Masterarbeit

Prozessmonitoring in serviceorientierten Architekturen

Bakk. Peter Kröpfl

angestrebter akademischer Grad

Diplom-Ingenieur

Wien, 2010

Studienkennzahl lt. Studienblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik UG2002
Betreuerin / Betreuer:	o. Univ. Prof. Dr. Dimitris Karagiannis

Autor
Bakk. Peter Kröpfl
Matr.Nr.: 0100700
Kennzahl: 066 926

Betreuer
Dr. Harald Kühn

Begutachter
o. Univ.-Prof. Dr.
Dimitris Karagiannis

10. Februar 2010

Danksagung

Zu allererst möchte ich mich bei meiner Familie bedanken. Ohne sie hätte ich dieses Studium und damit auch diese Arbeit nicht erfolgreich abschließen können. Ihre materielle, vor allem aber auch ihre persönliche Unterstützung, ihr Vertrauen in mich und ihr Zuspruch in schwierigeren Tagen haben es mir ermöglicht dieses Studium zu Ende zu führen. Dafür mein ganz besonderer, zutiefst empfundener Dank.

Des weiteren möchte ich mich bei Dr. Harald Kühn für seine tatkräftige Unterstützung und seinen wertvollen Input bedanken. Ebenso gilt mein Dank Prof. Dr. Dimitris Karagiannis, dem Departement für Knowledge Engineering und der Universität Wien, sowie der TU Wien für die Ausbildung und Betreuung auf dem Weg zu dieser Abschlussarbeit.

Ich bedanke mich außerdem bei der Firma Mercatis, den Geschäftsführern Andreas Grüner, Armin Haaf und Wolfgang Traxler, sowie allen meinen geschätzten Kollegen, insbesondere Dr. Utz Westermann, die mich unterstützt haben parallel zur beruflichen Laufbahn dieses Studium und diese Arbeit erfolgreich zu beenden.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Wien, am 10. Februar 2010

.....

(Peter Kröpfl)

Abstract

In this thesis a novel approach to process monitoring will be presented. This approach leverages the concept of complex event processing (CEP) for near real time monitoring of processes and process instances. As a major difference it does not rely on the integration of processes in a workflow management system (WfMS) to tackle the typical problems such as distribution of services, modelling of process flows or reuse of services in different business contexts.

CEP allows to overcome those problems without direct integration of services in an execution evironment, such as a workflow managemet system. Monitoring will be enabled by rules applied to the services' output that will be monitored constantly.

The foundation for this approach is a detailed analysis of the major problems and challenges in process monitoring. Based on this analysis and the analysis of related monitoring tools the requirements for process monitoring in distributed systems have been developed.

The concept of complex event processing as defined by David Luckham meets many of those requirements. To determine whether this theoretical concept can stand real life challenges a prototype has been developed and tested in several scearios.

The results are very promising. Process monitoring in service oriented architectures can be done using complex event processing technology. All of the defined monitoring scenarios could be implemented with the prototype.

As expected the monitoring of process instances especially in connection with composite services was the most challenging scenario and the complexity of the configuration increased substantially. However with more work on the the prototype the complexity surely can be reduced and maybe even a visual configuration tool can be developed. This area should be part of further research.

Zusammenfassung

In dieser Arbeit wird ein neuartiger Ansatz zum Monitoring von Prozessen präsentiert. Er basiert auf der Idee des Complex Event Processing (CEP) und soll nahezu Echtzeitmonitoring von Prozessen und Prozessinstanzen ermöglichen. Der große Unterschied zu den bisherigen Ansätzen ist, dass es nicht nötig ist Prozesse in eine Ablaufumgebung, wie ein Workflow-Managementsystem (WfMS) zu integrieren, um sie durchgängig überwachen zu können. Durch den Wegfall von aufwändigen WfMS-Projekten und starren Prozessen gewinnen Unternehmen und deren IT-Architekturen deutlich an Flexibiltät.

Der Verzicht auf eine integrierte, steuernde Workflow-Engine und die darin fest verdrahteten Prozesse bringt aber auch viele neue Herausforderungen mit sich. Vor allem die Verteilung der Services und die Verwendung in unterschiedlichen Kontexten, sowie die Abbildung der nurmehr implizit vorhandenen Prozesse sind die Hauptprobleme denen sich dieser Monitoringansatz stellen muss.

Die Anwendbarkeit des oben vorgestellten Ansatzes wird im Rahmen dieser Arbeit untersucht. Basis ist eine fundierte Analyse der Probleme und Herausforderungen im Prozessmonitoring. Gemeinsam mit einer Analyse verwandter Monitoring-Tools und verbreiteter Monitoring-Ansätzen bildet sie den Grundstock für die Definition der Anforderungen an ein Prozessmonitoring in verteilten Architekturen.

Das Konzept des Complex Event Processing nach David Luckham verspricht viele der definierten Anforderungen abzudecken. Mit CEP als Basis und den vielversprechendsten Architekturideen aus den verschiedenen existierenden Ansätzen wird eine Architektur definiert, die Monitoring von Prozessen ermöglicht.

Zum Test der Tauglichkeit des neu entwickelten Ansatzes wird ein Prototyp entwickelt und in verschiedenen Szenarios getestet. Diese Szenarios basieren auf einem Energiehandelsprozess und werden entsprechend der anfangs definierten Anforderungen modelliert.

Die Ergebnisse sind sehr vielversprechend. Der entwickelte Ansatz kann genutzt werden um in serviceorientierten Architekturen Prozessmonitoring zu betreiben. Alle aufgestellten Monitoring-Szenarios konnten mit Hilfe des Prototypen

umgesetzt werden.

