end for
9:  end procedure
```

Wie in den anderen Korrelationsschritten zuvor werden für jedes (bereits korrelierte) Service Event die Ereignisse für die in der Abhängigkeitenhierarchie „niedrigeren“ Komponenten bestimmt. Diese sind jedoch auf Grund der vorhergegangenen Korrelation notwendigerweise Ressourcen und damit Resource Events. Zeile 4 beschreibt diesen Vorgang. Der Aufruf von **CORRELATE** bewirkt also eine Korrelation von Resource zu Service Events.

Dieser Korrelationsvorgang führt dazu, dass zu jedem Dienst, dessen Score in den vorhergehenden Korrelationsprozessen erhöht wurde, eine Liste von Ressourcen angehängt werden kann, soweit diese natürlich durch Events als Fehlerursachen korreliert werden konnten. Eine Sortierung innerhalb dieser Liste von Ressourcen kann durch die Verwendung der Score dieser Komponenten ermöglicht werden. Die Komponente mit dem höchsten Score Wert ist auch die wahrscheinlichste Fehlerursache.

Active Probing

Wie bereits weiter oben erwähnt, können längst nicht alle Resource Events zu Service Events korreliert werden. Ein Grund hierfür ist, dass in der Regel Meldungen über fehlerhafte Ressourcen weitaus schneller als Benutzermeldungen eintreffen. Für diese Arbeit interessanter ist jedoch der andere Fall: Zu eingegangenen Dienstereignissen fehlen entsprechende Vorkommnisse auf dem Ressourcenlevel.

Ziel der Active Probing Komponente ist es, eine gezielte Auswahl an Funktionalitäten zu treffen, die auf ihre Funktion überprüft werden sollen. Dazu wählt diese Prozedur, die hier nicht in Pseudocode beschrieben wurde, die Dienste oder Ressourcen aus, die vom betroffenen Dienst abhängen. Um eine Priorisierung und gezieltere Prüfung dieser Komponenten zu erreichen, kann wiederum die Score als Sortierkriterium verwendet werden.

4.5.3 Zusammenfassung

Ziel dieses Kapitels war es eine allgemeine Vorgehensweise für eine regelbasierte Ereigniskorrelation zu finden. Nach einem initialem Ansatz eine Korrlation durch einfache logische Formeln durchzuführen, wurden in mehreren schrittweisen Erweiterungen eingeführt. In diesen konnten entscheidene Attribute bei den zu überwachenden Diensten und Ressourcen anhand der gestellten Anforderungen entwickelt werden. Eine ausschlaggebende Rolle spielt das Score-Attribut. Dessen Anwendung erlaubt sehr differenzierte Abbildungen von Ereignissen auf den Komponentenstatus.
Im letzten Schritt wurden allgemeine Prozeduren entwickelt, die die Anwendung der Attribute in kurzen Pseudocodeausschnitten demonstrieren. Neben der Applikation von Filterbedingungen erläutern diese das Zusammenspiel verschiedener Korrelationskomponenten, die auch das gezielte aktive Überprüfen von Ressourcen oder Diensten beinhalten. Wie Tabelle 4.7 zeigt, konnten weitesgehend alle geforderten Punkte durch diese Vorgehensweise erfüllt werden.

Anforderung	**erfüllt**	**Erläuterung**
Priorisierung	erfüllt	Siehe Abschnitt 4.4
Zeitproblematik	erfüllt	Siehe Abschnitt 4.2
Ressourcen- und Dienststruktur	teilweise erfüllt	Siehe Abschnitt 4.1
Messen der Dienstgüteparameter	erfüllt	Siehe Abschnitt 4.3
Dienste von Drittanbietern	erfüllt	Siehe Abschnitt 4.1
Multiple Ressourcenfehler	erfüllt	Siehe Abschnitt 4.1
Dienstinstanzen	erfüllt	Siehe Abschnitt 4.4
Ressourcenredundanzen	erfüllt	Siehe Abschnitt 4.4
Kandidatenliste	erfüllt	Siehe Korrelationsalgorithmus

Tabelle 4.7: Erfüllte Anforderungen

Um eine allgemeine Anwendung zu ermöglichen wurde zum Beispiel vermieden einen konkreten Weg zur Abspeicherung von Ereignissen in einem Eventset anzugeben. Dementsprechend fehlen auch die Methodenrümpfe von Methoden wie **CORRELATE**, da diese stark von der verwendeten Speicherstruktur abhängen. Je nach gewähltem System - eine Datenbanklösung ist nur eine Möglichkeit - müssen natürlich entsprechend die Aktionen für die Auswahl von Events aus dem Eventset implementiert werden.

Im nächsten Kapitel werden die entwickelten Konzepte aus diesem allgemeinen Ansatz übernommen und auf ein kommerzielles Produkt angewendet.

5 Anwendung am Szenario LRZ

Inhaltsverzeichnis

5.1 Szenario und Ausgangslage 61
5.1.1 Szenario am LRZ 62
5.1.1.1 Identifizierung von Abhängigkeiten beim Webhosting Dienst 62
5.1.2 Tivoli Enterprise Console 63
5.1.2.1 Correlation rule set 64
5.1.2.2 E-Business rule set 65
5.1.2.3 Graphischer Regeleditor 68
5.2 Implementierung 69
5.2.1 Ereignisdefinitionen 69
5.2.1.1 Die Basisklasse EVENT 69
5.2.1.2 Eigene Klassendefinitionen 71
5.2.2 Regelerstellung: lrz_correlation Regelsatz 74
5.2.2.1 Übersicht 74
5.2.2.2 Hilfsregeln 74
5.2.2.3 Duplikate erkennen 76
5.2.2.4 Korrelation 76
5.2.2.5 Resource Handler 79
5.2.2.6 Timer Regel 80
5.3 Resultate 80
5.3.1 Versuch 1 80
5.3.2 Versuch 2 84
5.4 Zusammenfassung 87

In diesem Kapitel wird die im vorhergehenden Kapitel vorgestellte allgemeine Vorgehensweise zur regelbasierten, dienstorientierten Ereigniskorrelation am Szenario LRZ angewendet werden. Dazu folgt zunächst nochmal eine präzisere Beschreibung des bereits in Kapitel 2.1 dargestellten Szenarios am LRZ in Abschnitt 5.1.1.

Die entwickelten Konzepte der Vorgehensweise sollen auf der Tivoli Enterprise Console von IBM (siehe auch „Vorhandene Ansätze“ in Abschnitt 3.5.2) angewendet werden. Dazu werden zunächst die schon vorhandenen Möglichkeiten der TEC analysiert und hinsichtlich des Anforderungskataloges bewertet (Abschnitt 5.1.2).

In Abschnitt 5.2 wird eine Implementierung von Regeln an der TEC und die Definition von entsprechenden Ereignissen detailliert beschrieben. Mit Hilfe von konkreten Versuchen kann die Anwendbarkeit der allgemeinen Vorgehensweise auf reale Problemstellungen demonstriert werden.

5.1 Szenario und Ausgangslage

Bereits in Kapitel 2.1 wurde ein Überblick über den Webhosting und den E-Mail Dienst am LRZ gegeben. Für die Implementierung mit Hilfe der Tivoli Console soll nur ein Teil dieses Szenarios modelliert werden. Dazu ist es notwendig, die in diese Modellierung einbezogenen Dienste und Ressourcen zu identifizieren und deren Zusammenhänge zu erläutern.

Darauf folgt eine präzisere Beschreibung der Ausgangslage an der Tivoli Enterprise Console. Schon bei der Auslieferung vorhandene Regelsätze werden betrachtet und die Anwendbarkeit dieser für die dienstorientierte Ereigniskorrelation diskutiert.

5.1.1 Szenario am LRZ

Die Beschreibung des Webhosting und des E-Mail Dienstes am LRZ und die hohe Anzahl an potenziellen Nutzern dieser Funktionalitäten in Abschnitt 2.1 zeigen, dass hier ein automatisches Fehlermanagementsystem zur Überwachung der angebotenen Dienstleistungen wünschenswert ist. Neben diesen beiden Diensten könnte so ein System auch für weitere Funktionalitäten nützlich sein. Dazu zählen der grundsätzliche Konnektivitätsdienst, also beispielsweise die Verbindung zum LRZ via Modem oder ISDN. Weiterhin wird der drahtlose Netzzugang an den Münchner Universitäten sehr stark genutzt. Einher geht damit die Verwendung von VPN Verbindungen.

Andere LRZ Dienste hingegen erscheinen weniger geeignet für eine automatische Fehlersuche. Dazu gehört zum Beispiel das Angebot den Höchstleistungsrechner SGI Altix 4700 zu nutzen. Zum einen ist der Nutzerkreis dieser Hardware sehr eingeschränkt und zum anderen wird hier der Zugriff auf eine einzelne singuläre Ressource gewährleistet. Für die darauf auszuführende Software sind die Nutzer weitesgehend selbst verantwortlich. Andere - im Umfang kleinere - Dienstleistungen, wie die Nutzung spezieller Drucker, scheidet aus ähnlichen Gründen aus. Zu einem kleinen Nutzerkreis kommen noch zeitunkritische Anwendungen hinzu, die mit der Bereitstellung von Hilfeseiten im Internet oft ausreichend abgedeckt sind.

Zusammenfassend kann also davon ausgegangen werden, dass die Dienste Webhosting und E-Mail gut geeignete Kandidaten für die Einführung eines automatischen Fehlermanagementsystems sind. Für diese Arbeit wird im Folgenden der Webhostingdienst weiter betrachtet, um die Anwendbarkeit der TEC anhand dieser Dienste zu prüfen.

5.1.1.1 Identifizierung von Abhängigkeiten beim Webhosting Dienst

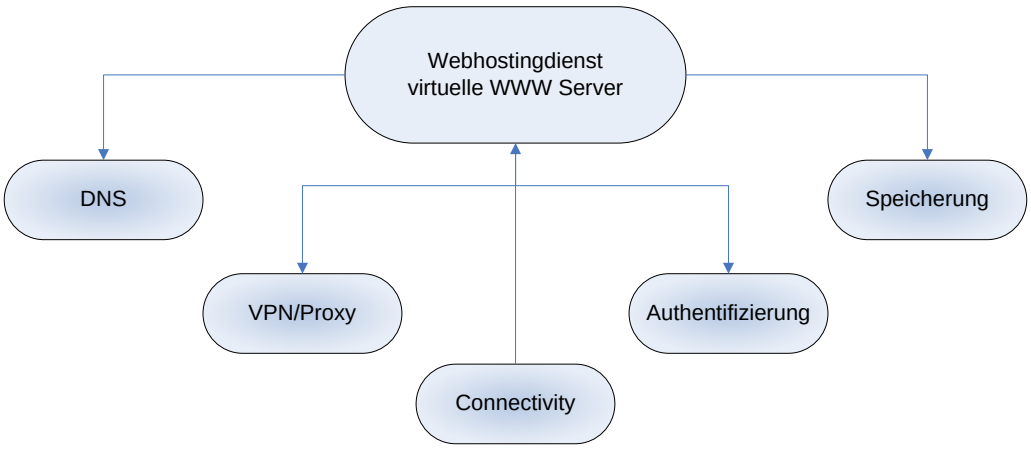


Abbildung 5.1: Abhängigkeiten des Webhosting Dienstes auf Dienstebene

Der Webhosting Dienst des LRZ basiert auf einigen Subdiensten und Ressourcen. Die Abbildungen 5.1 und 5.2 zeigen einen Überblick über die beteiligten Komponenten. In der ersten Grafik sind die Abhängigkeiten des Webhosting Dienstes zu weitern Subdiensten dargestellt, während die zweite Zeichnung direkte Abhängigkeiten zu Ressourcen zeigt. Obwohl die Darstellung keine indirekten Abhängigkeiten zwischen weiteren Res-

sourcen oder zwischen weiteren Subdiensten darstellt, lassen die welches hohe Maß an Komplexität die Realisierung dieser Funktionalitäten mit sich bringt.

Subdienste Der als *Speicherung* bezeichnete Dienst realisiert die Speicherung der dynamischen und statischen Webseiten. Der DNS Dienst dient zur Lokalisierung der angebotenen Webseiten und basiert dabei auf den grundsätzlichen *Connectivity* Dienst, der hier eine Abstraktion über die Netzwerkverbindungen, zu denen auch Komponenten wie Router oder Switches gehören, ist. Der VPN / Proxy Dienst kommt zum Einsatz, wenn Nutzer Webseiten über ein VPN aufrufen möchten. Möchte der Nutzer Änderungen an seinem Webangebot vornehmen, werden die Funktionalitäten des *Authentifizierungsdienstes* benötigt.

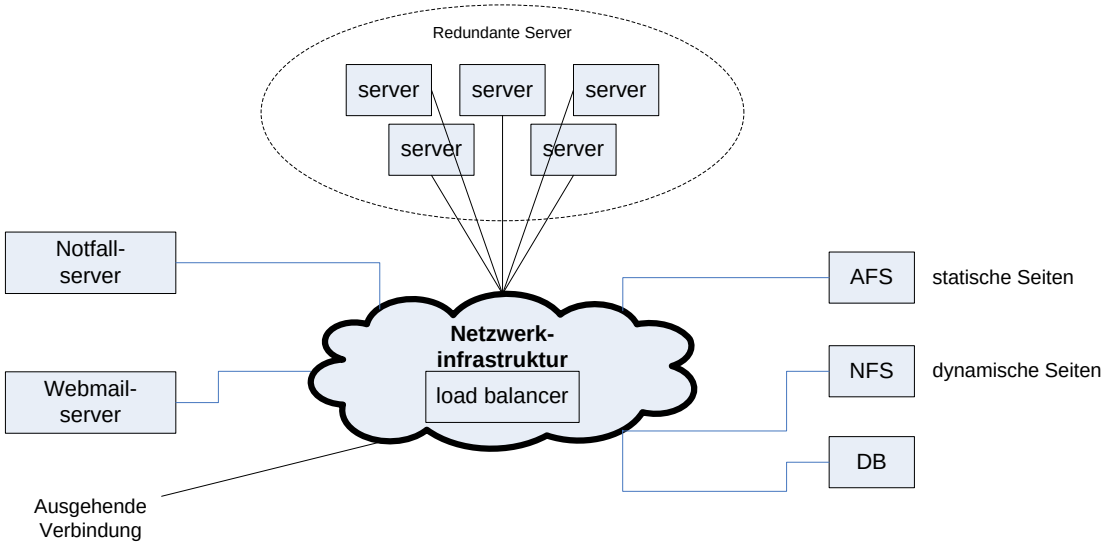


Abbildung 5.2: Abhängigkeiten des Webhosting Dienstes auf Ressourcenebene

Ressourcen Eine Lösung auf Ressourcenebene, die hohe Verfügbarkeit garantiert, wird durch einen Lastverteiler (*load balancer*) und zehn Paare von Webservern erreicht. Neben der benötigten Netzwerkinfrastruktur (in der Illustration als unbestimme Wolke dargestellt), die diese Konfiguration benötigt, zählen die Apache Webserver Anwendungen ebenso zu den Ressourcen.
Alle diese Ressourcen befinden sich in einem separaten Gebäude am LRZ. In diesem „Rechnerwürfel“ genannten Neubau hat niemand einen festen Arbeitsplatz. Das bedeutet, dass alle Ressourcen, die hier ihren Platz haben, ausschließlich per Fernwartung administriert werden. Dadurch entstehen allerdings auch noch größere Abhängigkeiten zu SSH oder Konnektivitätsdiensten.

5.1.2 Tivoli Enterprise Console

Schon in Abschnitt 3.5.2 wurden die grundlegenden Konzepte der IBM Tivoli Enterprise Console beschrieben. Letztendlich haben die relativ guten Erweiterungsmöglichkeiten der TEC dazu geführt, dass dieses Produkt als Plattform für die folgende Implementierung verwendet wurde. Zunächst erfolgt in diesem Teil der Arbeit eine genauere Beschreibung der Regelsätze der Standardregelbasis *default rule base*, die IBM mit der Auslieferung bereitstellt, und für die Zwecke der dienstorientierten Ereigniskorrelation geeignet erscheinen.

Bei der Betrachtung dieser Regelsätze müssen die Unterschiede, die zwischen der im Kapitel zuvor vorgestellten Vorgehensweise und den Konzepten der Enterprise Console bestehen, beachtet werden. Eine wesentliche Differenz ist, dass bei IBM das Eventkonzept sehr stark im Vordergrund steht. Alle Informationen werden mit

Hilfe von Ereignissen kommuniziert. So ist auch die Ereignisdatenbank am ehesten mit einer Wissensbasis vergleichbar. Nach den Überlegungen in Kapitel 4 hingegen, besteht die Wissensbasis aus den Abhängigkeitsinformationen und den Status (beziehungsweise der *score*) der einzelnen Ressourcen und Dienste.
Wie viele andere kommerzielle Produkte zur regelbasierten Ereignisverarbeitung geht auch die TEC von der Annahme aus, dass immer nur eine Ursache für einen Fehler vorliegt (*single root cause assumption*).
Der Algorithmus in dieser Arbeit konzentriert sich im Wesentlichen auf die Korrelation von Ereignissen auf Grund von Abhängigkeitsinformationen. IBM bietet in der Standardregelbasis zwei Regelsätze an, die am ehesten für diesen Zweck geeignet erscheinen.
Detailliert werden die Einsatz- und Konfigurationsmöglichkeiten dieser Regelsätze in der *Rule Set Reference* [IBM d] beschrieben. Eine Übersicht und Bewertung folgt in den nächsten Abschnitten.

5.1.2.1 Correlation rule set

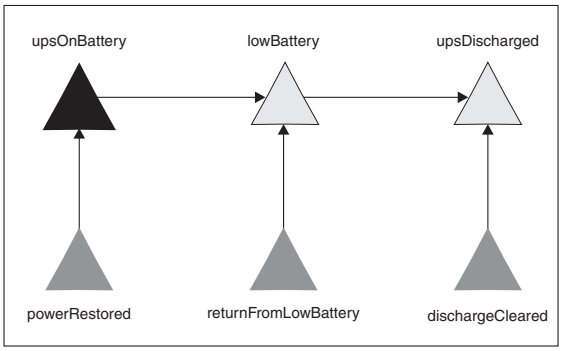


Abbildung 5.3: Beziehungen zwischen Events wie in Listing 5.1 definiert

Auch wenn der Name es vermuten lässt, erfüllt dieser Regelsatz nicht die Anforderungen zur Korrelation von Ereignissen, die in der Anforderungsanalyse gestellt wurden. Mit Hilfe von Ereignissequenzen (*event sequences*) sollen eintreffende Ereignisse korreliert werden. Diese Sequenzen werden vorher an einer Stelle im Regelsatz mit Hilfe des Schlüsselwortes *create_event_sequence* konfiguriert. Listing 5.1 zeigt so eine Definition. Leider ist der Begriff Sequenz hier sehr wörtlich genommen.

Als Beispiel wird hier immer wieder verwendet, dass ein Event, welches beschreibt, dass eine USV auf Batteriestrom umgeschaltet hat (*upsOnBattery*), die Ursache für ein weiteres Event darstellt, welches beschreibt, dass die Batterie einer USV zu Neige geht (*lowBattery*). Dieses wiederum ist die Ursache für ein Ereignis, welches die Entladung der Batterie als Information trägt (*upsDischarged*). Siehe dazu auch Abbildung 5.3. Treffen diese Ereignisse am Event-Server ein, so wird nach weiteren Ereignissen, die in so einer Sequenz definiert wurden, in der Ereignisdatenbank gesucht und diese korreliert.
Zusätzlich können noch sogenannte *clearing events* festgelegt werden, die andere Events in ihrer Wirkung aufheben. So ist möglicherweise ein *powerRestored* Event ein clearing event für *upsOnBattery*.

Listing 5.1: Anwendung von *create_event_sequence*