Wie erwartet war die größte Herausforderung das Monitoring von Prozessinstanzen, speziell in Verbindung mit zusammengesetzten Services. Hier stieg die Komplexität der Regeln sehr stark an.

Dieser Teil ist auch ein wichtiges Gebiet für weiterführende Forschungen. Eine Vereinfachung der Konfiguration und der Regeln, würde die Arbeit deutlich vereinfachen.

Inhaltsverzeichnis

1 Einführung **1**

2 Grundlagen **3**
2.1 Prozess und Geschäftsprozess . . . 3
2.1.1 Definitionen in der Literatur . . . 4
2.1.2 Definition für diese Arbeit . . . 5
2.2 Monitoring . . . 5
2.2.1 Definitionen . . . 5
2.2.2 Abgrenzung zu Measurement . . . 6
2.2.3 Ebenen des Monitorings . . . 6
2.2.4 Definition Prozessmonitoring . . . 8
2.3 Service . . . 8
2.3.1 Business Service . . . 9
2.3.2 Technischer Service . . . 9
2.3.3 Serviceorientierte Architektur . . . 10
2.4 Event . . . 11
2.4.1 Event . . . 11
2.4.2 Complex Event . . . 13
2.4.3 Complex Event Processing . . . 14

3 Motivation für Monitoring **15**
3.1 Fehlersuche . . . 15
3.2 Prozessverbesserung . . . 16
3.3 Operations . . . 17
3.4 Entscheidungsunterstützung . . . 19
3.5 Compliance . . . 19
3.5.1 Sarbanes Oxley Act . . . 20
3.5.2 Six Sigma . . . 20
3.5.3 GxP . . . 21

4 Problembereiche im Monitoring **23**
4.1 Quelldaten für Monitoring . . . 23
4.1.1 Datenquellen verteilt . . . 23
4.1.2 Unterschiedliche Trägerformate . . . 24
4.1.3 Unterschiedlicher Detailierungsgrad . . . 25
4.2 Kontext . . . 25
4.3 Orchestrierung . . . 26
4.3.1 Abgrenzung zu Choreography . . . 27
4.4 Echtzeitanalyse . . . 28

5 Anforderungen an Prozessmonitoring **31**
5.1 Generelle Anforderungen . . . 31
5.1.1 Unabhängigkeit vom Quellformat . . . 31
5.1.2 Nahe Echtzeit . . . 32
5.1.3 Keine Datenspeicherung . . . 32
5.1.4 Datenströme verarbeiten . . . 32
5.1.5 Abblidung/Zuordnung Kontext . . . 33
5.1.6 Prozessinstanz Monitoring . . . 33
5.1.7 Historisierung . . . 33
5.1.8 Grafisches Frontend . . . 34
5.1.9 Regelbasiert . . . 34
5.2 Anforderungen an Regel-Engine . . . 35
5.2.1 Abbildung von Prozessfluss . . . 35
5.2.2 Verarbeitung von Ausnahmen . . . 37
5.2.3 Filtern von Daten . . . 40
5.2.4 Zeitbezug . . . 40
5.2.5 Aggregation von Events . . . 40
5.3 Marktüberblick Monitoringsysteme . . . 41
5.3.1 Infrastruktur Monitoring . . . 41
5.3.2 Grid Monitoring . . . 43
5.3.3 Business Activity Monitoring . . . 46

6 CEP als Basis für Prozessmonitoring **49**
6.1 Gründe für Complex Event Processing als Basis . . . 49

7 Konzeption **51**
7.1 Architektur . . . 51
7.1.1 Serviceorientierte Architektur . . . 51
7.1.2 EAI Architektur . . . 53
7.1.3 Grid Monitoring Architecture . . . 55

7.1.4 Architektur Prozessmonitoring-System für serviceorientierte Architekturen . . . 57
7.2 Datenmodell Event . . . 61
7.2.1 Gründe für XML als Datenformat . . . 62
7.2.2 XML Schema des Datenmodells . . . 63
7.2.3 Event . . . 65
7.2.4 Datenbank Schema des Datenmodells . . . 69

8 Prototypische Implementierung **75**
8.1 Message Bus . . . 75
8.2 CEP Engine: Esper . . . 76
8.2.1 Abdeckung Anforderungen . . . 76
8.3 CEP Konfiguration . . . 77
8.3.1 CEP Konfigurationsdatei . . . 77
8.3.2 Beispiel . . . 78
8.3.3 Hotloading von CEP Konfigurationen . . . 79
8.4 Event Storage . . . 79
8.5 GUI . . . 82
8.5.1 Screenshot . . . 83
8.5.2 Aufbau des GUI . . . 83

9 Test **85**
9.1 Test Setup . . . 85
9.1.1 Überblick . . . 85
9.1.2 Testsystem . . . 86
9.1.3 Event Generator . . . 86
9.2 Testmethode . . . 87
9.2.1 Testfall . . . 87
9.2.2 Testrun . . . 88
9.3 Testfall Beschreibungen . . . 88
9.3.1 Grundfunktionalität . . . 88
9.3.2 CEP Query Sprache . . . 89
9.3.3 Hotloader CEP Konfiguration . . . 91
9.4 Lasttest . . . 94
9.4.1 Testaufbau . . . 95
9.4.2 Testdurchführung . . . 95
9.4.3 Konfiguration JMeter . . . 95
9.4.4 Testdaten . . . 97
9.4.5 Ergebnisse Lasttest . . . 98