```
create_event_sequence(
  ['upsOnBattery','lowBattery','upsDischarged'],
    %attribute conditions for the cause chain of events
    [hostname,['status',not_equals,'CLOSED']],
      [
        %clearing relationships in the cause chain of events
        clears('powerRestored',[],['upsOnBattery'],[]),
        clears('returnFromLowBattery',[],['lowBattery'],[]),
        clears('dischargeCleared',[],['upsDischarged'],[])
      ]
),
```

Bewertung Die Definition von Sequenzen erlaubt keine Verzweigungen. Betrachtet man Abhängigkeiten mit Hilfe eines zyklenfreien Graphen (eines Baumes), so müssten für jeden Knoten des Baumes alle Pfade zu seiner *root cause*, wie in Listing 5.1, aufgeschrieben werden. Dies bedeutet einen erheblichen Mehraufwand für die Modellierung von Abhängigkeiten. Auch ist es beim Correlation Rule Set nicht möglich mehrere Root Causes für ein Problem festzuhalten.
Anwendung kann dieser Regelsatz dennoch für einfache, kausal zusammenhängende Folgen von Ereignissen finden, wie im Beispiel von IBM demonstriert. Die Idee von *Clearing Events* entspricht am ehesten dem Konzept der positiven Events, welches in Abschnitt 4.4 bei der Einführung einer Score für Komponenten vorgestellt wurde.

5.1.2.2 E-Business rule set

Das E-Business rule set analysiert Ereignisse, die von einem *IBM WebSphere Application Server* (WAS), von IBM DB2 Datenbankdiensten oder von *WebSphere MQ Software* (WebSphere MQ) stammen. Der Regelsatz untersucht dabei Events auf Abhängigkeiten, die vom Benutzer in einer externen Datei festgelegt werden können. Diese Abhängigkeiten werden wie in Listing 5.2 definiert und müssen vor der Benutzung am Event-Server geladen werden. Hier werden Abhängigkeiten zwischen mehreren Hosts (host[a-e].lrz.de) definiert. Dieser Regelsatz unterstützt nur Beziehungen zwischen WebSphere MQ Ressourcen, welche mit dem Schlüsselwort *WMQ_DEPENDS_ON_WMQ* gekennzeichnet werden, und zwischen WAS und DB2 Datenbanken auf verschiedenen (!) Hosts *WAS_DEPENDS_ON_DB2*.

Listing 5.2: Definition von Abhängigkeiten für E-Business rule set

```
hosta.lrz.de    WAS_DEPENDS_ON_DB2      hostb.lrz.de
hosta.lrz.de    WAS_DEPENDS_ON_DB2      hostc.lrz.de
hosta.lrz.de    WMQ_DEPENDS_ON_WMQ      hoste.lrz.de
hostc.lrz.de    WAS_DEPENDS_ON_DB2      hostd.lrz.de
```

Empfängt der Event-Server also ein Event für einen WAS auf der Ressource mit dem Hostnamen *hosta.lrz.de* und liegt bereits ein Event für einen Datenbankserver mit dem Hostnamen *hostb.lrz.de* vor, so kann eine Regel den in der Abhängigkeitendatei definierten Zusammenhang finden und eine kausale Verbindung zwischen den beiden Events herstellen. Die Ursache für den Ausfall des WAS scheint also gefunden zu sein.
Im E-Business rule set sind eine ganze Reihe an Regeln definiert, um diesen Ablauf zu ermöglichen. Neben den eigentlichen Korrelationsregeln werden noch zahlreiche „Hilfsregeln“ benötigt, um den reibungslosen Ablauf sicherzustellen. Diese sind zum Beispiel für so profane Aufgaben, wie die Umwandlung von Hostnamen in Kleinschreibung, um das korrekte Auffinden der Abhängigkeiten zu gewährleisten.

Wie die Korrelationsregel, die die Korrelation zwischen WAS und DB2 Events durchführt, aussieht, zeigt Listing 5.3 bis 5.8. Für das Verständnis der Regel nicht notwendige Abschnitte wurden mit einem *...* auskommentiert, um die Quelltextausschnitte in ihrer Länge zu beschränken. Dabei ist zu beachten, dass die Regeln vorkompilierte Prädikate der Prolog Sprache benutzen. Zusätzlich können auch bestimmte Prologelemente in den Regeln verwendet werden (beschrieben in Appendix A von [IBM c]).

Listing 5.3: E-Business Regel: associate_was_db2

```
1 rule: associate_was_db2:
2 (
3         event: _event
4         of_class _class within ['WebSphereAS_high_DBPool_faults',
5                                 'WebSphereAS_high_DBPool_avgWaitTime',
6                                 'WebSphereAS_high_Transaction_avg_response_time',
7                                 'WebSphereAS_high_Transaction_timeout_ratio',
8                                 'WebSphereAS_high_DBPool_percentUsed'
9                                ]
10         where [
11                 status: outside ['CLOSED'],
{*...*}
18         ],
```

Zeile 1 - 18 zeigt den Beginn der Regel. Diese Regel *associate_was_db2* wird auf Events den in Zeilen 4 - 8 aufgeschriebenen Klassen hin aktiviert. Als einzige Filterbedingung gilt die Bedingung in Zeile 11: Nur Events, die noch nicht vom Event-Server geschlossen wurden, aktivieren diese Regel. Abhängig von der Klasse des eintreffenden Events werden mögliche Ursachenevents festgelegt (Zeilen 30 - 34), nach denen dann unter den vorher eingetroffenen Ereignissen gesucht wird.

Listing 5.4: E-Business Regel: associate_was_db2

```
30  % Define cause events list for WebSphereAS_high_DBPool_faults event
31  member(_class, ['WebSphereAS_high_DBPool_faults']),
32  _cause_classes = ['DB2_Down_Status',
33                    'DB2_High_ConnectionErrors'
34                   ]
```

Das Prädikat *check_dep* ist dafür verantwortlich, dass alle *WAS_DEPENDS_ON_DB2* Abhängigkeiten, die zuvor definiert wurden, analysiert werden. Weiterhin werden die diesen Abhängigkeiten und dem Hostnamen des aktuellen Events entsprechende Ereignisse aus dem Eventcache ermittelt. Dieses Prädikat liegt nur in kompilierter Form vor, so dass auch auf Nachfrage bei einem IBM-Mitarbeiter keine Einsicht in den Quelltext möglich war.
Das Ergebnis dieser Suche, also ein Link auf ein entsprechendes Event, wird in der Variable *_cause_event* gespeichert. Für die eigentliche Korrelation, also die Verknüpfung der beiden Events, ist das Prädikat *link_effect_to_cause* in Zeile 89 zuständig. Dieses Prädikat erzeugt eine Referenz bei dem WAS Events (*effect event*), auf das in der Variable *cause event* gespeicherte DB2 Event, indem die Eventattribute *cause_event_handle* und *cause_event_date* mit Werten des Ursachenereignisses gefüllt werden.
Ein Event wird in der TEC mit diesen beiden Attributen (*handle* und *date*) eindeutig identifiziert. Dabei ist *date* ein UNIX Timestamp. Das Eventhandle wird für jedes Event, welches innerhalb einer Sekunde eintrifft, erhöht. Eine Ausnahme bilden von Regeln generierte Ereignisse. Diesen wird ein Eventhandle ¿ 1000 zugeteilt.

Listing 5.5: E-Business Regel: associate_was_db2

```
81 check_dep(_event, _cause_classes, 'WAS_DEPENDS_ON_DB2', 'IBM_DB2', _cause_event),
82    bo_get_class_of(_cause_event, _cause_class),
83    tl_fmt(ebusiness_tl, '\t...event %s identified\n', [_cause_class]),
84    (
85
86    % Check for DB2 events
87    member(_cause_class, _cause_classes),
88    %  Associate event with cause event
89    link_effect_to_cause(_event, _cause_event),
```

Neben Events für eine DB2-Datenbank werden zusätzlich noch Events gesucht, die sich auf den Netzwerkstatus der entsprechenden Hosts beziehen. Werden solche gefunden, generiert die Regel ein *TEC_Probable_Event_Association* Event, welches entsprechende Informationen für den Administrator enthält. Außerdem prüft die Regel, ob eventuelle Wartungsevents, die beschreiben, dass bestimmte Hosts gerade Wartungsarbeiten unterzogen werden, vorliegen und stellt dann eine Verknüpfung zwischen den Events, wie in Quelltextauszug 5.5 Zeile 89 beschrieben, her.

Die Regel setzt das Eintreffen der Events in der zeitlichen Reihenfolge, wie die Abhängigkeiten definiert sind, voraus. Träfe also das Ereignis für eine DB2-Ressource vor dem Ereignis für den WAS ein, könnte keine Verbindung zwischen diesen hergestellt werden. Dieses Problem löst das Regelset mit einer sogenannten „redo“ Regel (siehe Listing 5.6).

Listing 5.6: E-Business Regel: redo_db2_was

```
1 rule: redo_db2_was:
2 (
3         event: _event
4         of_class _class within ['DB2_Down_Status',
5                                 'DB2_High_ConnectionErrors',
6                                 'DB2_High_ConnWaitingForHost',
```

```
7                                   'DB2_High_MostRecentConnectResponse',
{*...*}
18                                   ]
19          where [ status: outside['CLOSED'],
20                  fqhostname: _fqhostname
21          ],
```

Diese wird für DB2-Events aktiviert und sucht, genau umgekehrt wie die vorherige Regel, nach WAS Events. Die Verknüpfung wird allerdings nicht durch diese redo-Regel hergestellt, stattdessen wird eine erneute Aktivierung der *associate_was_db2* Regel angefordert. Dies geschieht in Zeile 44 in Listing 5.7 mit dem Befehl *redo_analysis*. Zuvor wurde mit dem *all_instances* Prädikat, welches in einer Art Schleife den Eventcache nach angegeben Kriterien durchsucht, nach WAS Events gesucht.

Listing 5.7: E-Business Regel: redo_db2_was

```
29 % Search for dependent PAC effect event
30      all_instances(
31 event: _dep_event
32 of_class within [ 'WebSphereAS_high_DBPool_faults',
33                   'WebSphereAS_high_DBPool_percentUsed',
34                   'WebSphereAS_high_DBPool_avgWaitTime',
35                   'WebSphereAS_high_Transaction_avg_response_time',
36                   'WebSphereAS_high_Transaction_timeout_ratio'
37                 ]
38                 where [ status: outside['CLOSED'],
39                 fqhostname: equals _dep_hostname
40                 ],
41 _event - _latency - 0),
42 bo_get_class_of(_dep_event, _dep_class),
43 tl_fmt(ebusiness_tl,'\t\t...reanalysis requested for %s event\n', [_dep_class]),
44 redo_analysis(_dep_event)
```

Ähnliche Regeln existieren für Netzwerkfehlermeldungen oder für Events, die in einem *WMQ_DEPENDS_ON_WMQ* Zusammenhang stehen.

Bewertung und Ergebnisse Zunächst erschien dieser Regelsatz sehr geeignet dafür, ihn für die dienstorientierte Ereigniskorrelation einzusetzten. Gründe dafür sprechen die Möglichkeit, in einer externen Datei Abhängigkeiten festzulegen. Die Fähigkeit mit Hilfe der redo-Regeln Unabhängigkeit von der Eintreffreihenfolge der Events zu erreichen, trug ebenso zu dieser Annahme bei. Im Gegensatz zur *Correlation Rule Base* konnten hier eine Ressource mehrfache Abhängigkeiten definiert werden. Die Idee war zunächst, eigene Abhängigkeitstypen (zum Beispiel wie in [Marc 06] vorgeschlagen) festzulegen. Zusammen mit der Definition von eigenen Eventklassen, wäre es mit nur relativ geringen Änderungen an den Regeln möglich gewesen, diesen Regelsatz für zumindest einige der in Kapitel 4 entwickelten Konzepte zu verwenden.
Die Definition eigener Eventklassen konnte relativ einfach geschehen. Jedoch war es, wie weiter oben bereits erwähnt, nicht möglich die Implementierung des entscheidenden Prädikates *check_dep* zu ermitteln. So war auch völlig unklar, welche Parameter es erwartet und wie die abhängigen Hosts aus der Abhängigkeitendatei ermittelt werden. Zusätzliche Restriktionen ergaben sich, da es nicht möglich war, außer den von IBM implementierten beiden Abhängigkeitstypen *WMQ_DEPENDS_ON_WMQ* und *WAS_DEPENDS_ON_DB2*, eigene zu verwenden. Diese Einschränkungen können nicht hingenommen werden, wenn eine allgemeine Anwendbarkeit auf nahezu beliebige Szenarien gewährleistet werden soll. Mit der Anpassung der *cause* und *effect* Klassen in den Regeln wurde die Logausgabe aus Listing 5.8 erzielt.

Listing 5.8: Log Ausgabe nach Modifizierung des e-business rule sets

```
<<Entering associate_was_db2 action>>
   ...processing TEC_LRZ_SERVICE_DOWN event with fully qualified hostname: hosta.lrz.de
   ...ebusiness_hints = yes
   <<Entering check_dep_itm>>
      ...searching for dependency facts for hostname hosta.lrz.de
```

```
      ...identified dependent hostname hostb.lrz.de
   <<Exiting check_dep_itm>>
   ...event TEC_LRZ_NODE_DOWN identified
   ...TEC_LRZ_SERVICE_DOWN event associated with TEC_LRZ_NODE_DOWN event
<<Exiting associate_was_db2 action action>>
```

Diese Ausgabe zeigt, wie ein Event der Klasse *TEC_LRZ_SERVICE_DOWN* mit einem Event der Klasse *TEC_LRZ_NODE_DOWN* unter Verwendung einer externen Datei mit Abhängigkeiten korreliert wird. Dies entspricht einer Dienst-Ressource Abhängigkeit. Schon aus den Gründen, dass hier die von IBM vorgegebenen Abhängigkeitstypen verwendet werden müssen und keine Transparenz bei der Ermittlung der Abhängigkeiten existiert, wurden die Versuche mit diesem Regelsatz beendet.

5.1.2.3 Graphischer Regeleditor

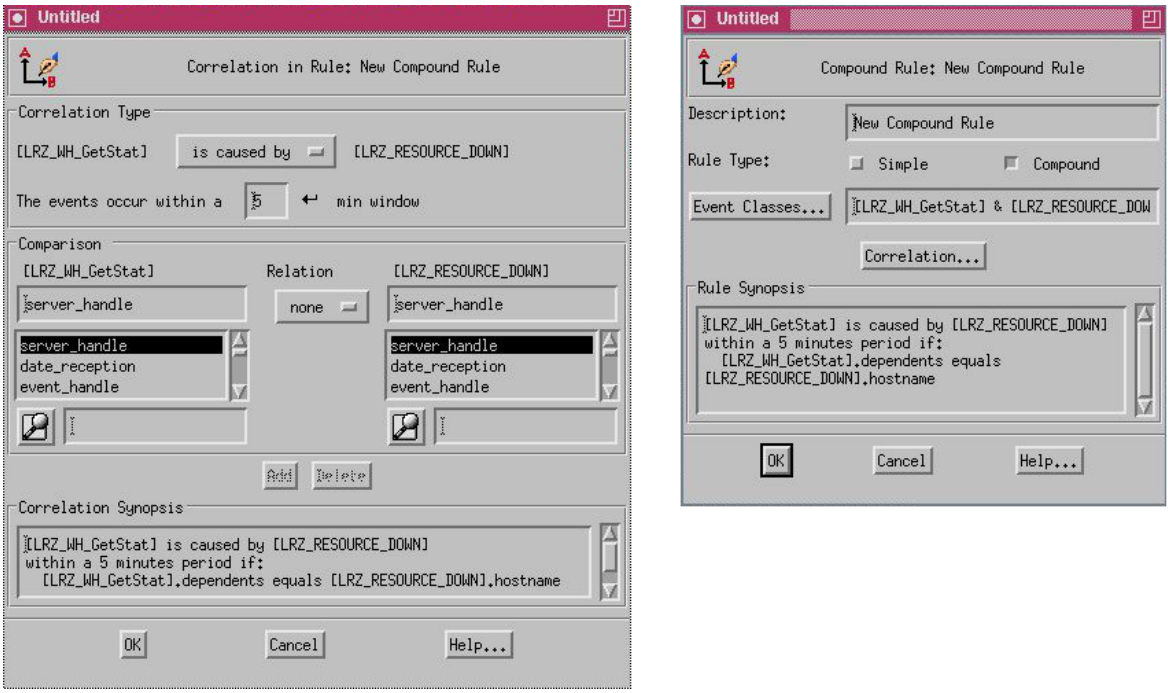


Abbildung 5.4: Graphischer Regeleditor der TEC

Die TEC bietet mit der Java-Oberfläche auch eine Möglichkeit, Regeln mit Hilfe eines graphischen Regeleditors zu erzeugen. Dieser bietet Anwendern ein Werkzeug, mit dem auch ohne Kenntnis der Regel- oder Prologsprache, eigene Regeln mit Hilfe von natürlichen Sprachelementen und anklickbaren Objekten zu gestalten. Die Abbildung 5.4 zeigt Bildschirmfotos von diesem Regeleditor. Ebenso erlaubt die graphische Oberfläche eine gut bedienbare Verwaltung von Regelbasen und den darin enthaltenen Regelsätzen.
Zum Arbeiten mit diesem Teil der Software bietet Kapitel 7 von [IBM c] gibt eine sehr genaue Schritt für Schritt Anleitung.

Bewertung der graphischen Regeleditors Die simple Bedienung des graphischen Regeleditors birgt auch gewisse Restriktionen: Neben sehr einfachen Regeln, wie beispielsweise das automatische Schließen von Events, die eine bestimmte Zeit im Eventcache verweilten, können auch Regeln definiert werden, die die Korrelation von Events zweier Klassen durchführen. Jedoch bleiben dem Anwender nur sehr eingeschränkte Möglichkeiten, um zum Beispiel Filtermöglichkeiten und damit Bedingungen für eine Korrelation festzulegen. Sehr schnell versagt das System, wenn versucht wird einen anderen Vergleichsoperator außer *equals* zu benutzen, obwohl die Regelsprache hier weitaus mehr Möglichkeiten bieten würde.

Ein Ansatz war, eine vom Regeleditor erstellte Regel als Basis zu nehmen, um anschließend eigene Regelelemente einzubauen. Die Art und Weise, wie automatisch generierte Regeln im Quelltext niedergeschrieben werden, verhinderte eine Weiterverwendung. Listing 5.9 zeigt einen Teil einer solchen Regel.

Listing 5.9: Durch den graphischen Regeleditor erstellte Regeln