10 Case Study **103**
10.1 Business Case . . . 103
10.1.1 Prozessbeschreibung . . . 103
10.1.2 Abhängigkeiten zu weiteren Prozessen . . . 104
10.1.3 Typische Probleme . . . 105
10.1.4 Ziele des Monitorings . . . 105
10.2 Services im Prozess . . . 106
10.3 Abbildung im GUI . . . 107
10.4 Szenarios . . . 108
10.4.1 Szenario 1: Alles OK . . . 108
10.4.2 Szenario 2: Fehlermeldung des Market Interface . . . 113
10.4.3 Szenario 3: Fehlermeldung eines tieferliegenden Service . . . 115
10.4.4 Szenario 4: Maximale Durchlaufzeit überschritten . . . 116
10.4.5 Szenario 5: Weiterverarbeitung von Basis-Eventdaten aus komplexen Events . . . 117
10.5 Durchführung . . . 119
10.5.1 Technische Umsetzung . . . 119
10.5.2 Testfälle . . . 119
10.6 Ergebnisse . . . 122
10.6.1 Abdeckung Anforderungen des Business Case . . . 122
10.6.2 Vorteile dieses Monitoring-Ansatzes . . . 123

11 Zusammenfassung und Ausblick **125**

A Lebenslauf **127**
A.1 Persönliche Daten . . . 127
A.2 Akademischer Werdegang . . . 127
A.3 Beruflicher Werdegang . . . 128

B Testdaten **129**
B.1 Testing . . . 129
B.1.1 Grundfunktionalität . . . 129
B.1.2 Hotloader CEP Konfiguration . . . 130
B.2 Case Study . . . 136
B.2.1 Eventdefinitionen . . . 136
B.2.2 CEP-Eventdefinitionen . . . 141

Abbildungsverzeichnis

2.1 Ebenen des Monitorings 7

3.1 BPMS Ansatz nach Dimitris Karagiannis 17
3.2 Prozess Tagesabschluss des Handelsystems 18

4.1 Services in unterschiedlichem Kontext 26
4.2 Fehler in zusammengesetztem Service 27
4.3 Choreography vs. Composition 28
4.4 Kreditkartenbetrug in Großbritannien 2005-2009 29

5.1 Sequence 35
5.2 Split 36
5.3 Merge 36
5.4 Work Item Failure 37
5.5 Deadline Expiry 38
5.6 Resource Unavailability 38
5.7 External Trigger 39
5.8 Constraint Violation 39

7.1 Serviceorientierte Architektur nach Sun Microsystems 52
7.2 EAI Architektur 54
7.3 Grid Monitoring Architektur 56
7.4 Architektur Prozessmonitoring-System 58
7.5 XML Schema Event 63
7.6 ER Diagram Datastore 70

8.1 Screenshot Lighthouse 3 GUI 83

9.1 Architektur des Testsystems 85
9.2 Screenshot JMeter 97
9.3 Verarbeitungsrate 101

9.4 Verarbeitungsdauer 101

10.1 Geschäftsprozess: Abwicklung Energiehandel 104
10.2 Screenshot Abbildung Prozess und Services im GUI 108
10.3 Szenario 1 Abbildung in JMeter 109
10.4 Szenario 1: Abbildung im GUI 113
10.5 Szenario 2: Fehler im Market Interface 114
10.6 Szenario 3: Fehler in Composite Service 116
10.7 Szenario 4: Verspätung im Prozess 117

Abkürzungsverzeichnis

APACS	Association for Payment Clearing Services
BAM	Business Activity Monitoring
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modelling Notation
BPMS	Business Process Management System
CEP	Complex Event Processing
CIO	Chief Information Officer
CORBA	Common Object Request Broker Architecture
DPMO	Defects per Million Opportunities
DSS	Decision Support Systems
EAI	Enterprise Application Integration
EPL	Esper Event Processing Language
EPTS	Event Processing Technical Sciety
ER	Entiy Relationship
ESB	Enterprise Service Bus
GMA	Grid Monitoring Architecture
GPL	General Public License
GUI	Graphical User Interface
IKS	Internes Kontroll System
IT	Informations Technologie
JMS	Java Messaging Service
JVM	Java Virtual Machine
KPI	Key Performance Indicators
MVC	Model View Controller
RFID	Radio-Frequency Identification
SOA	Service Oriented Architecture
SoC	Separation of Concerns
SOX	Sarbanes Oxley Act
SQL	Structured Query Language

UDF ..... User Defined Field
WfMS ..... Workflow Management System
XML ..... Extensible Markup Language

Kapitel 1

Einführung

In den vergangenen Jahren sind zwei große Trends zusammengewachsen. Der erste ist die Service-Orientierung und damit die serviceorientierte Architektur als neues Paradigma der Unternehmens-IT-Architektur. Der zweite Trend ist die Prozessorientierung. Technologien wie Webservices, BPEL oder BPMN haben zum Zusammenwachsen dieser beiden Trends geführt.

In vielen Publikationen wird Monitoring und Analyse von Geschäftsprozessen als wichtiger Bestandteil im Geschäftsprozessmanagement verstanden. Diese Aktivitäten stellen die Basis für Verbesserung der Prozesse dar. Die Umsetzung von Prozessmonitoring geschieht zumeist auf zwei verschiedene Arten. Entweder Prozesse werden in einem Workflow-Management-System abgebildet und das Monitoring damit der Workflow Engine überantwortet, oder es werden KPIs definiert und regelmäßig ausgewertet. Beide Lösungsansätze sind valide, haben jedoch verschiedene Nachteile.