```
/*****rule6:New Compound Rule*****/
rule: plain_compound_rule6_33554437: (event: _ev6 of_class within ['LRZ_WH_GetStat']
where [ dependents: _ev6_dependents ] ,
         redo_action: action1: (check_redo_ticket(_ev6, [redo1_compound_rule6_33554437]),
         first_instance(event: _ev7 of_class within ['LRZ_RESOURCE_DOWN'] where [
                hostname: _ev7_hostname equals _ev6_dependents ],
                _ev6 - 300 - 300),  re_causes_event( _ev7, _ev6))
, reception_action: action0: ((  first_instance(event: _ev7 of_class
within ['LRZ_RESOURCE_DOWN']
where [
                hostname: _ev7_hostname equals _ev6_dependents ],
                _ev6 - 300 - 300),
        (  re_causes_event( _ev7, _ev6))))
).
```

Leicht ist zu erkennen, dass die automatische Generierung dieser Regeln mit einer großen Unübersichtlichkeit einhergeht. Die fehlende Dokumentation von Prädikaten wie *check_redo_ticket* machten es äußerst schwer die gewünschte Funktionalität der Regel durch Einfügen eigener Prädikate und anderer Regelemente zu erreichen. Der Aufwand, der nötig gewesen wäre diese Modifikationen vorzunehmen, wäre wahrscheinlich genauso hoch, wenn nicht höher gewesen, wie der, die Regeln vollständig selbst aufzuschreiben.
Im folgenden Abschnitt 5.2 wird diese Möglichkeit genutzt, um neue, eigene Regeln zu erstellen.

5.2 Implementierung

Nachdem die Tivoli Console keine verwertbaren Möglichkeiten bietet, die entwickelten Konzepte umzusetzen, fiel die Entscheidung, Regeln komplett selbst zu erstellen. Dabei konnte auf einige Regelelemente aus den weiter oben vorgestellten Standardregelsätzen zurückgegriffen werden. Der Aufbau der eigens entwickelten Regeln wird in Abschnitt 5.2.2 abgefasst.
Vor der Entwicklung von eigenen Regeln ist es zunächst notwendig, die Ereignisse auf die zu modellierenden Dienste abzustimmen.

5.2.1 Ereignisdefinitionen

Wie bereits in 3.5.2 erwähnt, sind mit der TEC auch Vererbungshierarchien bei der Implementierung von Eventklassen möglich. Wie in der Vorgehensweise vom vorigen Kapitel beschrieben, ist es entscheidend welche Informationen ein Event trägt, um entsprechende Korrelationsregeln darauf anwenden zu können.

5.2.1.1 Die Basisklasse EVENT

Alle Events, die an der TEC zum Einsatz kommen, erben von einer Basisklasse *EVENT*. Diese Klasse, die, möchte man die an der Objektorientierung angelehnte Sprachweise beibehalten, im Übrigen als abstrakt aufgefasst werden kann, ist in der Datei `root.baroc` beschrieben. Listing 5.10 zeigt die implementierten Attributwerte, deren Typ (zum Beispiel *STRING* für Zeichen, oder *INTEGER* für Zahlenwerte) und Standardwerte (mit *default*= gekennzeichnet) dieser Klasse.

Listing 5.10: Definition der Basisklasse *EVENT*

```
TEC_CLASS : EVENT
  DEFINES {
```

```
    server_handle:          INTEGER,
    date_reception:         INT32,
    event_handle:           INTEGER,
    source:                 STRING;
    sub_source:             STRING;
    origin:                 STRING;
    sub_origin:             STRING;
    hostname:               STRING;
    fqhostname:             STRING;
    adapter_host:           STRING;
    date:                   STRING;
    status:                 STATUS,          default = OPEN;
    administrator:          STRING,
    acl:                    LIST_OF STRING,  default = [admin],
    credibility:            INTEGER,         default = 1,
    severity:               SEVERITY,        default = WARNING;
    msg:                    STRING;
    msg_catalog:            STRING;
    msg_index:              INTEGER;
    duration:               INTEGER,
    num_actions:            INTEGER,
    repeat_count:           INTEGER;
    cause_date_reception:   INT32,
    cause_event_handle:     INTEGER,
    server_path:            LIST_OF STRING;
    ttid:                   STRING;
  };
END
```

In keiner der zur Verfügung stehenden Dokumentationsunterlagen war eine detaillierte Beschreibung zu diesen Attributen zu finden. Jedoch erklären sich die meisten dieser durch ihre Namensgebung. Zum Verständnis der später folgenden Regel, werden einige Attribute, auch im Hinblick auf die in der allgemeinen Vorgehensweise festgehaltenen Eventattribute, kurz genauer beschrieben:

- `date_reception` speichert den UNIX Zeitstempel, wann das Ereignis beim Event-Server eingegangen ist. Zusammen mit `event_handle` kann ein Event über diese beiden Attribute identifiziert werden.
  Da dieser Wert, wie schon in 4.2 erklärt, nicht zwingend mit der Zeit, wann der Fehler eigentlich aufgetreten ist übereinstimmt, muss für diesen Zweck das Attribut `date` verwendet werden.
- `status` beschreibt den Bearbeitungsstatus des Events. Eine Enumeration mit den Werten *OPEN, CLOSE, RESPONSE* und *ACK* ist der Typ dieses Attributes. Verwendung findet es bei der Filterung von Ereignissen. Wurde ein Event von einer Regel schon einmal verarbeitet, kann der Status *ACK* zugewiesen werden. Wird das Event nicht mehr im Laufe eines Korrelationsprozesses gebraucht, kann es geschlossen werden.
- Auch `severity` ist eine Enumeration und kommt dem Konzept der Score sehr nahe. Die Werte dieses Schweregrades für ein Event können nur diskrete Werte wie *HARMLESS* oder *FATAL* einnehmen. Ein numerischer Wert wie in der entwickelten Vorgehensweise erlaubte eine differnziertere Abstufung. Dieses Attribut kann genutzt werden, um Schwellwerte festzulegen, oder Events nach erfolger Korrelation abzuschwächen oder aufzuwerten.
- Die Annahme der *single root cause* wird bei `cause_date_reception` und `cause_event_handle` deutlich. Ist ein Event eine Auswirkung auf ein anderes Ursachenereignis, werden mit diesen Attributen der Verweis auf dieses gespeichert. Mit diesem Konzept ist es nicht möglich, dass auf mehrere Ereignisse verwiesen wird. Damit wird die wichtige Anforderung, dass mehrere Komponenten zu einem Fehler beitragen können, nicht erfüllt.
- `credibility` wurde in den oben vorgestellten Regelsätzen von IBM nicht verwendet. Auch hier fehlt eine Dokumentation. Jedoch ist es möglich mit diesem Attribut die Priorität von Ereignissen zu

modellieren.

IBM rät die Klassendefinition von *EVENT* nicht zu verändern, da zahlreiche interne Events auch von dieser Klasse erben. Modifikationen könnten also die Funktionlität dieser beeinträchtigen. Da jedoch der eigenen Definition von Klassen, Attributen und Aufzählungen keine Grenzen gesetzt sind, stellt dies keinen Nachteil dar.

5.2.1.2 Eigene Klassendefinitionen

Klassen für Ressourcen und Dienste Bei der Bildung eigener Eventklassen muss zunächst eine Unterscheidung zwischen Ereignissen für Ressourcen und Ereignissen für Dienste getroffen werden. Allgemein sind Events für Ressourcen oft vom Hersteller fest vorgegeben, dagegen müssen Service Events an bestehende SLAs und angebotene Dienstfunktionalitäten angepasst werden.

Wie weiter unten noch beschrieben, wird beim Testen der eigenen Regeln nicht auf reale Events zurückgegriffen, sondern mit Hilfe eines TEC Befehls Ereignisse manuell in das System eingespeist. Aus diesem Grunde ist es zunächst notwendig, eine Basisklasse für Resource Events festzuhalten. Diese Klasse ist in Listing 5.11 dargestellt.

Listing 5.11: Definition der Basisklasse für Resource Events

```
TEC_CLASS:
 TEC_LRZ_RESOURCE_QOD_EVENT ISA EVENT
  DEFINES {
   source: STRING, default = "LRZ resource monitoring";
   severity: SEVERITY,     default = WARNING;
   status: STATUS,         default = OPEN;
   resource: STRING;
   linked_cause_handles: LIST_OF INTEGER,       default = [];
   linked_cause_dates: LIST_OF INT32,           default = [];
  };
END
```

Entscheidend in dieser Klasse ist das Attribut `resource`, welches die betroffene Komponente bezeichnet. Diese Klasse dient jedoch nur als abstrakte Basisklasse. Instanziert werden nur die Klassen, wie in Listing 5.12 aufgeführt. Zur einfacheren Überprüfung der Funktion in später folgenden Beispielen werden hier nur sehr einfache Events, die die Verfügbarkeit einer Ressource beschreiben, implementiert. Das *TEC_LRZ_RESOURCE_AVAIL_OK* Event ist ein positives Event, deswegen wurde diesem auch der Schweregrad *HARMLESS* per Standard zugewiesen.

Listing 5.12: Definition der Klassen für Resource Events

```
TEC_CLASS:
  TEC_LRZ_RESOURCE_AVAIL_NOK ISA TEC_LRZ_RESOURCE_QOD_EVENT;
END

TEC_CLASS:
  TEC_LRZ_RESOURCE_AVAIL_OK ISA TEC_LRZ_RESOURCE_QOD_EVENT
DEFINES {
   severity: SEVERITY,          default = HARMLESS;
  };
END
```

Für *Service Events* fungiert die Klasse *TEC_LRZ_SERVICE_QOS_EVENT* in Listing 5.13 als Basisklasse. Neu sind die Attribute `service`, welches den betreffenden Dienst bzw. die betroffene Dienstfunktionalität kennzeichnet und *service_access_point*, welches den Eingangspunkt für den Dienst speichert, den der Nutzer, der die Fehlermeldung absetzte, nutzt. In dem *valid_to* Attribut kann ein Zeitpunkt für die maximale Gültigkeit eines Events festgehalten werden. Dies entspricht dem Konzept der individuellen Gültigkeitszeiträume (Correlation Window in Abschnitt 4.2). Das Textfeld *description* bietet Platz für zusätzliche Informationen. Um

die Annahme der singulären Ursache aufzuheben, wurden die letzten beiden Attribute eingeführt. Hier können in einer Liste mehrere Verweise auf mögliche Ursachenevents gespeichert werden. Standardmäßig ist diese Liste, welche auch bei den Ressourceneventdefinitionen vorhanden ist, leer.
Zur Modellierung von Dienstfunktionalitäten dient ein weiteres Attribut *service_func*, welches den Namen der Funktionalität speichert. Mit der zusätzlichen Klasse *TEC_LRZ_SERVICEFUNC_QOS_EVENT* ist eine Unterscheidungsmöglichkeit zwischen Events für einen gesamten Dienst und Events für eine spezielle Dienstfunktionalität gegeben.

Listing 5.13: Definition der Basisklassen für Service Events

```
TEC_CLASS:
 TEC_LRZ_SERVICE_QOS_EVENT ISA EVENT
  DEFINES {
   source: STRING, default = "LRZ service monitoring";
   severity: SEVERITY,     default = WARNING;
   service: STRING;
   service_func: STRING;
   service_access_point: STRING;
   valid_to: INT32;
   description: STRING;
   linked_cause_handles: LIST_OF INTEGER,       default = [];
   linked_cause_dates: LIST_OF INT32,           default = [];
  };
END

TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_QOS_EVENT ISA TEC_LRZ_SERVICE_QOS_EVENT;
END
```

Instanziert werden wiederum nur die Klassen wie in Listing 5.14 aufgeführt. Dazu folgende Erläuterungen:

- `TEC_LRZ_SERVICE_AVAIL_NOK` und `TEC_LRZ_SERVICE_AVAIL_OK`
  beschreiben die Verfügbarkeit eines gesamten Dienstes.
- `TEC_LRZ_SERVICEFUNC_AVAIL_NOK` und `TEC_LRZ_SERVICEFUNC_AVAIL_OK`
  beschreiben die Verfügbarkeit einer Dienstfunktionalität
- `TEC_LRZ_SERVICEFUNC_TIME_NOK` und `TEC_LRZ_SERVICEFUNC_TIME_OK`
  Diese Eventklassen dienen als Beispiel, wie eine Zeitbedingung bei einem Event implementiert werden kann. Mit Hilfe des Attributes `time` könnte beispielshalber ein Service Event, welches eine Verletzung eines per SLA festgehaltenen Grenzwertes einer Zeitbedingung darstellt, modelliert werden.

Listing 5.14: Definition der Klassen für Service Events

```
TEC_CLASS:
  TEC_LRZ_SERVICE_AVAIL_NOK ISA TEC_LRZ_SERVICE_QOS_EVENT;
END

TEC_CLASS:
  TEC_LRZ_SERVICE_AVAIL_OK ISA TEC_LRZ_SERVICE_QOS_EVENT
    DEFINES {
      severity: SEVERITY,          default = HARMLESS;
    };
END

TEC_CLASS:
  TEC_LRZ_SERVICEFUNC_AVAIL_NOK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT;
END

TEC_CLASS:
  TEC_LRZ_SERVICEFUNC_AVAIL_OK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
    DEFINES {
```

```
      severity: SEVERITY,          default = HARMLESS;
    };
END

TEC_CLASS:
  TEC_LRZ_SERVICEFUNC_TIME_NOK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
    DEFINES {
      time: INT32;
    };
END

TEC_CLASS:
  TEC_LRZ_SERVICEFUNC_TIME_OK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
    DEFINES {
      time: INT32;
      severity: SEVERITY,          default = HARMLESS;
    };
END
```

Die Definition von Events, die sich auf einzelne Dienstfunktionalitäten beziehen, erfordern eine Erweiterung der Klassen für Ressourcenereignisse. Listing 5.15 zeigt zwei zusätzliche Klassen, welche als Ursachenevents für Events der oben festgelegten *TEC_LRZ_SERVICEFUNC_TIME_NOK* dienen. Sie beschreiben dabei als Beispiel die Auslastung der CPU eines Hosts.

Listing 5.15: Erweiterung der Klassen für Resource Events

```
TEC_CLASS:
  TEC_LRZ_RESOURCE_CPU_UTIL_NOK ISA TEC_LRZ_RESOURCE_QOD_EVENT
    DEFINES {
      utilization: REAL;
    };
END