Die KPI Methode hat zwei besondere Nachteile. Der erste Nachteil ist, dass diese Methode nur auf die Vergangenheit ausgerichtet ist. Zu definierten KPIs werden über die Zeit hinweg Quelldaten erhoben, die in gewissen Intervallen untersucht werden. Das zweite Problem ist, dass diese KPIs per Definition Indikatoren sind, die erst interpretiert werden müssen. Diese beiden Probleme zeigen, dass es mit dieser Methode nicht möglich ist, operative Probleme aufzuzeigen wenn sie auftreten. Damit ist diese Form des Prozessmonitorings für den operativen Betrieb nicht geeignet.

Der zweite Ansatz für Prozessmonitoring ist die Abbildung der Prozesse in einem Workflow-Management-System (WfMS). Dadurch, dass das WfMS die Steuerung des Prozesses und häufig auch der Datenflüsse übernimmt, können operative Probleme und technische Fehler innerhalb des BPMS aufgezeigt werden. So wird die Aufgabe des Monitorings von der Workflow-Engine übernommen. Problematisch an diesem Ansatz ist jedoch, dass sämtliche Prozesse in einem WfMS

abgebildet werden müssen, um durchgängiges Monitoring zu gewährleisten. Dies ist jedoch in der Realität sehr unwahrscheinlich, da dies mit sehr viel Aufwand verbunden ist.

In dieser Arbeit wird deshalb ein neuartiger Ansatz zum Monitoring von Prozessen präsentiert. Er basiert auf der Idee des Complex Event Processing und soll nahezu Echtzeit Monitoring von Prozessen und Prozessinstanzen ermöglichen. Der große Unterschied zu den bisherigen Ansätzen ist, dass es nicht nötig ist alle Prozesse in eine Ablaufumgebung zu integrieren. Typische Probleme im Prozessmonitoring wie zum Beispiel die Verteilung der Services, Abbildung von Prozessabläufen oder Wiederverwendung von Prozessen in unterschiedlichen Kontexten können mit Hilfe von CEP gelöst werden. Es ist nicht nötig alle Prozesse in eine Ablaufumgebung zu bringen und durch Workflow Engines ausführen zu lassen. Das Monitoring wird durch kontinuierliches Auswerten von Regeln, die den Output der Services nutzen, umgesetzt.

Kapitel 2

Grundlagen

Im Kapitel Grundlagen werden zunächst die wichtigsten Begriffe definiert. Dieser Schritt ist nötig, da viele Begriffe wie zum Beispiel "Service" einen weiten Spielraum für Interpretation bieten. Oftmals sind diese Begriffe Homonyme oder es fehlt eine eindeutige anerkannte wissenschaftliche Definition.

In solchen Fällen werden die relevantesten Definitionen aus der wissenschaftlichen Literatur präsentiert und die für diese Arbeit geltende bestimmt. Ist keine direkte Bestimmung möglich, weil keine Definition ausreichend ist, so wird eine eigene abgeleitet.

Ziel dieser Definitionen ist nicht die allgemeine Gültigkeit, sondern die Erhöhung des Verständnisses dieser Arbeit.

2.1 Prozess und Geschäftsprozess

Am Beginn der Neunziger Jahre prägten Hammer und Champy, sowie Davenport die Begriffe Geschäftsprozesses und Geschäftsprozess Reengineering (engl. Business Process Reengineering). [46] [50] Der Grundgedanke war eine radikale Neuausrichtung des Unternehmens an den Geschäftsprozessen, anstelle klassischer Strukturen. Der Ansatz des radikalen Business Process Reengineering war später starker Kritik ausgesetzt und sogar Davenport selbst schreibt in einem späteren Artikel:
When I wrote about "business process redesign" in 1990, I explicitly said that using it for cost reduction alone was not a sensible goal. And consultants Michael Hammer and James Champy, the two names most closely associated with reengineering, have insisted all along that layoffs shouldn't be the point. But the fact is, once out of the bottle, the reengineering genie quickly turned ugly. [47]
Trotz der Kritik blieb aber die Grundidee der Prozessorientierung bestehen. Ent-

sprechend diesem Paradigma soll im Rahmen dieser Arbeit untersucht werden, wie diese Prozesse systematisch überwacht werden können.

2.1.1 Definitionen in der Literatur

Wahrigs deutsches Wörterbuch beschreibt Prozess allgemein als *Ablauf, Verlauf, Vorgang.* [37]

Diese Definition ist noch sehr allgemein und kann durch die verschiedenen Definitionen in der wissenschaftlichen Literatur ergänzt beziehungsweise eingeschränkt werden. Die folgenden vier Definitionen behandeln den Begriff Geschäftsprozess.

2.1.1.1 Davenport (1993)

In definitional terms, a process is simply a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what.

A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers. [46]

2.1.1.2 Hammer & Champy (1993)

A collection of activities that takes one or more kinds of input and creates an output that is of value to the customer. [50]

2.1.1.3 Scheer & Zimmermann (1996)

Ablauf eines für die Wertschöpfung einer Organisation wichtigen Vorgangs von seiner Entstehung bis zu seiner Beendigung [72]

2.1.1.4 Gadatsch (2005)

Zielgerichtete, zeitlich-logische Abfolge von Aufgaben, die arbeitsteilig von mehreren Organisationen oder Organisationseinheiten unter Nutzung der Informations- und Kommunikationstechnologien ausgeführt werden können. [72]

2.1.1.5 Kühn & Karagiannis (2001)

Ein Geschäftsprozess ist eine Abfolge von Aktivitäten bzw. Subprozessen, die zur Erstellung eines Produktes von Akteuren durch Bearbeitung von Artefakten unter Zuhilfenahme von Ressourcen durchgeführt werden. [64]

2.1.2 Definition für diese Arbeit

Für diese Arbeit am passendsten ist die Definition von Gadatsch mit einer Erweiterung: Die arbeitsteilige Ausführung der Aktivitäten über Organisationen und Organisationseinheiten ist nicht mehr Grundbedingung.

Hintergrund ist, dass Monitoring hier sehr tief, auf einer niedrigen Detailebene beginnt und deshalb diese Einschränkung nicht mehr gelten soll. Durch diesen Schritt ist es möglich kleinere und detailliertere Betrachtungen zu machen. Bei so kleinen Prozessen noch von Geschäftsprozessen im allgemeinen zu sprechen wäre missverständlich, deshalb wird in dieser Arbeit nur noch von Prozessen gesprochen.

Definition Prozess

Unter einem Prozess versteht man in dieser Arbeit eine zielgerichtete, zeitlich-logische Abfolge von Aufgaben, die unter Nutzung von Informations- und Kommunikationstechnologien ausgeführt werden können.

2.2 Monitoring

Zunnächst soll der Begriff Monitoring für diese Arbeit gefasst werden. Im folgenden werden verschiedene Definitionen gegenübergestellt und danach der Begriff von dem verwandten Begriff des Measurements abgegrenzt.

2.2.1 Definitionen

In englischsprachigen, wie deutschen Wörterbüchern finden sich Definitionen zu dem Begriff Monitoring.

- **Cambridge Dictionary** *to watch and check a situation carefully for a period of time in order to discover something about it* [1]
- **Oxford English Dictionary** *keep under observation, especially so as to regulate, record, or control.* [2]

- **Webster Dictionary** *to watch, keep track of, or check usually for a special purpose* [3]
- **Duden** *Monitoring [zu engl. to monitor = überwachen, kontrollieren] Systematisches Beobachten, Analysieren und Auswerten einer bestimmten Situation* [4]

Für diese Arbeit am passendsten ist die Definition aus dem Oxford English Dicitionary, da sie neben Überwachung den Zweck der Regulierung, Aufzeichnung und Kontrolle beinhaltet.

2.2.2 Abgrenzung zu Measurement

Measurement, englisch für Messung ist eng mit dem Begriff Monitoring verwoben. In [63] werden die beiden Begriffe gut gegenübergestellt. Zu bemerken ist, dass Measurement eine wichtige Komponente fehlt: Die Kontinuität des Vorgangs. Monitoring ist ein fortgesetzter Vorgang, wobei Measurement einzelne, für sich alleinstehende Aktivitäten darstellt.

Aus diesem Grund wird in dieser Arbeit von Monitoring gesprochen, um den andauernden Charakter der Beobachtungen herauszustreichen.

2.2.3 Ebenen des Monitorings

Monitoring von Daten kann auf unterschiedlichen Ebenen geschehen. Je nach Detaillierungsgrad oder Sichtweise können unterschiedliche Informationen und Daten überwacht und ausgewertet werden.

In dieser Arbeit werden drei Ebenen des Monitorings unterschieden:

2.2.3.1 Infrastruktur

Dies ist die unterste Ebene wo Monitoring stattfinden kann. Informationen über die Verfügbarkeit von Netzwerkressourcen, die Auslastung von Festplatten oder auch die Antwortzeiten von Webservice Aufrufen werden dieser Ebene zugeordnet.

2.2.3.2 Applikation

In der zweiten Ebene werden Informationen überwacht, die von den Applikationen selbst generiert werden. Im Unterschied zur nächsten Ebene, dem Business Layer ist aber zur Interpretation dieser Daten kein domänenspezifisches Wissen nötig. Die Monitoringdaten können also unabhängig vom Anwendungskontext interpretiert werden.

2.2.3.3 Business

In der obersten Ebene, der Business Schicht oder auch Business Layer werden Daten überwacht, die Ergebnisse der Applikationen darstellen. Zu ihrer Interpretation ist zusätzliches Wissen erforderlich

2.2.3.4 Beispiel

Anhand dieses Beispiels wird die Abgrenzung der einzelnen Ebenen noch einmal verdeutlicht:

Ein Webservice berechnet anhand gewisser Inputdaten die maximale Größe eines Portfolios und gibt die Summe in Dollar zurück. Soll dieser kleine Prozess überwacht werden, so kann zum einen die Netzwerkverbindung zum Webservice überprüft werden, die durchschnittliche Antwortzeit und ähnliches. Dieses Monitoring würde in der Infrastruktur Ebene passieren.

In der Applikationsebene würde überwacht, ob die Daten maschienenlesbar angekommen sind, ob bei der Berechnung eine Ausnahme aufgetreten ist und ob das Webservice eine Antwort liefert.

Erst in der Business Ebene wird geprüft ob das Ergebnis semantisch korrekt ist. Hier wird das Ergebnis zum Beispiel $500.000 interpretiert und den Erwartungen der Endbenutzer gegenüber gestellt.