TEC_CLASS:
  TEC_LRZ_RESOURCE_CPU_UTIL_OK ISA TEC_LRZ_RESOURCE_QOD_EVENT
    DEFINES {
      utilization: REAL;
      severity: SEVERITY,          default = HARMLESS;
    };
END
```

Die Gestaltung dieser Eventklassen ist stark davon beeinflusst, wie die TEC in ihren Regeln auf Ereignisse reagiert. So wurde eine eigene Klasse (*TEC_LRZ_SERVICEFUNC_TIME_NOK*) für einen QoS Parameter, wie es das Attribut `time` darstellt, erstellt. Dies geschah aus dem Grund, dass so die Relation zwischen Ursachen- und Effektklassen einfacher in der Regel modelliert werden konnte.
Mit der Festlegung dieser Eventklassen ist der Prozess der Ereignisdefinition längst nicht abgeschlossen. Zur Demonstration der im Folgenden vorgestellten Regel sind diese Klassen jedoch ausreichend.

Hilfsklassen Zusätzlich zu den oben vorgestellten Eventklassen werden noch einige Events benötigt, die zur Ablaufsteuerung und zur Informationsverarbeitung innerhalb des Regelsatzes benötigt werden.

- `TEC_LRZ_CLOSE_ALL ISA EVENT`
  Wie der Name schon sagt, werden auf Grund dieses Events alle anderen offenen Events geschlossen. Es dient nur zu Testzwecken.
- `TEC_LRZ_PROBABLE_CAUSE_EVENT`
  Mit der Generierung dieses Events wird ein Korrelationsvorgang angezeigt. In der Attributenliste werden Verweise auf ein *effect event* und ein *cause event* gespeichert.

- `TEC_LRZ_ACTIVE_PROBING_EVENT`
  Dieses Event wird vom Regelsatz erzeugt und hat nur temporären Bestand. In den Attributen `services` und `resources` werden jeweils Listen von Komponenten gespeichert, für die Active Probing (AP) angefordert wird. Eine Regel generiert jetzt für jeden Eintrag in dieser Liste ein entsprechendes Ereignis aus der Unterliste und verwirft das ursprüngliche.
  - `TEC_LRZ_ACTIVE_PROBING_RESOURCE`
  - `TEC_LRZ_ACTIVE_PROBING_SERVICE`

5.2.2 Regelerstellung: lrz_correlation Regelsatz

5.2.2.1 Übersicht

Mit der Erstellung eines neuen Regelsatzes, dem *lrz_correlation rule set*, wurde versucht folgende Ziele zu erreichen:

- Korrelation von Resource und Service Events nach frei definierbaren Abhängigkeiteninformationen
- Anfordern von gezieltem Active Probing für Dienste und Ressourcen
- Verarbeitung von doppelt eintreffenden Service Events
- Verarbeitung von nicht korrelierten Dienstereignissen (für die keine Ursachenevents gefunden werden konnten)
- Benutzung eigener Eventklassen

Um diese zuvor in der Vorgehensweise ausgearbeiteten Zielsetzungen demonstrieren zu können, sind eine Reihe von Annahmen nötig, die getroffen werden konnten, ohne die Demonstration der Funktionalität zu gefährden.

- Ein vorhandenes System zur Ressourcenverwaltung liefert die in 5.2.1.2 vorgestellten Ressourcenevents.
- Events werden nicht von realen Systemen generiert, sondern mit Hilfe eines Befehls, über welchen manuell Ereignisattribute gesetzt werden können.
- Werden Anforderungen für Active Probing versendet, so ist mit einem negativen Event zu rechnen, sollte die zu prüfende Komponente nicht funktionsfähig sein.
- Es kann angenommen werden, dass positive Events negative aufheben. Für diesen Korrelationsvorgang wurde keine explizite Regel erstellt.

Der neu erstellte Regelsatz wurde, wie in der TEC üblich, in mehrere Regeln und diese teilweise in mehrere Aktionen unterteilt. Letztere Unterteilung hat den Zweck, die Abarbeitung von Events innerhalb einer Regel zu steuern.
Eine Übersicht über den gesamten Regelsatz gibt Abbildung 5.5.
Den gesamten Quelltext dieses Regelsatzes, sowie alle benötigten Eventklassen finden sich aus Platzgründen in Anhang A. Im weiteren Text werden nur die für die Erklärung der jeweiligen Funktion benötigten Quelltextabschnitte angegeben.

5.2.2.2 Hilfsregeln

Die Regeln *startup*, *shutdown*, *probable_cause* und *active_probing* können als Hilfsregeln kategorisiert werden. Sie tragen nur indirekt zur eigentlichen Korrelation bei. Während die ersten beiden Regeln ausschließlich der Initialisierung von globalen Variablen oder dem Öffnen und Schließen einer Logdatei dienen, führt die *probable_cause* Regel die eigentliche Verknüpfung von Ereignissen durch. In *active_probing* werden für ein allgemeines AP Ereignis, spezifischere Ereignisse für zu prüfende Komponenten erstellt.

In der *startup* Regel, die immer beim Initialisieren des Event-Servers ausgelöst wird, können für die ganze Regel gültige Variablen festgelegt werden. Dies geschieht mit dem Prädikat `rerecord` und ist gleichzeitig ein

Abbildung 5.5: Übersicht über das *lrz_correlation rule set*

gutes Beispiel für die fehlende Dokumentation. Denn obwohl es in fast jedem Standardregelsatz angewendet wird, gibt es keine exakte Beschreibung dafür.

5.2.2.3 Duplikate erkennen

Eine Zielsetzung neben der Korrelation war die Verarbeitung von doppelt auftretenden Servcie Events. Diese Aufgabe übernimmt die Regel *duplicate_services*. Wie Listing 5.16 zeigt, untersucht die Regel alle eintreffenden Dienstereignisse.

Listing 5.16: Klassenfilter für die Regel *duplicate_services*

```
rule: duplicate_services:
(
  event: _event
  of_class _class within  ['TEC_LRZ_SERVICE_AVAIL_NOK',
                           'TEC_LRZ_SERVICE_AVAIL_OK',
                           'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
                           'TEC_LRZ_SERVICEFUNC_AVAIL_OK',
                           'TEC_LRZ_SERVICEFUNC_TIME_NOK',
                           'TEC_LRZ_SERVICEFUNC_TIME_OK'
                          ]
```

Wird die Regel also aktiviert, sucht sie nach nach weiteren Events im Eventcache. Dabei wird das Prädikat `first_duplicate` benutzt, welches das erste, den Duplikatsbedingungen entsprechende Event, das innerhalb eines festzulegenden Zeitraumes angekommen ist, findet. Um das Konzept der Score zu demonstrieren, wird der Schweregrad des „alten“ Events erhöht, was einem Administrator einen Hinweis auf ein dringendes Problem geben soll. Das nachfolgende Schließen des Evens (durch Statusänderung) verhindert, dass das Event später noch einmal von einer Regel verarbeitet wird. Ähnliche Funktion hat das Prädikat `commit_set`: Es verhindert, dass das Event von einem Regelset nochmals aufgegriffen wird. Wichtiges Kriterium für die Duplikatsfindung ist das Zeitfenster, in dem nach Duplikaten gesucht wird. In der *startup* Regel wurde mit einer globalen Variable ein Beispielwert (in Sekunden) festgelegt, der bei der praktischen Verwendung noch Anpassung benötigt.

5.2.2.4 Korrelation

Die *correlate* Regel ist die aufwändigste in dem Regelsatz. Eine Unterteilung in mehrere Aktionen hilft die Regel in mehrere logische Abschnitte zu gliedern. Diese Unterteilung hat jedoch auch einen funktionalen Grund. So wird zum Beispiel das Prädikat `all_instances` verwendet, welches ähnlich wie `first_duplicate` Events sucht, jedoch nicht nur eines, sondern alle, die den Bedingungen entsprechen. Laut [IBM c] werden alle diesem Aufruf bis zum Ende der aktuellen Aktion folgenden Prädikate für jedes entsprechende Event ausgeführt. So war es nötig eine neue Aktion zu erstellen, möchte man von diesem Prädikat wieder losgelöst weiterarbeiten. Die Unterteilung der Regel in mehrere Aktionen hilft auch eine bessere Übersicht über den Regelablauf zu gewinnen.

War zunächst noch geplant, Abhängigkeiten für Dienste in einer externen Datei festzuhalten, so wie es im E-Business Regelsatz angewendet wird, wurde dieses Vorhaben aus Vereinfachungsgründen unterlassen. Diese Einschränkung hat keine Auswirkung auf die Korrelationsergebnisse. Die Festlegung der Abhängigkeiten zwischen Diensten und Ressourcen erfolgt in der Aktion *correlate_resources*. Entsprechende Dienst-Dienst Abhängigkeiten werden analog in der späteren Aktion *correlate_services* definiert. Listing 5.17 zeigt anhand von Beispielen aus dem Webhosting und E-Mail Dienst, wie Abhängigkeiten definiert werden. Die Zeilennummern beziehen sich nur auf die *correlate* Regel.

Listing 5.17: Bestimmung von Abhängigkeiten

```
48  tl_str(lrz_tl, '\t<<Entering correlation of resources>>\n'),
49
50  % Based on received effect event class, define the possible cause classes
51       (
```

```
52    member(_class, ['TEC_LRZ_SERVICE_AVAIL_NOK','TEC_LRZ_SERVICEFUNC_AVAIL_NOK']),
53      _cause_classes = ['TEC_LRZ_RESOURCE_AVAIL_NOK']
54  ;
55      member(_class, ['TEC_LRZ_SERVICEFUNC_TIME_NOK']),
56      _cause_classes = ['TEC_LRZ_RESOURCE_CPU_UTIL_NOK']
57  ),
58
59  tl_str(lrz_tl, '\t...cause classes identified\n'),
60  (
61      _service == 'wh_getstat',
62      _dependent_resources = ['slb1','swk4-2wr','nx110','nx112','nx114','nx116']
63  ;
64      _service == 'wh_getdyn',
65      _dependent_resources = ['swm2-2wr','nas']
66  ;
67      _service == 'wh_dns',
68      _dependent_resources = ['dns1', 'dns2']
69  ;
70      _service == 'em_mailout',
71      _dependent_resources = ['lxmhs01', 'lxmhs02']
72  ;
73      _service == 'em_dns',
74      _dependent_resources = ['lxmhs19', 'lxmhs20']
75  ),
```

In den Zeilen 51 - 57 werden zunächst je nach eintreffendem Event (dessen Klasse die Variable `_class` speichert) die entsprechenden Ursacheneventklassen festgelegt:

- Für Ereignisse, die den Ausfall eines Dienstes oder einer Dienstfunktionalität beschreiben, kommen nur Ursachenereignisse in Frage, die den Ausfall einer Ressource propagieren.
- Für Ereignisse, die die Überschreitung einer Zeitbedingung bei einer Dienstfunktionalität beschreiben, wurde hier als Ursache eine hohe CPU Last bei einer Ressource festgelegt.

Diese Festlegungen sind bilden nicht die vollständige Abhängigkeitenstruktur ab, reichen jedoch aus, um die Möglichkeiten des Regelsatzes zu demonstrieren. Nach dieser Identifizierung von Abhängigkeiten, erfolgt in den Zeilen 61 - 74 die eigentliche Definition der Abhängigkeiten je nach Dienst des aktuellen Events. So werden zum Beispiel für den Webhostingdienst „Zugriff auf dynamische Seiten“, hier mit *wh_getdyn* bezeichnet, die Ursachenressourcen *swm2-2wr* und *nas* bestimmt. Unter der Voraussetzung, dass Ressourcen vorliegen, von denen der vom aktuellen Event betroffene Dienst abhängt, wird jetzt nach entsprechenden Events gesucht. Dabei kommt das weiter oben erwähnte Prädikat *all_instances* zum Einsatz. Werden jetzt hier Events unter entsprechenden Filterbedingungen gefunden, wird die betroffene Ressource aus der Liste der möglichen Fehlerquellen gelöscht (Zeile 106) und die Regel erzeugt in den Zeilen 110 - 120 (Regelauszug 5.18) ein *TEC_LRZ_PROBABLE_CAUSE_EVENT*. Die Anwendung eines Zeitfensters als Bedingung für die Korrelation beschreibt Zeile 97. Das aktuelle Event `_event` als Referenz, wird nach Ereignissen jeweils 120 Sekunden in die Vergangenheit und Zukunft gesucht. Dies wurde im Regelsatz auskommentiert, da sonst entgegen der Dokumentation kein Finden von nachfolgend eintreffenden Events möglich war.
Mit entsprechenden Filterbedingungen wäre hier die Anwendung des Konzepts der Score möglich: Zum Beispiel könnte man nur Events berücksichtigen, die einen enstsprechenden Schweregrad aufweisen (im Listing angedeutet).

Listing 5.18: Suchen nach Events für abhängige Ressourcen

```
84 % Search for cause events
85  not empty_list(_dependent_resources),
86  (
87  tl_str(lrz_tl, '\t...searching for cause events...\n'),
88  all_instances(
89  event: _cause_event
90  of_class within _cause_classes
```

```
91    where [% Score filter conditions here
92      status: _cause_status outside ['CLOSED'],
93      resource: _cause_resource within _dependent_resources,
94      date_reception: _cause_date outside _linked_cause_dates,
95      event_handle: _cause_event_handle outside _linked_cause_handles
96    ],
97    %_event - 120 - 120 %Commented out to find earlier events
98  ),
99
100 bo_get_class_of(_cause_event, _cause_class),
101 tl_fmt(lrz_tl, '\t...found %s event on resource %s\n',
      [_cause_class, _cause_resource]),
102
103 % Remove this _cause_resource from list of dependents
104 _default = [],
105 get_global_var('lrz_correlation', 'dependent_resources',
      _new_dependent_resources, _default),
106 delete(_new_dependent_resources, _cause_resource,
      _restof_dependent_resources),
107 set_global_var('lrz_correlation', 'dependent_resources',
      _restof_dependent_resources),
108
109 % Generate probable cause event
110 generate_event('TEC_LRZ_PROBABLE_CAUSE_EVENT',
111   [
112   effect_service=_service,
113   effect_event_handle=_event_handle,
114   effect_event_date=_date,
115   effect_class=_class,
116   cause_event_handle=_cause_event_handle,
117   cause_event_date=_cause_date,
118   cause_class=_cause_class
119   ]
120 ),
121 tl_str(lrz_tl, '\t...TEC_LRZ_PROBABLE_CAUSE_EVENT generated\n')
```

Das Erzeugen dieses Events führt zur Aktivierung der bereits erwähnten Hilfsregel *probable_cause*. Hier wird der Verweis des gefundenen Ursachenevents in die Liste des aktuellen Service Events angefügt. Gleichzeitig könnte in einer realen Umgebung dieses Event zur Benachrichtigung eines Administrators dienen. Das gezielte Anfordern aktiver Überprüfungen von Ressourcen erfolgt in der nächsten Aktion *check_resource_restlist*. Listing 5.19 zeigt einen Ausschnitt dieser Aktion, wo ein *TEC_LRZ_ACTIVE_PROBING_EVENT* erzeugt wird. Die Regel *active_probing* erzeugt wiederum einzelne Events für jede zu überprüfende Ressource. Details dazu werden unter 5.2.2.5 formuliert.

Listing 5.19: Anfordern von Active Probing

```
134 generate_event('TEC_LRZ_ACTIVE_PROBING_EVENT',
135 [
136   sender_handle=_event_handle,
137   sender_date=_date,
138   resources=_probe_resources
139 ]
140 ),
```

Im nächsten Schritt der Regel wird auf analoger Weise mit den Dienst-Dienst Abhängigkeiten verfahren. Wiederum erfolgt zunächst die Festlegung der möglichen Ursachenklassen und der Dienste, von denen der Dienst des aktuellen Events abhängt. Anschließend wird nach passenden Ereignissen gesucht und gezieltes Active Probing angefordert.

Fügte man keine weiteren Aktionen dieser Regel hinzu, so könnten nur Ereignisse korreliert werden, die in

„richtiger“ chronologischer Reihenfolge am Event-Server eingetroffen sind. Mit „richtig“ ist gemeint, dass zuerst die Events für Komponenten eintreffen, die in der Abhängigkeitenhierarchie unterhalb eines übergeordneten Dienstes liegen. Um diese Einschränkung für Service Events aufzuheben, wurde die Aktion *do_redo* angefügt.
5.20 zeigt einen Ausschnitt aus dieser Aktion. Hier wird nach schon früher eingegangenen, den aktuellen Dienst betreffenden, Active Probing Events gesucht (Zeilen 271 - 280). Wird eine solches gefunden, wird es zunächst geschlossen (Zeile 282), um es als abgearbeitet zu markieren. Die Aktion fordert eine wiederholte Analyse des Service Events an, welches dieses Active Probing Event erzeugt hat. Dies geschieht mit dem Prädikat `redo_\-analysis` in Zeile 294. Bei dieser Analyse kann jetzt das aktuelle Event als Ursachenevent für ein anderes Servcie Event identifiziert werden. Analoges auf Ressourcenebene führt die Regel *resource_handler* durch. Wie dies am Beispiel aussieht, kann im folgenden Abschnitt 5.3 an angegebenen Debugausgaben verfolgt werden.
Bei der wiederholten Analyse wird die komplette *correlate* Regel durchlaufen. Um zum Beispiel nicht überflüssige AP Events zu erzeugen, wurden einige Aktionen mit dem Schlüsselwort `reception_action:` und nicht mit `action:` gekennzeichnet. Der Unterschied liegt darin, dass die mit erstem deklarierten Aktionen nicht bei einer Redo-Analyse ausgeführt werden.

Listing 5.20: Reanalyse für Dienstereignisse anfordern

```
269  % Request redoanalysis of events previous to this
270  tl_fmt(lrz_tl, '\t...searching for AP event for service %s...\n', _service),
271  all_instances(
272    event: _ap_event
273    of_class 'TEC_LRZ_ACTIVE_PROBING_SERVICE'
274    where [
275                   status: outside ['CLOSED'],
276                            service: equals _service,
277                            sender_handle: _sender_handle,
278                            sender_date: _sender_date
279                     ]
280  ),
281  tl_fmt(lrz_tl, '\t...found AP event for %s\n', [_service]),
282  change_event_status(_ap_event, 'CLOSED'),
283  tl_str(lrz_tl, '\t...searching for service event\n'),
284  first_instance(
285    event: _se_event
286      of_class within ['TEC_LRZ_SERVICE_AVAIL_NOK',
           'TEC_LRZ_SERVICEFUNC_AVAIL_NOK']
287      where [
288                              status: within ['ACK'],
289                              date_reception: equals _sender_date,
290                              event_handle: equals _sender_handle
291                     ]
292  ),
293  tl_str(lrz_tl, '\t...request redo analysis of previous service event\n'),
294  redo_analysis(_se_event)
```

Die letzte Aktion *start_timer* Regel, die auch nicht bei einer wiederholten Analyse ausgeführt wird, setzt einen Timer für das aktuelle Event. Was beim Ablauf dieses Timers, dessen Zeit in einer globalen Variable definiert wird, geschieht, beschreibt 5.2.2.6. In den Eventklassen ist für Dienstereignisse zusätzlich das Attribut `valid_to` vorgesehen. Hier könnte man für jedes *Service Event* individuell eine Gültigkeitsdauer festlegen, die in dieser Regel verwendet werden könnte.

5.2.2.5 Resource Handler

Sollen auch Ressourcenereignisse mit Dienstereignissen korreliert werden, die erst durch die Anforderung von Active Probing entstanden sind, wird eine weitere Regel nötig. Diese wird ausschließlich auf Ressourcenevents hin aktiviert.

Listing 5.21 zeigt einen Ausschnitt aus dieser Regel. Die Zeilennummern beziehen sich wiederum nur auf diese Regel. Zunächst wird überprüft, ob für die das Event betreffende Ressource eine Anforderung für Active Probing vorliegt (Zeilen 18 - 28). Ist dies der Fall, wird für das Dienstereignis, welches diese Anforderung ausgelöst hat, eine Reanalyse (Zeile 47) verlangt.

Listing 5.21: Verarbeitung von Ressourcenereignissen