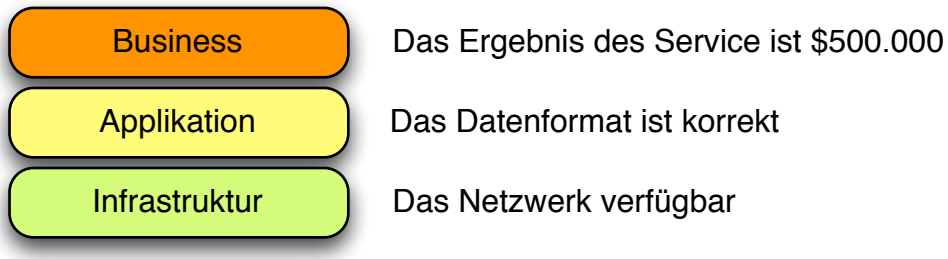

Abbildung 2.1: Ebenen des Monitorings

In diesem Beispiel könnte argumentiert werden, dass für die semantische Korrektheit des Ergebnisses das Webservice selbst zu sorgen hat. Selbstverständlich ist es möglich im Webservice zusätzliche Kontrollen, wie zum Beispiel Maximalwerte für das Ergebnis, oder Ähnliches einzubauen. Dies birgt aber zwei Probleme mit sich:

1. In Zusammenhang mit dem Architekturkonzept der Serviceorientierung und damit der angepeilten Wiederverwendung von Services wird durch eine der-

artige Maßnahme der Handlungsspielraum für die Wiederverwendung dieses Service eingeschränkt.

2. Die maximale Größe eines Portfolios kann sich im Laufe der Zeit häufig ändern, hier müsste jedes mal das Service selbst angepasst werden.

Aus diesen Gründen ist es sinnvoll Überwachungsaktivitäten wie diese zu externalisieren.

2.2.4 Definition Prozessmonitoring

Entsprechend der ausgewählten Definitionen in den vorhergehenden Abschnitten wird der Begriff Prozessmonitoring in dieser Arbeit wie folgt verwendet:

Prozessmonitoring ist die gemeinsame, andauernde Überwachung einzelner Aktivitäten, die in zeitlich, logischer Abfolge in einem gewissen Geschäftskontext einen Wert erbringen.

Diese Definition unterscheidet sich von der Definition von "Process Monitoring" in [63]. Die Autoren dieser Publikation definieren Prozessmonitoring als Überprüfung einer Situation. Im Gegensatz dazu unterstreicht die hier gültige Definition den Prozess.

2.3 Service

Für den sehr allgemeinen Begriff Service gibt es viele Definitionen. In dieser Arbeit soll der Begriff Service im Zusammenhang mit IT Architekturen verstanden werden.

In [73] wird der Begriff Service in diesem Zusammenhang sehr gut definiert:

Ein Service ist ein Dienst mit einer klar definierten Leistung. Für jeden Service gibt es eine oder mehrere Schnittstellen und einen Service-Contract. Der Service-Contract enthält normalerweise eine formale Beschreibung der Funktionalität, der Nutztung und der Anwendungsbedingungen des Service. Die Implementierung, also die konkrete Verarbeitungslogik und der Datenzugriff werden vor dem Benutzer versteckt.

Diese Definition eines Service entspricht grundsätzlich dem Verständnis dieses Begriffes in dieser Arbeit, wenngleich auch dem Service-Contract hier keine weitere Beachtung geschenkt wird. Zwar beschreibt der Service-Contract das Rahmenwerk eines Service und liefert dadurch wertvollen Input welche Aspekte für Monitoring interessant sein können, doch liegt der Fokus dieser Arbeit nicht auf der Übersetzung von Service-Contracts in Monitoring-Anforderungen, sondern

auf der Untersuchung der Durchführbarkeit von prozessorientiertem Monitoring in verteilten Architekturen.

Für diese Arbeit ist eine zusätzliche Erweiterung dieser Definition nötig. Die Anforderung der groben Granularitätsstufe eines Service gilt hier nicht. Hintergrund dieser Anpassung ist, dass auch schon auf der Ebene eines kleinen, einzelnen Webservice Monitoring ansetzen kann.

Aus diesem Grund wird hier noch zwischen *Business Service* und *technischem Service* unterschieden.

2.3.1 Business Service

Unter dem Begriff Business Service ist ein grobgranularer Dienst zu verstehen. Zu Vergleichen ist er mit einer Aktivität in einem Geschäftsprozess, jedoch muss der den Grundbedingungen eines Service, die oben definiert wurden entsprechen. Also ist ein Business Service ein Dienst mit einer klar definierten Leistung, deren Output direkt dem Geschäftszweck dient.

Das klassische Beispiel für ein grobgranulares Business Service ist die Flugreservierung. Unzählige Publikationen wie zum Beispiel [61] nutzen dieses Beispiel. Entsprechend soll dieses Beispiel auch hier genutzt werden.

Das Business Service *Flugbuchung* entspricht sehr gut der Definition von oben. Es ist grobgranular, kann gut durch einen Service-Contract beschrieben werden, die Leistung ist klar definierbar und der Output dient eindeutig dem Geschäftszweck.

2.3.2 Technischer Service

Hinter einem technischen Service steht in dieser Arbeit stets ein konkretes Stück Software. Bei einem technischen Service kann die Granularität bereits sehr fein sein. Trotzdem ist hier noch keine konkrete Technologie festgelegt. Es handelt sich auch auf einer sehr tiefen, feinen Ebene noch immer um Architekturbetrachtungen. Es gibt keine Festlegung konkreter Technologien zur Umsetzung von Services, wie Webservices oder CORBA .

Das zuvor genannte Service Flugbuchung besteht mit Sicherheit aus sehr vielen kleinen technischen Services. Ein Beispiel für ein solches wäre zum Beispiel *verfügbare Sitzplätze anzeigen*. Dieser kleine Bestandteil ist nötig, um das Business Service Flugbuchung erfolgreich durchzuführen.

2.3.3 Serviceorientierte Architektur

Für eine serviceorientierte Architektur gibt es sehr viele Definitionen. Die Standardisierungsorganisation OASIS [5] liefert eine abstrakte Beschreibung einer serviceorientierten Architektur:

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. [36]

Sieben weitere Definitionen des Begriffs *serviceorientierte Architektur* sind in [55] angeführt.