```
16  % Request redoanalysis of events previous to this
17  tl_fmt(lrz_tl, '\t...searching for AP event for resource %s...\n', _resource),
18  all_instances(
19    event: _ap_event
20    of_class 'TEC_LRZ_ACTIVE_PROBING_RESOURCE'
21    where [
22      status: outside ['CLOSED'],
23      resource: equals _resource,
24      sender_handle: _sender_handle,
25      sender_date: _sender_date
26    ],
27   _event -120 -0
28  ),
[...]
46  tl_str(lrz_tl, '\t...request redo analysis of previous service event\n'),
47  redo_analysis(_se_event)
```

5.2.2.6 Timer Regel

Wie oben beschrieben, wird diese Regel mit dem Ablauf eines Timers, welcher für ein Service Event gestartet wurde, aktiv. Hier wird überprüft, ob in dem Zeitraum zwischen Eintreffen des Service Events und Ablauf des Timers eine mögliche Fehlerursache gefunden wurde. Ist dies nicht der Fall, könnte hier eine Weiterleitung an einen Administrator oder an ein fallbasiertes System wie in [Hane 07] implementiert werden.
In jedem Fall aber führt der Ablauf des Timers zu einer Schließung des Service Events, um die Menge an aktiven Ereignissen im Event-Server zu reduzieren.

5.3 Resultate

5.3.1 Versuch 1

Um die Funktionalität des im vorigen Abschnitt beschriebenen Regelsatzes zu demonstrieren, wird in diesem ersten Versuch ein Testfall für den Webhostingdienst vorgestellt. Wie schon beschrieben, werden als Quelle für Ereignisse keine realen Komponenten verwendet, sondern Events mit einem Befehl der TEC erzeugt. Der beispielhafte Aufruf (auf einem UNIX System)

```
wpostemsg service=wh_getstat TEC_LRZ_SERVICE_AVAIL_NOK LOGFILE
```

schickt ein Service Event mit dem auf „wh_getstat“ gesetzten Attribut `service` an den Event Server. Als Quelle wurde der Typ Logfile gewählt, der in der weiteren Betrachtung keine Rolle mehr spielt. Um sich den Inhalt des Eventcaches anzeigen zu lassen, kann mit dem Befehl

```
wtdumper -d
```

eine formatierte Ausgabe von eingetroffenen oder generierten Ereignissen ausgegeben werden. Weitere Befehle sind im Anhang A zu finden.

In den folgenden Paragraphen wird in zeitlicher Reihenfolge die Funktionalität des *correlation* Regelsatzes demonstriert. Diese kann in Abbildung 5.6 nachvollzogen werden. Dort stehen Rechtecke für Ressourcenevents, Kreise für Service Events und Rauten für Events, die von der Regel generiert wurden oder informellen Charakter haben.

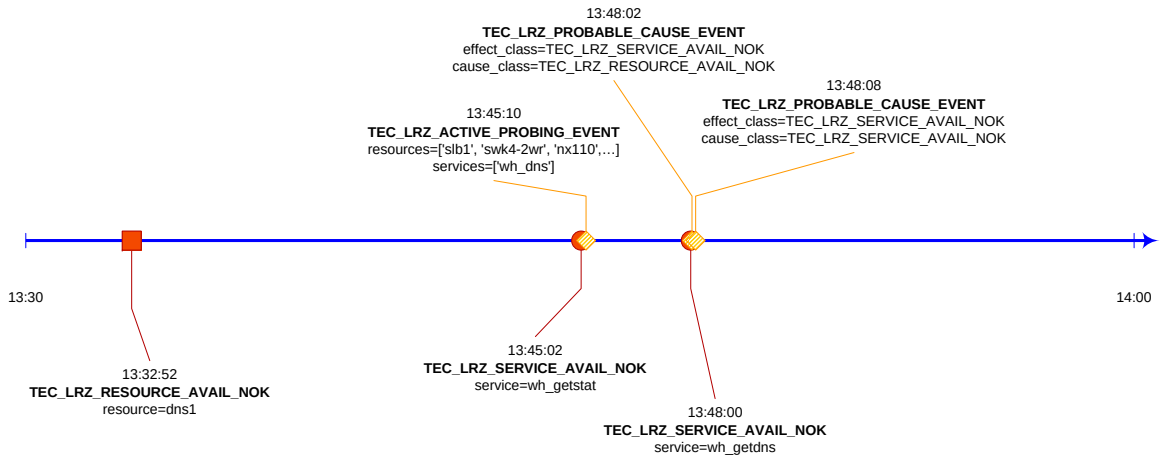


Abbildung 5.6: Zeitlicher Abfolge der eintreffenden und generierten Events

Schritt 1 Es wird angenommen, dass einer der beiden für die Namensauflösung zuständigen Hosts *dns1* ausgefallen ist und ein entsprechendes Resource Event erzeugt wurde. Ein *Service Event*, den Ausfall des Zugriffs auf statische Seiten (*wh_getstat*) beim Webhostingdienst beschreibend, erreicht den Event-Server. Beim Eintreffen dieses Events wird zunächst überprüft, ob eine Active Probing Anforderung für diese Ressource vorliegt, was hier nicht der Fall ist. Die Regel *resource_handler* bleibt hier ohne Wirkung:

Listing 5.22: Debugausgabe

```
<<Entering resource_handler rule>>
        ...searching for AP event for resource wh_getstat...
<<Exiting resource_handler rule>>
```

Bei der Ankuft des Service Events wird zunächst nach Duplikaten gesucht, aber keine gefunden. Die *correlation_rule* findet sechs Ressourcen und einen Dienst (den *wh_dns* Dienst), von denen *wh_getstat* direkt abhängt. Da bis jetzt keine weiteren Events vorliegen, wird Active Probing für diese untergeordneten Ressourcen und Dienste angefordert. Die eigentliche Fehlerursache, nämlich die fehlerhafte Resource *dns1* taucht hier nicht auf, da nur der *wh_dns* Dienst von dieser direkt abhängig ist.

Listing 5.23: Debugausgabe

```
<<Entering duplicate rule>>
        ...processing TEC_LRZ_SERVICE_AVAIL_NOK event
<<Exiting duplicate rule>>

<<Entering correlation rule>>
        ...processing TEC_LRZ_SERVICE_AVAIL_NOK event on service wh_getstat
                                                with 0 already linked events
        <<Entering correlation of resources>>
        ...cause classes identified
        ...6 dependent resources for service wh_getstat identified
        ...searching for cause events...
        ...TEC_LRZ_ACTIVE_PROBING_EVENT generated for 6 resource(s)
        <<Exiting correlation of resources>>
        <<Entering correlation of services>>
        ...1 dependent subservices for service wh_getstat identified
        ...searching for cause events...
        ...TEC_LRZ_ACTIVE_PROBING_EVENT generated for 1 service(s)
```

```
        <<Exiting correlation of services>>
        <<Entering redo request>>
        ...searching for AP event for service wh_getstat...
<<Exiting correlation rule>>

<<Entering active probing rule>>
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for slb1 generated
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for swk4-2wr generated
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for nx110 generated
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for nx112 generated
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for nx114 generated
        ...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for nx116 generated
<<Exiting active probing rule>>

<<Entering active probing rule>>
        ...TEC_LRZ_ACTIVE_PROBING_SERVICE event for wh_dns generated
<<Exiting active probing rule>>
```

Der Cache zeigt das eingetroffene Event, sowie die generierten *Active Probing Events*.

Listing 5.24: Eventcache (gekürzt)

```
TEC_LRZ_RESOURCE_AVAIL_NOK;
        server_handle=1;
        date_reception=1166185972;
        event_handle=1;
{*...*}
        resource=dns1;
END

TEC_LRZ_SERVICE_AVAIL_NOK;
        server_handle=1;
        date_reception=1166186702;
        event_handle=1;
{*...*}
        linked_cause_dates=[];
        linked_cause_handles=[];
        service=wh_getstat;
{*...*}
END

TEC_LRZ_ACTIVE_PROBING_EVENT;
        server_handle=1;
        date_reception=1166186709;
        event_handle=1008;
{*...*}
        resources=['slb1', 'swk4-2wr', 'nx110', 'nx112', 'nx114', 'nx116'];
{*...*}
END

{*...*}

TEC_LRZ_ACTIVE_PROBING_RESOURCE
        server_handle=1;
        date_reception=1166192131;
        event_handle=1008;
{*...*}
        service=wh_dns;
{*...*}
END
```

Schritt 2 In diesem Schritt wird angenommen, dass die Anforderung zur Überprüfung des *wh_dns* Dienstes ein Service Event generiert hat. Wie im ersten Schritt wird jetzt für dieses Ereignis nach möglichen Fehlern gesucht, und das zuerst eingegangene Event für die Ressource *dns1* gefunden, ein *TEC_LRZ_PROBABLE_CAUSE_EVENT* erstellt und die Events korreliert.
Die Suche nach vorhergehenden Service Events in der *do_redo* Aktion führt zu dem ersten Event für den *wh_getstat* Dienst. Für diesen wird eine wiederholte Analyse angefordert (`request redo analysis`). Daraufhin kann der *wh_getstat* Dienst als Fehler identifiziert und wiederum ein entsprechendes *TEC_LRZ_PROBABLE_CAUSE_EVENT* generiert werden.

Listing 5.25: Debugausgabe

```
<<Entering correlation rule>>
        ...processing TEC_LRZ_SERVICE_AVAIL_NOK event on service wh_dns
                                                with 0 already linked events
        <<Entering correlation of resources>>
        ...cause classes itentified
        ...2 dependent resources for service wh_dns identified
        ...searching for cause events...
        ...found TEC_LRZ_RESOURCE_AVAIL_NOK event on resource dns1
        ...TEC_LRZ_PROBABLE_CAUSE_EVENT generated
        ...TEC_LRZ_ACTIVE_PROBING_EVENT generated for 1 resource(s)
        <<Exiting correlation of resources>>
        <<Entering correlation of services>>
        ...no services found for active probing
        <<Exiting correlation of services>>
        <<Entering redo request>>
        ...searching for AP event for service wh_dns...
        ...found AP event for wh_dns
        ...searching for service event
        ...request redo analysis of previous service event
<<Exiting correlation rule>>
{*...*}
<<Entering probable cause rule>>
        ...searching for effect event TEC_LRZ_SERVICE_AVAIL_NOK...
        ...found TEC_LRZ_RESOURCE_AVAIL_NOK event
        ...Events correlated!
<<Exiting probable cause rule>>

<<Entering correlation rule>>
        ...processing TEC_LRZ_SERVICE_AVAIL_NOK event on service wh_getstat
                                                with 0 already linked events
{*...*}
        <<Entering correlation of services>>
        ...1 dependent subservices for service wh_getstat identified
        ...searching for cause events...
        ...found TEC_LRZ_SERVICE_AVAIL_NOK event on service wh_dns
        ...TEC_LRZ_PROBABLE_CAUSE_EVENT generated
        <<Exiting correlation of services>>
<<Exiting correlation rule>>
{*...*}
<<Entering probable cause rule>>
        ...searching for effect event TEC_LRZ_SERVICE_AVAIL_NOK...
        ...found TEC_LRZ_SERVICE_AVAIL_NOK event
        ...Events correlated!
<<Exiting probable cause rule>>
```

Listing 5.26 zeigt zuerst das Service Event für den DNS Dienst, das Event für die wahrscheinliche Fehlerursache *dns1*, und schließlich das Event für die wahrscheinliche Fehlerursache des *wh_getstat* Dienstes.

Listing 5.26: Eventcache (gekürzt)

```
TEC_LRZ_SERVICE_AVAIL_NOK;
        server_handle=1;
        date_reception=1166186880;
        event_handle=1;
{*...*}
        linked_cause_dates=[];
        linked_cause_handles=[];
        service=wh_dns;
{*...*}
END

TEC_LRZ_PROBABLE_CAUSE_EVENT;
        server_handle=1;
        date_reception=1166186882;
        event_handle=1011;
{*...*}
        cause_date_reception=0;
        cause_event_handle=1;
        cause_class=TEC_LRZ_RESOURCE_AVAIL_NOK;
        cause_event_date=1166185972;
        effect_class=TEC_LRZ_SERVICE_AVAIL_NOK;
        effect_event_date=1166186880;
        effect_event_handle=1;
        effect_service=wh_dns;
END

TEC_LRZ_PROBABLE_CAUSE_EVENT;
        server_handle=1;
        date_reception=1166186882;
        event_handle=1012;
{*...*}
        cause_date_reception=0;
        cause_event_handle=1;
        cause_class=TEC_LRZ_SERVICE_AVAIL_NOK;
        cause_event_date=1166186880;
        effect_class=TEC_LRZ_SERVICE_AVAIL_NOK;
        effect_event_date=1166186702;
        effect_event_handle=1;
        effect_service=wh_dns;
END
```

5.3.2 Versuch 2

Dieser zweite Versuch demonstriert, wie der Regelsatz mit Abhängigkeiten zwischen Diensten und Subfunktionalitäten umgeht. Zudem zeigt er, wie Ereignisse für Ressourcen, die nach einem *Service Event* eintreffen, zum Beispiel auf eine Active Probing Anforderung hin, korreliert werden können. Wie schon im ersten Versuch hilft eine Abbildung (5.7) den Versuch nachzuvollziehen. Dessen Semantik ist gleich der in Abbildung 5.6. Die kurzen Zeitabstände sind auf die zu Testzwecken stark verkürzten Zeitfenster in der Regel zurückzuführen.

Ein Service Event beschreibt zunächst den Ausfall des E-Maildienstes (*em_mail*). Für diesen sind im Regelsatz die beiden Funktionalitäten *em_mailin* und *em_mailout* festgelegt. Nach der Anforderung für Active Probing trifft ein `TEC_LRZ_SERVICEFUNC_TIME_NOK` Event für den *em_mailout* Dienst ein. Grund für diese Überschreitung einer Zeitbedingung ist eine (ungewöhnlich) hohe CPU Auslastung auf zwei Ressourcen (hier: *lxmhs01* und *lxmhs02*), von welcher die Funktionalität abhängig ist. In der Praxis könnte diese Situation

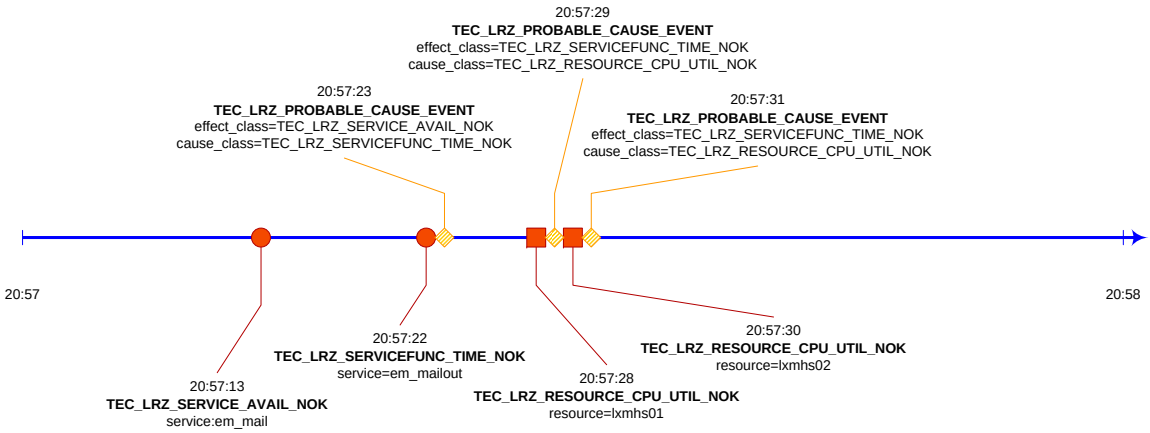


Abbildung 5.7: Zeitlicher Abfolge der eintreffenden und generierten Events

möglicherweise auf Grund einer übermäßigen Anzahl zu verarbeitender Spam Mails (Spam) entstehen. Auch dieses Events treffen erst nach der Anforderung zur Überprüfung in das System ein.
Dieser Versuch zeigt sehr deutlich, wie einer sehr abstrakten Fehlermeldung, die eher ein Symptom beschreibt („Mein E-Mail Zugang funktioniert nicht“), einem realen Mißstand auf der Ressourcenebene zugeordnet werden kann. Als zusätzlichen Aspekt wird die Aufhebung der *single root cause* Annahme demonstriert, da zwei Events als Ursache identifiziert werden können.

Die Debugausgabe wird hier nur stark gekürzt wiedergegeben, da die meisten Ausgaben schon oben gezeigt wurden. Zunächst wird gezeigt, wie auf Grund einer Re-Analyse des Events für den *em_mail* Dienst, das Event für die *em_mailout* Funktionalität als Ursache gefunden und korreliert wird.

Listing 5.27: Debugausgabe

```
<<Entering correlation rule>>
        ...processing TEC_LRZ_SERVICE_AVAIL_NOK event on service em_mail
           with 0 linked events
        <<Entering correlation of resources>>
        ...cause classes itentified
        <<Exiting correlation of resources>>
        <<Entering correlation of services>>
        ...2 dependent subservices for service em_mail identified
        ...searching for cause events...
        ...found TEC_LRZ_SERVICEFUNC_TIME_NOK event on service em_mailout
        ...TEC_LRZ_PROBABLE_CAUSE_EVENT generated
        <<Exiting correlation of services>>
<<Exiting correlation rule>>