Eine für diese Arbeit sehr passende Definition von SOA liefern die Autoren Jan-Peter Richter, Harald Haller und Peter Schrey in [54]:

SOA ist ein Architekturmuster, das den Aufbau einer Anwendungslandschaft aus einzelnen fachlichen Anwendungsbausteinen beschreibt, die jeweils eine klar umrissene fachliche Aufgabe wahrnehmen. Die Anwendungsbausteine sind lose miteinander gekoppelt, indem sie einander ihre Funktionalitäten in Form von Services anbieten. [54]

Sie streichen zwei Elemente klar heraus, die auch in dieser Arbeit als besonders wichtig erachtet werden:

1. **Klare Trennung der Aufgaben**

   Vergleichbar mit dem Paradigma *Separation of Concerns (SoC)* bei der Programmierung, wird auch hier gefordert, Aufgaben in klar abgegrenzte Bereiche zu teilen. In der Programmierung wird angestrebt die Verflechtung von unterschiedlichen Aufgaben im selben Programmcode so gering wie möglich zu halten. Dazu sind verschiedene Design Patterns wie zum Beispiel *Model-View-Controller (MVC)* entstanden. MCV beispielsweise trennt die Darstellung für den Nutzer, die Verarbeitung der Eingaben und die Abbildung des Inhalts von einander.

   Diesem Beispiel folgend wird hier versucht fachlich unterschiedliche Aufgaben zu kapseln und klar zu trennen. Natürlich geschieht diese Trennung auf einer höheren Ebene. Man spricht deshalb auch von *programming in the large* und *programming in the small*. Ensprechend dieser Einteilung ist MVC dem programming in the small und SOA programming in the large zuzuordnen.

2. **Lose Kopplung**

   Ein weiterer wichtiger Bestandteil einer SOA ist die lose Kopplung der einzelnen Services. Auch hier können Analogien zur Programmierung hergestellt werden.

In der objektorientierten Programmierung sind die Begriffe Kohäsion und Kopplung Maßzahlen für die objektive Qualität des Codes. Die Kohäsion zeigt die Genauigkeit der Abbildung einer Aufgabe. Dagegen gibt die Kopplung die direkten Querverbindungen der einzelnen Abbildungen untereinander an. Robustheit, Testbarkeit, Sicherheit aber auch Verständlichkeit und vor allem Wiederverwendbarkeit des Programmcodes steigen bei niedriger Kopplung und hoher Kohäsion.

In einer serviceorientierten Architektur wird durch diese Elemente versucht eine flexiblere IT-Architektur zu erstellen. Durch den Fokus auf klare Trennung und lose Kopplung können einmal erstellte Services einfacher angepasst oder neu kombiniert werden.

Gerade in großen, gewachsenen IT Landschaften sind Anpassungen in der IT oft sehr kostenintensiv und riskant. So berichtete Martin Frick, der CIO von Avis Europe auf der Konferenz DW2008 in St. Gallen, dass die durchgängige Anpassung der IDs ihrer Niederlassungen einen zweistelligen Euromillionenbetrag kosten würde.

An Beispielen wie diesen kann leicht erkannt werden, wie groß der Bedarf an leichter anpassbaren IT-Systemen ist.

2.4 Event

2.4.1 Event

In diesem Kapitel werden die letzten Grundbegriffe dieser Arbeit definiert. Es handelt sich dabei um die Begriffe Event, Complex Event und Complex Event Processing. Die beiden Begriffe Complex Event und Complex Event Processing entstammen der Arbeit von Professor David Luckham. Er leitete das RAPIDE Projekt [6] an der Stanford University, das von der Defense Advanced Research Projects Agency (DARPA) [7] unterstützt wurde. Ziel dieses Forschungsprojekts war es, ein eventbasiertes Analyse- und Simulationswerkzeug zu schaffen.

Grundlage dieser Forschung ist der Begriff Event. Die Event Processing Technical Sciety (EPTS) definiert den Begriff Event im ersten Schritt ganz allgemein als:

Anything that happens, or is contemplated as happening. [48]

Als Beispiele werden genannt:

- A financial trade
- An airplane lands

- A sensor outputs a reading
- A change of state in a database or a finite state machine
- A key stroke
- A natural occurrence such as an earthquake
- A social or historical happening, e.g., the abolition of slavery, the battle of Waterloo, the Russian Revolution, and the Irish potato famine.

Neben dieser breiten und sehr allgemeinen Definition eines Events liefert die EPTS noch eine weitere Definition des Begriffs Event:

An object that represents, encodes, or records an event, generally for the purpose of computer processing. [48]

Auch die Beispiele umfasssen nun Events, die in IT-Systemen entstehen und dort verarbeitet werden können:

- A purchase order (records a purchase activity)
- An email confirmation of an airline reservation
- Stock tick message that reports a stock trade
- A message that reports an RFID sensor reading
- A medical insurance claim document

Diese Definition entspricht auch schon der Definition von Events, die David Luckham in seinem Buch *The Power of Events* beschreibt:

An event is an object that is a record of an activity in a system. The event signifies the activity. An event may be related to other events. [67]

Im Kontext dieser Arbeit wird Event ebenfalls grundsätzlich sehr allgemein aufgefasst. Events können durch sehr viele verschiedene Aktivitäten ausgelöst werden und sehr verschiedenartig sein. Fokus dieser Arbeit sind Events, die innerhalb von Unternehmen entstehen und Aktivitäten auslösen. Von besonderem Interesse sind Events, die in Computersystemen entstehen und zwischen unterschiedlichen Systemen ausgetauscht werden.