<<Entering probable cause rule>>
        ...searching for effect event TEC_LRZ_SERVICE_AVAIL_NOK...
        ...found TEC_LRZ_SERVICE_AVAIL_NOK event
        ...Events correlated!
<<Exiting probable cause rule>>
```

Der nächste Ausschnitt zeigt die Ausgaben der *resource_handler* Regel nach dem Eintreffen des Events für die Komponenten *lxmhs01* und *lxmhs02*.

Listing 5.28: Debugausgabe

```
<<Entering resource_handler rule>>
        ...searching for AP event for resource lxmhs01...
        ...found AP event for lxmhs01 with date 1167076642
        ...searching for service event
        ...request redo analysis of previous service event
<<Exiting resource_handler rule>>

<<Entering resource_handler rule>>
        ...searching for AP event for resource lxmhs02...
        ...found AP event for lxmhs02 with date 1167076642
        ...searching for service event
        ...request redo analysis of previous service event
<<Exiting resource_handler rule>>
```

Welche Auswirkungen die Events auf den Eventcache haben zeigt der letzte Ausschnitt. Wichtig sind hier besonders die mit `(*)` und `(**)` gekennzeichneten Zeilen, welche jeweils einen Verweis auf das verknüpfte Event zeigen. Zum Beispiel entspricht der Zeitstempel und das Eventhandle in `(**)` einem Verweis auf das *TEC_LRZ_RESOURCE_CPU_UTIL_NOK* Ereignis.

Listing 5.29: Eventcache (gekürzt)

```
TEC_LRZ_SERVICE_AVAIL_NOK;
        server_handle=1;
        date_reception=1167076633;
{*...*}
        linked_cause_dates=[ '1167076642'];                              (*)
        linked_cause_handles=[ '1'];                                     (*)
        service=em_mail;
{*...*}
END


TEC_LRZ_SERVICEFUNC_TIME_NOK;
        server_handle=1;
        date_reception=1167076642;
{*...*}
        linked_cause_dates=[ '1167076650'];                              (**)
        linked_cause_handles=[ '1'];                                     (**)
        service=em_mailout;
{*...*}
END


TEC_LRZ_PROBABLE_CAUSE_EVENT;
        server_handle=1;
        date_reception=1167076642;
{*...*}
        cause_event_handle=1;
        cause_class=TEC_LRZ_SERVICEFUNC_TIME_NOK;
        cause_event_date=1167076642;
        effect_class=TEC_LRZ_SERVICE_AVAIL_NOK;
        effect_event_date=1167076633;
        effect_event_handle=1;
        effect_service=em_mail;
```

```
END

TEC_LRZ_RESOURCE_CPU_UTIL_NOK;
        server_handle=1;
        date_reception=1167076648;
{*...*}
        resource=lxmhs01;
        utilization=97.000000e+00;
END

TEC_LRZ_RESOURCE_CPU_UTIL_NOK;
        server_handle=1;
        date_reception=1167076650;
{*...*}
        resource=lxmhs02;
        utilization=95.000000e+00;
END

TEC_LRZ_PROBABLE_CAUSE_EVENT;
        server_handle=1;
        date_reception=1167076648;
{*...*}
        cause_event_handle=1;
        cause_class=TEC_LRZ_RESOURCE_CPU_UTIL_NOK;
        cause_event_date=1167076648;
        effect_class=TEC_LRZ_SERVICEFUNC_TIME_NOK;
        effect_event_date=1167076642;
        effect_event_handle=1;
        effect_service=em_mailout;
END

TEC_LRZ_PROBABLE_CAUSE_EVENT;
        server_handle=1;
        date_reception=1167076650;
{*...*}
        cause_event_handle=1;
        cause_class=TEC_LRZ_RESOURCE_CPU_UTIL_NOK;
        cause_event_date=1167076650;
        effect_class=TEC_LRZ_SERVICEFUNC_TIME_NOK;
        effect_event_date=1167076642;
        effect_event_handle=1;
        effect_service=em_mailout;
END
```

5.4 Zusammenfassung

In diesem letzten Abschnitt des Kapitels soll auf die erreichten Ziele und kurz auf die Probleme, die beim Umgang mit der TEC aufgetreten sind, eingegangen werden.

Im ersten Schritt wurden die bei der Auslieferung bereitgestellten Möglichkeiten zur Ereigniskorrelation auf ihre Eignung zur Einbeziehung von Diensten in den Korrelationsprozess geprüft. Diese Analyse erstreckte sich auf drei Regelsätze, welche dem Kunden ein weites Spektrum an schnell einsatzfähigen Regeln zur Überwachung von Ressourceninfrastrukturen bieten und einen hohen Prozentsatz der Kundenwünsche abdecken. Jedoch konnte ein Großteil der Anforderungen zur dienstorientierten Ereigniskorrelation nicht erfüllt werden, wie die Versuche zeigten. Neben fehlender Funktionalität, waren zu starke Einschränkungen auf bestimmte Spezialfälle die Gründe für dieses Scheitern.

Die Ergebnisse dieser Betrachtungen zeigten, dass nur die Neuentwicklung von Regeln die Konzepte der allge-

meinen Vorgehensweise umsetzen kann. Die Übertragung Teile der Konzeptionen aus Kaptiel 4 gelang unter bestimmten Voraussetzungen. Der Funktionsumfang der TEC konnte mit diesen Regeln auf einen Einbezug von Diensten erweitert werden. Für einige Elemente, wie das Scorekonzept wurden bei der Regelerstellung Andeutungen gemacht, wie sie zum Beispiel in Filterprozesse eingebunden werden können.

Bei der Implementierung dieser neuen Regeln bereiteten insbesondere Merkmale der verwendeten Programmiersprache Prolog Schwierigkeiten. Ohne erweiterte Vorkenntnisse in dieser Logiksprache ist sehr schwierig den Ablauf der Regeln zu verstehen. IBM verwendet zahlreiche proprietäre Prädikate, deren Implementierung dem Anwender in kompilierten `*.wic` Dateien verborgen bleibt. Dazu fehlt jegliche Dokumentation zu einigen dieser. Besondere Probleme bereitete die Ablaufsteuerung innerhalb der Regeln, da Konstrukte wie `if-then-else` in dieser Form nicht existieren. Einschneidende Restriktionen bewirkte die Gültigkeitsdauer von Variablen, die sich in der Regel nur über eine Aktion hinweg erstreckt. Um auf Variablen über mehrere Aktionen hinweg zugreifen zu können, musste auf globale Variablen zugegriffen werden. Ein bedeutender Teil der Regelentwicklung beschäftigte sich damit, das Konzept der TEC, dass immer nur zwei Events miteinander korreliert werden können, was auch zur *single root cause* Annahme führt, aufzuheben und zu umgehen.

Ein entscheidender Unterschied war, dass sich die Wissensbasis bei der TEC ausschließlich aus Informationen über Events zusammensetzt, während sie sich in der Vorgehensweise dieser Arbeit im Wesentlichen aus Status- und Abhängigkeitsinformationen über die verwalteten Komponenten zusammensetzt. Trotz dieser konzeptuellen Differenzen konnten Teile der allgemeinen Vorgehensweise umgesetzt werden.

Wie bereits erwähnt ist der gesamte neu entwickelte Regelsatz und die `*.baroc` Datei mit den Eventdefinitionen im Anhang B und C zu finden.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Vorgehensweise modelliert, die IT-Providern bei der Gestaltung eines dienstorientierten Fehlermanagements unterstützen kann.

Diese Vorgehensweise beschreibt einen regelbasierten Ansatz, der darstellt, wie Ereignisse in einer typischen Providerinfrastruktur miteinander verknüpft werden können. Das Ziel war eine möglichst schnelle Ursachenfindung in der Ressourcenebene für Fehler oder Symptome auf einer höheren Dienstebene.

Dazu wurde zunächst ein Anforderungskatalog ausgearbeitet, dessen Inhalte schrittweise erfüllt werden konnten.

In einer zukünftigen Arbeit ([Hane 07]) wird ein umfassenderes Rahmenwerk vorgestellt, welches unter anderem auch die Kombination mit einer fallbasierte Komponente zur Problemlösung vorstellt.

Möchte ein Dienstanbieter ein System zur Überwachung seiner angebotenen Dienste und seiner zur Verfügung stehenden Ressourceninfrastruktur benutzen, so steht er vor einem entscheidenden Zielkonflikt:
Modelliert er seine IT-Infrastruktur möglichst akkurat, erhält er entsprechend präzisere Ergebnisse seines Fehlermanagementsystems. Bei wesentlich geringerem Modellierungsaufwand kann er unter Umständen nur unwesentlich schlechtere Ergebnisse erzielen. Aus diesem Grund sind die meisten der in Kapitel 4 aufgezeigten Konzepte optionaler Natur, so dass deren Wegfall eine grundlegendere Form der Korrelation ermöglicht.

In Kapitel 5 wurden die entwickelten Konzepte zur Dienstorientierung auf einem verbreitetem Softwareprodukt für das Ressourcenmanagement angewendet. Hier wurde die Tivoli Enterprise Console (TEC) von IBM ausgewählt, da hier dem Anwender ein breites Spektrum an Eingriffsmöglichkeiten offen steht, welches genutzt werden konnte, um die entwickelten Konzepte einzubringen. Eine detaillierte Betrachtung der vorhandenen Eingriffsmöglichkeiten in das Tivoli System zeigte, dass diese nicht im gewünschten Umfang für eine dienstorientierte Ereigniskorrelation verwendet werden können.
Aus diesem Grund wurden die in den Abschnitten 5.2.1.2 und 5.2.2 entwickelten Ereignisklassen und Regelsätzen neu in die TEC implementiert. Dadurch wurde die TEC um neue Funktionalitäten erweitert.

Die in praxisnahen Versuchen erreichten Ergebnisse belegen eine erfolgreiche Verknüpfung zwischen abstrakten Ereignissen einen Dienst betreffend und realen Problemen auf Seiten der Ressourceninfrastruktur.

Das Konzept kann insbesondere durch folgende Punkte besser in die TEC integriert werden.

- Optimierung der Regelperformanz
  In einigen Regeln ist durch intensivere Verwendung von Prologkonzepten eine Steigerung der Performanz möglich.
- Herauslösen von Abhängigkeiteninformationen aus den Regeln
  Für den eigentlichen Korrelationsablauf nicht entscheidend war die feste Implementierung der Abhängigkeiteninformationen in den Regelsätzen. Hier wäre eine Lösung möglich, die diese Informationen beispielsweise aus einer Datei einliest.
- Integration von realen Ereignissen
  Die entsprechende Verwendung des Eventadapter Konzepts von IBM ermöglicht eine Einbindung von realen Ereignissen - zum Beispiel eines Ressourcenüberwachungssystems.

Nach diesen Optimierungen wird ein Praxiseinsatz des entwickelten Regelsatzes möglich.

Um nachfolgenden Arbeiten die Fortführung der Entwicklung dieses Regelsatzes zu erleichtern und weitergehende Arbeiten in diesem Bereich zu gewährleisten, wurde im Anhang A eine Dokumentation zum Umgang mit der TEC und den neuen Regeln formuliert.

A Verwenden der Tivoli Enterprise Console

Dieser Anhang beschreibt die Verwendung der Tivoli Enterprise Console, welche auf einem Rechner am LRZ installiert ist. Mit dieser „Anleitung“ soll es nachfolgenden Studenten erleichtert werden den implementieren Regelsatz zu erweitern oder neue zu erstellen. Dabei wird ausschließlich auf die Verwendung von Konsolenbefehlen (UNIX) eingegangen. Die Verwendung des graphischen Editors wird nicht erklärt.
Die verwendeten Befehle und Hinweise wurden hauptächlich mit Hilfe von [IBM b], [IBM d], [IBM c] und [IBM a] erstellt.

A.1 Zugriff und Rechte

Prinzipiell stehen dem Nutzer zwei Möglichkeiten offen auf diesem Rechner zu arbeiten. Entweder man arbeitet direkt in der LRZ Rechnerhalle oder von Zuhause mit Hilfe des LRZ VPN Clients (erhältlich auf der LRZ Webseite). In beiden Fällen muss zunächst eine SSH Verbindung zu einem der Gatewayrechner *wsc33.lrz-muenchen.de* oder *wsc22.lrz-muenchen.de* hergestellt werden. Dazu benötigt man eine AFS Kennung. Von dort erreicht man dann wiederum durch SSH den Rechner namens *soec* („soec“ steht für *service oriented event correlation*).

Möchte man Regelsätze verändern (also auch kompilieren und laden) dürfen, sind **root** Rechte auf *soec* notwendig.

> **Hinweis**: Um Dateien von der soec kopieren zu können, müssen diese zuerst mit dem Programm `scp` auf die Gateway Maschine kopiert werden. Erst von dort können sie aus dem LRZ Netz heraus geholt werden. Verwendet man unter Windows (zum Beispiel von Zuhause) das Programm `pscp`, so muss mit der Option `-SCP` explizit das SCP Protokoll verwendet werden.

Ist man erfolgreich auf *soec* angemeldet, so muss zunächst der Befehl

```
source /etc/Tivoli/setup_env.sh
```

ausgeführt werden, um die TEC-eigenen Befehle verwenden zu können. Jetzt kann zunächst geprüft werden, ob der Ereignisserver der TEC überhaupt läuft. Diese Information gewinnt man mit dem Befehl `wstatesvr`. Das Beenden des Ereignisservers erreicht man mit `wstopesvr` - das Starten mit `wstartesvr`. Beide Befehl haben ca. 10 Sekunden Ausführungszeit. Ersterer beendet den Server bei Erfolg ohne Rückmeldung.

A.2 Arbeiten mit Regelbasen

Um Regelbasen der TEC zu verändern, sind vor allem Kenntnisse des Befehls `wrb` der TEC notwendig. Hilfe dazu findet man in [IBM a] ab Seite 74.

Die bisher erstellen Regelbasen wurden in den Verzeichnissen `/tivoli/server/work` oder `/tivoli/server/hans_rb` abgelegt.
Zur Erstellung einer eigenen Regelbasis (mit dem eindeutigen Namen `lrz02`) verwendet man den Befehl. Der Name muss dabei auf dem Eventserver eindeutig sein.

```
wrb -crtrb -path /tivoli/server/work/lrz02 lrz02
```

Dieser Befehl erstellt u.a. drei Unterverzeichnisse `TEC_RULES`, `TEC_CLASSES` und `TEC_TEMPLATES`, in welchen auch die Regelsatzdateien (`*.rls`) und Ereignisdefinitionen (`*.baroc`) Platz finden.
Bei der weiteren Vorgehensweise bieten sich zwei Möglichkeiten

1. Regelbasis komplett neu erstellen
2. Vorhandene Regelbasis übernehmen und modifizieren

Bei Möglichkeit 2 ist es notwendig, die Quellregelbasis in die neu erstellte Basis zu kopieren. Möchte zum Beispiel von der `default` Regelbasis kopieren, erledigt dies der Befehl

```
wrb -cprb default lrz02
```

> **Hinweis**: Vor dem Kopieren von Regelbasen muss beachtet werden, dass die Quellregelbasis keine Fehler enthalten darf, da sonst der Kopiervorgang fehlschlägt. Ist dies einmal geschehen, muss man die Option `-overwrite` angeben, um die bereits kopierten Dateien zu überschreiben. Mit der Option `-force` wird diese Fehlerprüfung unterdrückt.

Jetzt können die Regelsätze in der neuen Regelbasis bearbeitet werden. Möchte man einen neuen Regelsatz hinzufügen, darf nicht vergessen werden die Datei `TEC_RULES/rule_sets` anzupassen. Ein neuer Regelsatz, der auch aktiv sein soll, wird mit folgender Zeile hinzugefügt:

```
rule_set(neu,'neu.rls',active).
```

> **Hinweis**: Nach dem Hinzufügen eines Regelsatzes ist es ratsam, den Ereignisserver neu zu starten.

Ähnliches Vorgehen erfordert die Erstellung neuer Ereignisdefinitionsdateien. Möchte man die neue Datei `TEC_CLASSES/neu.baroc` hinzufügen, so ist ein Eintrag in der Datei `TEC_CLASSES/.load_classes` notwendig.

> **Hinweis**: Nach dem Hinzufügen einer neuen `*.baroc` Datei ist ein Neustart des Ereignisservers notwendig. Mit dem Befehl `wrb -lsrbclass <regelbasisname>` kann überprüft werden, ob die neuen Ereignisklassen vorhanden sind.

A.3 Arbeiten mit Regelsätzen

Wurde eine Regelsatzdatei modifiziert, ist eine Neukompilierung der Regelbasis notwendig. Dazu dient der Befehl

```
wrb -comprules [- trace] <regelbasisname>
```

Die `-trace` Option hilft den Ablauf der Regeln in einer Datei zu verfolgen. Dazu muss der Ereignisserver zunächst mit dem Befehl `wsetesvrcfg -t <datei>` die Tracefunktion aktivieren.

> **Hinweis**: Eine Deaktivierung der Tracefunktion erreicht man mit demselben Befehl ohne Angabe einer Datei. Nach dem Aktivieren oder Deaktivieren dieser Funktion muss der Ereignisserver neu gestartet werden.

Nach der fehlerfreien Kompilierung der Regelbasis ist entwender ein Neustart des Ereignisservers oder ein Neuladen der Regelbasis notwendig. Letzteres ist dabei auf Grund der Schnelligkeit vorzuziehen und mit diesem Befehl erledigt:

```
wrb -loadrb -use <regelbasisname>
```

Die Option `-use` ist dabei wichtig, da ohne sie die Änderungen nicht angewendet werden!

Der in dieser Arbeit entwickelte Regelsatz ist unter dem Namen *lrz01* zu finden. Im Dateisystem befinden sind die entsprechenden Dateien unter

```
/tivoli/server/hans_rb/lrz01/TEC_RULES/lrz.rls.
```

> **Hinweis**: Bei Fehlern in den Regelsätzen kann es beim Neustarten des Ereignisservers zu dieser Fehlermeldung kommen:
> `Error::FRWTE0017E Fri 12 Dec 2006 12:44:36 PM CET (17): system problem: 'ERROR: Process T/EC Rule exits with exit code 70: Problem occurred while loading Rules´`
> Hier hilft oft nur noch folgendes Vorgehen:

- Den Ereigniscache mit „`wtdbclear -e -f -l -t0´´` löschen (Achtung: Alle Events gehen verloren!)
- Mit „`wrb -loadrb default´´` die `default` Regelbasis laden
- Neustart des Ereignisservers
- Problem behaftete Regelbasis modifizieren
- Regelbasis kompilieren und mit der Option `-use` laden

A.4 Arbeiten mit Regeln

Zur vernünftigen Erstellung von Regeln / Regelsätzen kommt man um eine Debugausgabemöglichkeit nicht umhin. Eine aus den `default` Regelsätzen abgeschaute Vorgehensweise dazu ist, eine Regel zu schreiben, die auf ein „`TEC_START`“ Ereignis hin aktiv wird. In dieser Regel wird eine Logdatei initialisiert. Dies erledigen diese Prädikate:

```
rerecord(lrz_debug, 'yes'),
rerecord(lrz_logger, '/tivoli/server/work/lrz02/lrz.log'),
tl_init(lrz_tl, lrz_debug, lrz_logger),
```

Die Logdatei wird dabei immer beim Start des Ereignisservers neu erstellt.

> **Hinweis**: Oft ist es schwer bei zu vielen Debugausgaben die Einträge zu finden, nach denen man sucht. Dann darf aber nicht die Logdatei gelöscht werden, da sonst noch folgende Ausgaben ins Leere laufen. Stattdessen kann man mit
> `cat /dev/null > lrz.log` den Inhalt der Datei leeren.

Um Traceausgaben nutzen zu können, müssen Regeln, die „getraced´´ mitverfolgt werden sollen mit einer Direktive versehen werden: `directive: trace,`
Dies ist jedoch mit Vorsicht zu genießen. Die Tracedatei kann sehr schnell sehr groß (und unübersichtlich) werden.

Um mit einem „sauberen“ Eventcache beginnen zu können (zum Beispiel nach Modifizierung einer Regel), soll laut IBM der Befehl „`wtdbclear`“ helfen. Es ist jedoch zu beobachten, dass Prädikate, die auch zurückliegende Events suchen, diese auch nach der Anwendung dieses Befehls finden. Im implementierten Regelsatz (Anhang B) wurde deswegen eine Regel erstellt, die den Status aller vorhandenen Events auf *CLOSED* setzt.

A.5 Arbeiten mit Events

Da in dieser Arbeit nicht mir realen Events gearbeitet wurde, geschah das Senden von Ereignissen an den Server mit Hilfe des Befehls `wpostemsg`. Der folgende Aufruf zeigt ein Versenden eines Events mit Hilfe dieses Befehl:

```
wpostemsg service=wh_dns TEC_LRZ_SERVICE_AVAIL_NOK LOGFILE
```

Da die Ereignisquelle in den implementierten Regeln keine Rolle spielt, wurde hier immer `LOGFILE` verwendet. Weitere Eventattribute (zusätzlich zu `service`) können mit Leerzeichen getrennt angegeben werden. Weitere Optionen dieses Befehls (siehe [IBM a]) können verwendet werden, um zum Beispiel den Status des Events zu festzulegen.

Die für diese Arbeit entwickelten Ereignisklassen wurden in dieser Datei auf dem *soec* Rechner abgelegt:

```
/tivoli/server/hans_rb/lrz01/TEC_CLASSES/lrz.baroc
```

B Implementierter Regelsatz

Im Folgenden findet sich der gesamte Quelltext des implementierten Regelsatzes:

```
%-------------------------------------------------------------
% This is a startup rule used to initialize global parameters.
%-------------------------------------------------------------
rule: startup:
(
  event: _event
  of_class 'TEC_Start'
  where [
    hostname: _hostname
  ],

  % Set up global variables for the rule set.
  reception_action: setup:
  (
    % Debug flag
    rerecord(lrz_debug, 'yes'),

    % Debug file
    rerecord(lrz_logger, '/tivoli/server/hans_rb/lrz01/lrz.log'),

    % Latency
    rerecord(lrz_latency, 200),

    % Latency for duplicate events
    rerecord(lrz_dup_latency, 30),

    % Time to keep service events open, if no root cause found
    rerecord(lrz_timer, 30)

  ),

  % Initializes trace/log/debug files.
  reception_action: initialize:
  (
    tl_init(lrz_tl, lrz_debug, lrz_logger),
    commit_rule
  )
).



%-------------------------------------------------------------
% This is a shutdown rule used to finalize global parameters.
%-------------------------------------------------------------
rule: shutdown:
(
  event: _event
  of_class 'TEC_Stop'
  where [],
```

```
  % Closes trace/log/debug files.
  reception_action: finalize:
  (
    tl_stop(lrz_tl),
    commit_rule
  )
).



%-------------------------------------------------------------------
% Timer rule for expiration of service events
%-------------------------------------------------------------------
timer_rule: timer_expiration:
(
  event: _event of_class _class
  where [
    event_handle: _event_handle,
    date_reception: _date
  ],

  timer_info: equals 'ServiceEvent_expiration',

  action:
  (
    tl_fmt(lrz_tl, '\nServiceEvent %s expired.\n', [_class]),

    % Search for cause event
    (
      first_instance(
        event: _pc_event
        of_class 'TEC_LRZ_PROBABLE_CAUSE_EVENT'
        where [
          effect_event_handle: equals _event_handle,
          effect_event_date: equals _date,
          cause_class: _cause_class
        ]
      ),
      tl_fmt(lrz_tl, 'Probable cause event %s was found.\n',
       _cause_class)
    ;
      % No causes found for this service event
      % -> forward to administrator / case-based reasoner
      tl_str(lrz_tl, 'No probable cause event found.\n')
    ),

    change_event_status(_event, 'CLOSED'),
    tl_fmt(lrz_tl, '%s event closed\n', _class)
  )
).



%-------------------------------------------------------------------
% Rule for closing all events
%-------------------------------------------------------------------
rule: close_all:
(
  event: _event
  of_class 'TEC_LRZ_CLOSE_ALL'
  where [],
```

```
  action:
  (
    all_instances(
      event: _ev
      of_class _ev_class %within ['TEC_LRZ_CLOSE_ALL']
      where [
        status: outside ['CLOSED']
      ]
    ),
    set_event_status(_ev, 'CLOSED'),
    tl_str(lrz_tl, '*')
  ),

  action:
  (
    tl_str(lrz_tl, 'all CLOSED\n\n\n'),
    commit_rule
  )
).



%--------------------------------------------------------------------
% Rule to handle duplicate service events
%--------------------------------------------------------------------
rule: duplicate_services:
(
  event: _event
  of_class _class within [
         'TEC_LRZ_SERVICE_AVAIL_NOK',
         'TEC_LRZ_SERVICE_AVAIL_OK',
         'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
         'TEC_LRZ_SERVICEFUNC_AVAIL_OK',
         'TEC_LRZ_SERVICEFUNC_TIME_NOK',
         'TEC_LRZ_SERVICEFUNC_TIME_OK'
        ]
  where [
    status: outside ['CLOSED'],
    severity: _severity,
    server_handle: _srv_handle,
    date_reception: _date,
    event_handle: _ev_handle,
    hostname: _hostname,
    service: _service,
    service_access_point: _service_access_point
  ],

  action: start:
  (
    tl_str(lrz_tl, '\n<<Entering wh duplicate rule>>\n')
  ),

  action: check_duplicate:
  (
    %get 'global' variables
    recorded(lrz_dup_latency, _dup_latency),
    tl_fmt(lrz_tl, '\t...processing %s event\n', [_class]),

    first_duplicate(_event, event: _dup_event
      where [
```

```
        status: outside ['CLOSED'],
        severity: _dup_severity,
        service: equals _service,
        service_access_point: equals _service_access_point
      ],
      _event -10 -0
    ),
    change_event_severity(_dup_event, 'CRITICAL'),
    tl_str(lrz_tl, '\t...severity of orig. event changed
    to CRITICAL\n'),

    % Instead of dropping we prefer closing the event
    change_event_status(_event, 'CLOSED'),

    % OR: increment repeat_count attribute and drop
    %add_to_repeat_count(_dup_ev, 1),
    %drop_received_event,

    tl_fmt(lrz_tl, '\t...event %s identified as duplicate
    and closed\n', [_class]),

    % Prevent analysis of this event in the current rule
    commit_set
  ),

  action: end:
  (
    tl_str(lrz_tl, '<<Exiting wh duplicate rule>>\n')
  )
).



%--------------------------------------------------------------------
% Correlation rule
%--------------------------------------------------------------------
rule: lrz_correlate:
(
  %enable tracing for debug reasons
  %directive: trace,
  event: _event
  of_class _class within [
      'TEC_LRZ_SERVICE_AVAIL_NOK',
      'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
      'TEC_LRZ_SERVICEFUNC_TIME_NOK'
      ]
  where [
    status: outside ['CLOSED'],
    severity: _severity,
    event_handle: _event_handle,
    date_reception: _date,
    %valid_to: _valid_to smaller NOW(), ***TODO***
    event_handle: _ev_handle,
    hostname: _hostname,
    service: _service,
    service_access_point: _service_access_point,
    linked_cause_handles: _linked_cause_handles,
    linked_cause_dates: _linked_cause_dates
  ],

  action: setup_correlation:
```

```
(
  tl_str(lrz_tl, '\n<<Entering correlation rule>>\n'),

  %get variables
  recorded(lrz_latency, _latency),

  length(_linked_cause_handles, _l),
  % ugly workaround to get integer in debug
  sprintf(_tmp, '%u', _l),
  tl_fmt(lrz_tl, '\t...processing %s event on service %s with %s
  linked events\n', [_class, _service, _tmp]),

  %Resetting correlation vars
  reset_global_grp('lrz_correlation', [])
),

action: correlate_resources:
(
  tl_str(lrz_tl, '\t<<Entering correlation of resources>>\n'),

  % Based on received effect event class, define the possible cause classes
  (
    member(_class, ['TEC_LRZ_SERVICE_AVAIL_NOK','TEC_LRZ_SERVICEFUNC_AVAIL_NOK']),
    _cause_classes = ['TEC_LRZ_RESOURCE_AVAIL_NOK']
  ;
    member(_class, ['TEC_LRZ_SERVICEFUNC_TIME_NOK']),
    _cause_classes = ['TEC_LRZ_RESOURCE_CPU_UTIL_NOK']
  ),

  tl_str(lrz_tl, '\t...cause classes identified\n'),
  (
    _service == 'wh_getstat',
    _dependent_resources = ['slb1','swk4-2wr','nx110','nx112','nx114','nx116']
  ;
    _service == 'wh_getdyn',
    _dependent_resources = ['swm2-2wr','nas']
  ;
    _service == 'wh_dns',
    _dependent_resources = ['dns1', 'dns2']
  ;
    _service == 'em_mailout',
    _dependent_resources = ['lxmhs01', 'lxmhs02']
  ;
    _service == 'em_dns',
    _dependent_resources = ['lxmhs19', 'lxmhs20']
  ),

  length(_dependent_resources, _dependent_length),
  sprintf(_tmp, '%u', _dependent_length),
  tl_fmt(lrz_tl, '\t...%s dependent resources for service %s identified\n',
  [_tmp, _service]),

  % Set global correlation var
  set_global_var('lrz_correlation', 'dependent_resources', _dependent_resources),

  % Search for cause events
  not empty_list(_dependent_resources),
  (
    tl_str(lrz_tl, '\t...searching for cause events...\n'),
    all_instances(
      event: _cause_event
```

```
      of_class within _cause_classes
      where [
        status: _cause_status outside ['CLOSED'],
        resource: _cause_resource within _dependent_resources,
        date_reception: _cause_date outside _linked_cause_dates,
        event_handle: _cause_event_handle outside _linked_cause_handles
      ],
      %_event - 120 - 120  %Commented out to find earlier events
    ),

    bo_get_class_of(_cause_event, _cause_class),
    tl_fmt(lrz_tl, '\t...found %s event on resource %s\n',
    [_cause_class, _cause_resource]),

    % Remove this _cause_resource from list of dependents
    _default = [],
    get_global_var('lrz_correlation', 'dependent_resources',
     _new_dependent_resources, _default),
    delete(_new_dependent_resources, _cause_resource,
    _restof_dependent_resources),
    set_global_var('lrz_correlation', 'dependent_resources',
    _restof_dependent_resources),

    % Generate probable cause event
    generate_event('TEC_LRZ_PROBABLE_CAUSE_EVENT',
      [
        effect_service=_service,
        effect_event_handle=_event_handle,
        effect_event_date=_date,
        effect_class=_class,
        cause_event_handle=_cause_event_handle,
        cause_event_date=_cause_date,
        cause_class=_cause_class
      ]
    ),
    tl_str(lrz_tl, '\t...TEC_LRZ_PROBABLE_CAUSE_EVENT generated\n')
  )
),

% reception_action! Will not be called in redo analysis
reception_action: check_resource_restlist:
(
  % Get list of dependent resources
  _default = [],
  get_global_var('lrz_correlation', 'dependent_resources',
  _probe_resources, _default),
  length(_probe_resources, _l),
  sprintf(_tmp, '%u', _l),
  (
    _l > 0,
    generate_event('TEC_LRZ_ACTIVE_PROBING_EVENT',
      [
        sender_handle=_event_handle,
        sender_date=_date,
        resources=_probe_resources
      ]
    ),
    tl_fmt(lrz_tl, '\t...active probing event generated for %s resource(s)\n', _tmp),
    change_event_status(_event_, 'ACK')
  ;
    _l == 0,
```

```
    tl_str(lrz_tl, '\t...no resources found for active probing\n'),
    change_event_status(_event, 'CLOSED')
  )
),

action: exit_resources:
(
  tl_str(lrz_tl, '\t<<Exiting correlation of resources>>\n')
),

action: correlate_services:
(
  tl_str(lrz_tl, '\t<<Entering correlation of services>>\n'),

  % Based on received effect event class, define the possible cause classes
  (
    member(_class, ['TEC_LRZ_SERVICE_AVAIL_NOK']),
    _cause_classes = ['TEC_LRZ_SERVICE_AVAIL_NOK',
    'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
    'TEC_LRZ_SERVICEFUNC_TIME_NOK']
  ;
    member(_class, ['TEC_LRZ_SERVICEFUNC_AVAIL_NOK']),
    _cause_classes = ['TEC_LRZ_SERVICE_TIME_NOK']
  ),

  (
    _service == 'wh_getstat',
    _dependent_services = ['wh_dns']
  ;
    _service == 'wh_getdyn',
    _dependent_services = ['wh_dns','wh_db']
  ;
    _service == 'em_mailout',
    _dependent_services = ['em_dns']
  ;
    _service == 'em_mail',
    _dependent_services = ['em_mailin','em_mailout']
  ),

  length(_dependent_services, _dependent_length),
  sprintf(_tmp, '%u', _dependent_length),
  tl_fmt(lrz_tl, '\t...%s dependent subservices for %s identified\n',
  [_tmp, _service]),

  % Set global correlation var
  set_global_var('lrz_correlation', 'dependent_services',
  _dependent_services),

  % Search for cause events
  not empty_list(_dependent_services),
  (
    tl_str(lrz_tl, '\t...searching for cause events...\n'),
    all_instances(
      event: _cause_event
      of_class within _cause_classes
      where [
        status: outside ['CLOSED'],
        service: _cause_service within _dependent_services,
        event_handle: _cause_event_handle outside _linked_cause_handles,
        date_reception: _cause_date outside _linked_cause_dates
      ],
```

```
      _event -120 -120
    ),

    bo_get_class_of(_cause_event, _cause_class),
    tl_fmt(lrz_tl, '\t...found %s event on service %s\n',
    [_cause_class, _cause_service]),

    % Remove this _cause_service from list of dependents
    _default = [],
    get_global_var('lrz_correlation', 'dependent_services',
    _new_dependent_services, _default),
    delete(_new_dependent_services, _cause_service,
    _restof_dependent_services),
    set_global_var('lrz_correlation', 'dependent_services',
    _restof_dependent_services),

    % Generate probable cause event
    generate_event('TEC_LRZ_PROBABLE_CAUSE_EVENT',
      [
        effect_service=_service,
        effect_event_handle=_event_handle,
        effect_event_date=_date,
        effect_class=_class,
        cause_event_handle=_cause_event_handle,
        cause_event_date=_cause_date,
        cause_class=_cause_class
      ]
    ),
    tl_str(lrz_tl, '\t...TEC_LRZ_PROBABLE_CAUSE_EVENT generated\n')
  )
),

reception_action: check_service_restlist:
(
  % Get list of dependent services
  _default = [],
  get_global_var('lrz_correlation', 'dependent_services',
  _probe_services, _default),
  length(_probe_services, _l),
  sprintf(_tmp, '%u', _l),
  (
    _l > 0,
    generate_event('TEC_LRZ_ACTIVE_PROBING_EVENT',
      [
        sender_handle=_event_handle,
        sender_date=_date,
        services=_probe_services
      ]
    ),
    tl_fmt(lrz_tl, '\t...TEC_LRZ_ACTIVE_PROBING_EVENT generated
    for %s service(s)\n', _tmp),
    change_event_status(_event, 'ACK')
  ;
    _l == 0,
    tl_str(lrz_tl, '\t...no services found for active probing\n'),
    change_event_status(_event, 'CLOSED')
  )
),

action: exit_services:
(
```

```
    tl_str(lrz_tl, '\t<<Exiting correlation of services>>\n'),
    change_event_status(_event, 'ACK')
  ),

  reception_action: do_redo:
  (
    tl_str(lrz_tl, '\t<<Entering redo request>>\n'),

    % Request redoanalysis of events previous to this
    tl_fmt(lrz_tl, '\t...searching for AP event for service %s...\n', _service),
    all_instances(
      event: _ap_event
      of_class 'TEC_LRZ_ACTIVE_PROBING_SERVICE'
      where [
        status: outside ['CLOSED'],
        service: equals _service,
        sender_handle: _sender_handle,
        sender_date: _sender_date
      ]
    ),
    tl_fmt(lrz_tl, '\t...found AP event for %s\n', [_service]),
    change_event_status(_ap_event, 'CLOSED'),
    tl_str(lrz_tl, '\t...searching for service event\n'),
    first_instance(
      event: _se_event
      of_class within ['TEC_LRZ_SERVICE_AVAIL_NOK','TEC_LRZ_SERVICEFUNC_AVAIL_NOK']
      where [
        status: within ['ACK'],
        date_reception: equals _sender_date,
        event_handle: equals _sender_handle
      ]
    ),
    tl_str(lrz_tl, '\t...request redo analysis of previous service event\n'),
    redo_analysis(_se_event)
  ),

  reception_action: start_timer:
  (
    recorded(lrz_timer, _timer),
    set_timer(_event, _timer, 'ServiceEvent_expiration')
  ),

  action: exit_rule:
  (
    tl_str(lrz_tl, '<<Exiting correlation rule>>\n')
  )
).



%-----------------------------------------------------------------
% Rule to link events
%-----------------------------------------------------------------
rule: probable_cause:
(
  %enable tracing for debug reasons
  directive: trace,
  event: _event
  of_class _class within ['TEC_LRZ_PROBABLE_CAUSE_EVENT']
  where [
    status: outside ['CLOSED'],
```

```
    effect_service: _effect_service,
    effect_event_handle: _effect_event_handle,
    effect_event_date: _effect_event_date,
    effect_class: _effect_class,
    cause_event_handle: _cause_event_handle,
    cause_event_date: _cause_event_date,
    cause_class: _cause_class
  ],

  action: setup:
  (
    tl_str(lrz_tl, '\n<<Entering probable cause rule>>\n'),

    % Search for effect event
    tl_fmt(lrz_tl, '\t...searching for effect event %s...\n', _effect_class),
    first_instance(
      event: _effect_event
      of_class _effect_class within [
        'TEC_LRZ_SERVICE_AVAIL_NOK',
        'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
        'TEC_LRZ_SERVICEFUNC_TIME_NOK'
      ]
      where   [
        event_handle: equals _effect_event_handle,
        date_reception: equals _effect_event_date
      ]
    ),
    tl_fmt(lrz_tl, '\t...found %s event\n', [_effect_class]),

    % Need to get attributes, because of compiler warning message
    bo_get_slotval(_effect_event, 'linked_cause_dates',
    _linked_cause_dates),
    bo_get_slotval(_effect_event, 'linked_cause_handles',
    _linked_cause_handles),

    % Add cause handle and date to effect event to link them
    (
      not empty_list(_linked_cause_handles),
      append([_cause_event_handle], _linked_cause_handles, _tmp_h),
      append([_cause_event_date], _linked_cause_dates, _tmp_d)
    ;
      empty_list(_linked_cause_handles),
      _tmp_h = [_cause_event_handle],
      _tmp_d = [_cause_event_date]
    ),

    % Update effect event
    bo_set_slotval(_effect_event, 'linked_cause_handles', _tmp_h),
    bo_set_slotval(_effect_event, 'linked_cause_dates', _tmp_d),
    tl_str(lrz_tl, '\t...Events correlated!\n')
  ),

  action: end:
  (
    change_event_status(_event, 'CLOSED'),
    tl_str(lrz_tl, '<<Exiting probable cause rule>>\n')
  )
).
```

```
%--------------------------------------------------------------
% Rule to generate specific AP events
%--------------------------------------------------------------
rule: active_probing:
(
  %enable tracing for debug reasons
  %directive: trace,
  event: _event
  of_class _class within ['TEC_LRZ_ACTIVE_PROBING_EVENT']
  where [
    status: outside ['CLOSED'],
    services: _services,
    resources: _resources,
    sender_handle: _sender_handle,
    sender_date: _sender_date
  ],

  action:
  (
    tl_str(lrz_tl, '\n<<Entering active probing rule>>\n'),
    length(_services, _ls),
    length(_resources, _rs),
    (
      _ls > 0,
      rremove(_service, _services, _new_services),
      generate_event('TEC_LRZ_ACTIVE_PROBING_SERVICE',
        [
          service=_service,
          sender_handle=_sender_handle,
          sender_date=_sender_date
        ]
      ),
      tl_fmt(lrz_tl, '\t...TEC_LRZ_ACTIVE_PROBING_SERVICE event for %s
       generated\n', _service),
      bo_set_slotval(_event, 'service', _new_services),
      tl_str(lrz_tl, 'nach slotval__S'),
      redo_analysis(_event)
    ;
      _rs > 0,
      rremove(_resource, _resources, _new_resources),
      generate_event('TEC_LRZ_ACTIVE_PROBING_RESOURCE',
        [
          sender_date=_sender_date,
          sender_handle=_sender_handle,
          resource=_resource
        ]
      ),
      tl_fmt(lrz_tl, '\t...TEC_LRZ_ACTIVE_PROBING_RESOURCE event for %s
      generated\n', _resource),
      bo_set_slotval(_event, 'resource', _new_resources),
      redo_analysis(_event)
    )
  ),

  action: exit_rule:
  (
    change_event_status(_event, 'CLOSED'),
    drop_received_event,
    tl_str(lrz_tl, '<<Exiting active probing rule>>\n')
  )
).
```

```
%--------------------------------------------------------------
% Rule to request redo analysis of SE
%--------------------------------------------------------------
rule: resource_handler:
(
  %enable tracing for debug reasons
  %directive: trace,
  event: _event
  of_class _class within [
    'TEC_LRZ_RESOURCE_AVAIL_NOK',
    'TEC_LRZ_RESOURCE_CPU_UTIL_NOK'
  ]
  where [
    status: outside ['CLOSED'],
    resource: _resource
  ],

  reception_action:
  (
    tl_str(lrz_tl, '\n<<Entering resource_handler rule>>\n'),

    % Request redoanalysis of events previous to this
    tl_fmt(lrz_tl, '\t...searching for AP event for resource %s...\n',
    _resource),
    all_instances(
      event: _ap_event
      of_class 'TEC_LRZ_ACTIVE_PROBING_RESOURCE'
      where [
        status: outside ['CLOSED'],
        resource: equals _resource,
        sender_handle: _sender_handle,
        sender_date: _sender_date
      ],
      _event -120 -0
    ),

    sprintf(_tmp, '%u', _sender_date),
    tl_fmt(lrz_tl, '\t...found AP event for %s with date %s\n', [_resource, _tmp]),
    change_event_status(_ap_event, 'CLOSED'),
    tl_str(lrz_tl, '\t...searching for service event\n'),
    first_instance(
      event: _se_event
      of_class within [
        'TEC_LRZ_SERVICE_AVAIL_NOK',
        'TEC_LRZ_SERVICEFUNC_AVAIL_NOK',
        'TEC_LRZ_SERVICEFUNC_TIME_NOK'
      ]
      where [
        status: within ['ACK'],
        date_reception: equals _sender_date,
        event_handle: equals _sender_handle
      ]
    ),
    tl_str(lrz_tl, '\t...request redo analysis of previous service event\n'),
    redo_analysis(_se_event)
  ),

  reception_action:
```

```
  (
    tl_str(lrz_tl, '<<Exiting resource_handler rule>>\n')
  )
).
```

C Definition der Eventklassen

Nachfolgend werden die eigens implementierten Eventklassen aufgelistet.

```
################################################
# Event without meaning to close all open events
################################################
TEC_CLASS:
  TEC_LRZ_CLOSE_ALL ISA EVENT;
END



################################################
# Informational events
################################################

# Probable cause for an effect service event found
TEC_CLASS:
  TEC_LRZ_PROBABLE_CAUSE_EVENT ISA EVENT
    DEFINES {
      effect_service: STRING;
      effect_event_handle: INTEGER;
      effect_event_date: INT32;
      effect_class: STRING;
      cause_event_handle: INTEGER;
      cause_event_date: INT32;
      cause_class: STRING;
    };
END

# Common active probing event
TEC_CLASS:
  TEC_LRZ_ACTIVE_PROBING_EVENT ISA EVENT
    DEFINES {
      sender_date: INT32;
      sender_handle: INTEGER;
      services: LIST_OF STRING,         default = [];
      resources: LIST_OF STRING,        default = [];
  };
END

# AP event for resource
TEC_CLASS:
  TEC_LRZ_ACTIVE_PROBING_RESOURCE ISA EVENT
    DEFINES {
     sender_date: INT32;
     sender_handle: INTEGER;
     resource: STRING;
  };
END

# AP event for service
TEC_CLASS:
```

```
  TEC_LRZ_ACTIVE_PROBING_SERVICE ISA EVENT
    DEFINES {
     sender_date: INT32;
     sender_handle: INTEGER;
     service: STRING;
  };
END


################################################
# Base event classes
################################################

# Resource Events
TEC_CLASS:
 TEC_LRZ_RESOURCE_QOD_EVENT ISA EVENT
  DEFINES {
   source: STRING, default = "LRZ resource monitoring";
   severity: SEVERITY,     default = WARNING;
   status: STATUS,         default = OPEN;
   date_reception: INT32;
   resource: STRING;
  };
END

# Service Events
TEC_CLASS:
 TEC_LRZ_SERVICE_QOS_EVENT ISA EVENT
  DEFINES {
   source: STRING, default = "LRZ service monitoring";
   severity: SEVERITY,     default = WARNING;
   status: STATUS,         default = OPEN;
   date_reception: INT32;
   date_referring: INT32;
   service: STRING;
   service_func: STRING;
   service_access_point: STRING;
   valid_to: INT32;
   description: STRING;
   linked_cause_handles: LIST_OF INTEGER,       default = [];
   linked_cause_dates: LIST_OF INT32,           default = [];
  };
END

TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_QOS_EVENT ISA TEC_LRZ_SERVICE_QOS_EVENT;
END



################################################
# Event classes to instantiate
################################################

# Service availability
TEC_CLASS:
        TEC_LRZ_SERVICE_AVAIL_NOK ISA TEC_LRZ_SERVICE_QOS_EVENT;
END

TEC_CLASS:
 TEC_LRZ_SERVICE_AVAIL_OK ISA TEC_LRZ_SERVICE_QOS_EVENT
```

```
  DEFINES {
   severity: SEVERITY,          default = HARMLESS;
  };
END


# Service functionality availability
TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_AVAIL_NOK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT;
END

TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_AVAIL_OK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
  DEFINES {
   severity: SEVERITY,          default = HARMLESS;
  };
END


# Service functionality time condition
TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_TIME_NOK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
  DEFINES {
   time: INT32;
  };
END

TEC_CLASS:
 TEC_LRZ_SERVICEFUNC_TIME_OK ISA TEC_LRZ_SERVICEFUNC_QOS_EVENT
  DEFINES {
   time: INT32;
   severity: SEVERITY,          default = HARMLESS;
  };
END


# Resource availability
TEC_CLASS:
        TEC_LRZ_RESOURCE_AVAIL_NOK ISA TEC_LRZ_RESOURCE_QOD_EVENT;
END

TEC_CLASS:
 TEC_LRZ_RESOURCE_AVAIL_OK ISA TEC_LRZ_RESOURCE_QOD_EVENT
DEFINES {
   severity: SEVERITY,          default = HARMLESS;
  };
END


# Resource CPU utility
TEC_CLASS:
 TEC_LRZ_RESOURCE_CPU_UTIL_NOK ISA TEC_LRZ_RESOURCE_QOD_EVENT
  DEFINES {
   utilization: REAL;
  };
END

TEC_CLASS:
 TEC_LRZ_RESOURCE_CPU_UTIL_OK ISA TEC_LRZ_RESOURCE_QOD_EVENT
  DEFINES {
   utilization: REAL;
```

```
   severity: SEVERITY,          default = HARMLESS;
  };
END
```

Literaturverzeichnis

[AJG 93] AVELINO J. GONZALEZ, DOUGLAS D. DANKEL: *The Engineering of Knowledge-Based Systems, Theory and Practice*. Prentice Hall, 1993.

[Apac06] FOUNDATION, THE APACHE SOFTWARE: *Apache HTTP Server Project*, 2006, http://httpd.apache.org/ .

[Bitk 05] BITKOM: *Business Process Outsourcing Leitfaden - BPO als Chance für den Standort Deutschland*, September 2005, http://www.bitkom.org/files/documents/BITKOM_Leitfaden_BPO_Stand_20.09.05.pdf .

[Broy 98] BROY, MANFRED: *Informatik - Eine grundlegende Einführung Band 1*. Springer-Verlag, 1998.

[EG 95] ERICH GAMMA, RICHARD HELM, RALPH E. JOHNSON: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[Forg 82] FORGY, CHARLES: *A fast algorithm for the many pattern/many object pattern match problem*, 1982.

[Gars 01] GARSCHHAMMER, M., HAUCK R. HEGERING H.-G. KEMPTER B. LANGER M. NERB M. RADISIC I. ROELLE H. SCHMIDT H.: *Towards generic Service Management Concepts - A Service Model Based Approach*. Munich, Germany, 2001. MNM Team, LRZ.

[HA 05] HANEMANN A., SAILER M., SCHMITZ D.: *Towards a Framework for IT Service Fault Management*. Munich, Germany, 2005. MNM Team, LRZ.

[Hane 06] HANEMANN, A.: *A Hybrid Rule-Based/Case-Based Reasoning Approach for Service Fault Diagnosis*. Munich, Germany, 2006. MNM Team, LRZ.

[Hane 07] HANEMANN, ANDREAS: *Automated IT Service Fault Diagnosis Based on Event Correlation Techniques*. Doktorarbeit, Ludwig-Maximilians-Universität München, 2007. Erscheint in 2007.

[IBM a] IBM: *IBM TEC Command and Task Reference*, http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.itecctref.doc/ecormst.pdf .

[IBM b] IBM: *IBM Tivoli Enterprise Console - Benutzerhandbuch*, http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.itecuser.doc/ecoumst.pdf .

[IBM c] IBM: *IBM Tivoli Enterprise Console - Rule Developers Guide*, http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.itecruledev.doc/ecodmst.pdf .

[IBM d] IBM: *IBM Tivoli Enterprise Console - RuleSet Reference*, http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.itecrsref.doc/ecosmst.pdf .

[JG 04] JAKOBSON G., BUFORD J., LEWIS L.: *Towards an Architecture for Reasoning about Complex Event-Based Dynamic Situations*. 2004, http://serl.cs.colorado.edu/ carzanig/debs04/debs04jakobson.pdf .

[LRZ] LRZ: *Jahresbericht 2005 des Leibniz-Rechenzentrums*, http://www.lrz-muenchen.de/wir/berichte/ .

[LRZ 06a] *Leibniz-Rechenzentrum*, Dezember 2006, http://www.lrz-muenchen.de/wir/intro/de/ .

[LRZ 06b] *Webhosting: Virtueller WWW-Server am LRZ*, Dezember 2006, http://www.lrz-muenchen.de/services/netzdienste/www/uns_virt/ .

[LRZ 06c] *Wir über uns*, Dezember 2006, http://www.lrz-muenchen.de/wir/ .

[Marc 06] MARCU, PATRICIA: *Abhängigkeitsmodellierung für das IT-Dienstmanagement*. Diplomarbeit, Ludwig-Maximilians-Universität München, 2006.

[MB 03] MARK BRODIE, IRINA RISH, SHENG MA-NATALIA ODINTSOVA ALINA BEYGELZIMER: *Active Probing Strategies for Problem Diagnosis in Distributed Systems*. IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, 2003. IJCAI.

[MN 06] M-NET: *M-net Telekommunikations GmbH Unternehmensprofil*, Oktober 2006, http://www.m-net.de/fileadmin/downloads/ueber_mnet/uprofil_aktuell.pdf .

[oGC 00] GOVERNMENT COMMERCE, OGC OFFICE OF: *Service Support*. Office of Government Commerce, 2000.

[Pack 06a] PACKARD, HEWLETT: *Event Correlation Services*, Dezember 2006, http://h20229.www2.hp.com/products/ecs/index.html .

[Pack 06b] PACKARD, HEWLETT: *Management Software: HP OpenView*, Dezember 2006, http://openview.hp.com/ .

[Pilo 04] PILONE, DAN: *UML kurz & gut*. O'Reilly Verlag, 2004.

[PPB 91] PETER P. BOTHNER, WOLF-MICHAEL KÄHLER: *Programmieren in PROLOG - Eine umfassende und praxisgerechte Einführung*. Friedr. Vieweg & Sohn, 1991.

[RGD 98] RODOSEK GABI DREO, KAISER THOMAS: *Platform Independent Event Correlation Tool for Network Management*. Munich, Germany, 1998. LRZ.

[Stor 06] STORZ, MICHAEL: *Policy für die zentralen Mailrelays des LRZ*, März 2006, http://www.lrz-muenchen.de/services/netzdienste/email/policy/policy.pdf .

[TB 04] TONY BHE, PETER GLASMACHER, JACQUELINE MECKWOOD-GUILHERME PAREIRA MICHAEL WALLACE: *IBM Redbook - Event Management Best Practices*, Juni 2004, http://www.redbooks.ibm.com/abstracts/sg246094.html .

[Vaar 02a] VAARANDI, RISTO: *Intelligent Assistant: User-Guided Fault Localization*. Department of Computer Engineering, Tallinn Technical University, Estonia, 2002. IEEE/IFIP Network.

[Vaar 02b] VAARANDI, RISTO: *SEC - a Lightweight Event Correlation Tool*. 2002, http://www.estpak.ee/ risto/publications/sec-ipom02-web.pdf .

[vB 02] BON, JAN VAN: *IT Service Management, eine Einführung*. Van Haren Publishing, ISBN 90-806713-5-5, 2002.