Um beispielsweise eine Flugbuchung durchzuführen, müssen unzählige Nachrichten zwischen unterschiedlichsten IT-Systemen fließen um den Prozess der Flugbuchung von der Sitzplatzreservierung bis zur Bezahlung zu ermöglichen. Jede Nachricht ist für jedes Computersystem ein Event, das wiederum weitere Arbeitsschritte, Nachrichten und damit Events nach sich zieht.

So betrachtet sind Events immer Ergebnisse und gleichzeitig auch Auslöser eines Arbeitsschritts. Events lösen Prozesse aus und treiben sie voran.

Diese Betrachtung liegt auch dem Begriff *Event Driven Architecture* zugrunde. Er beschreibt Computersysteme, die nur durch Events angestoßen werden und die ihrerseits wieder Events emittieren.

2.4.2 Complex Event

Events sind nie unabhängig [68]. Ein Event löst weitere Events aus, oder wird von anderen Events ausgelöst. Das folgende Beispiel soll dies Verdeutlichen.

2.4.2.1 Beispiel

Das Event *Flugticket jetzt drucken* wird von vielen anderen Events, wie zum Beispiel *Flugdaten eingegeben*, *Sitzplatz verfügbar*, *Sitzplatz reserviert*, *Zahlung erfolgreich* ausgelöst. Diese Events haben selbstverständlich eine klar definierte Reihenfolge und müssen zum selben Kontext gehören. So kann die Sitzplatzreservierung nicht vor der Verfügbarkeitskontrolle erfolgen und die Sitzplatzreservierung natürlich nicht beginnen, bevor die Flugdaten eingegeben wurden. Ebenso verhält es sich mit dem Kontext. Es reicht nicht, wenn zum korrekten Zeitpunkt das Event *Sitzplatz reserviert* eintritt. Es muss auch zum passenden Kontext, also dem korrekten Flug gehören.

2.4.2.2 Aggregation

Beim Event *Ticket jetzt drucken* spricht man also von einem Complex Event. Es besteht aus mehreren anderen Events, die über ein festes Regelwerk zusammengefasst werden können. So kann ein Complex Event auch als Aggregation seiner Einzelevents verstanden werden.

Ein Complex Event ist somit immer eine Aggregation mehrerer Events.

2.4.2.3 Abstraktion

Ein Complex Event kann aber auch eine Abstraktion mehrerer Events auf einer tieferen Ebene, wie zum Beispiel dem Infrastruktur Layer sein.

Beispiel

Der Complex Event *Feueralarm auslösen* ist in der Applikationsebene angesiedelt. Seine Bestandteile sind aber durchwegs auf der Infrastruktur Ebene angesiedelt. Er besteht zum Beispiel aus Events, die Raumtemperatur, Rauch und ähnliche Basisinformationen transportieren.

In dieser Arbeit wird der Begriff Event sowohl für Complex Events, als auch für einzelne Events genutzt. Hauptgrund dafür ist die einfachere Handhabung des Begriffs Event.

2.4.3 Complex Event Processing

Auf seiner Homepage definiert Professor David Luckham Complex Event Processing wie folgt:

The goal of CEP is to enable the information contained in the events flowing through all of the layers of the enterprise IT infrastructure to be discovered, understood in terms of its impact on high level management goals and business processes, and acted upon in real time. [8]

Dies ist auch die für diese Arbeit gültige Definition. Betrachtet man die einzelnen Schlagworte dieser Definition und stellt sie dem Ziel dieser Arbeit gegenüber so sind viele Übereinstimmungen erkennbar.

Speziell *discovered, understood in terms of its impact on high level management goals and business processes* entspricht dem Anspruch an Monitoring so wie es im Rahmen dieser Arbeit entwickelt werden soll.

2.4.3.1 Anwendungen von CEP

- **Algorithmic Trading**

  Unter *algorithmic trading* versteht man das Handeln an der Börse nach mathematischen Modellen. Events sind hier Stockticks und nach bestimmten Algorithmen werden Kauf und Verkaufsorders automatisch gefeuert.

- **Business Activity Monitoring**

  Ein weiterer Sektor wo CEP Technologie bereits sehr verbreitet ist, ist *Business Activity Monitoring (BAM)*. Im BAM wird versucht, gewisse Events so zu aggregieren, um möglichst in Echtzeit die Auslastung des Unternehmens anzuzeigen. In Dashboards werden dann einzelne Kennzahlen visualisiert.

Kapitel 3

Motivation für Monitoring

Motivationen für Monitoring gibt es viele. Hier seien ein paar grundlegende Bedürfnisse ausgeführt, die entsprechendes Monitoring notwendig machen. Zuerst wird die Fehlersuche, die mit Zunahme der Komplexität der IT Infrastrukturen exponential steigt betrachtet. Danach werden Themen, wie Prozessverbesserung oder externe Treiber, wie Compliance Richtlinien betrachtet.

3.1 Fehlersuche

Moderne IT-Architekturen und Systemlandschaften werden immer komplexer. Dies hat mehrere Gründe. Zum einen werden die Aufgaben, die die IT übernehmen muss immer größer. Zum anderen geht der Trend weg von großen Silosystemen hin zu verteilten Architekturen.

Betrachtet man nun eine verteilte Architektur, wie eine serviceorientierte Architektur, so erkennt man sehr schnell, wie komplex die Fehlersuche werden kann.

Beispiel

Bei einer online Flugbuchung bekommt ein Nutzer im letzten Schritt seines Buchungsvorgangs die Meldung *Buchung fehlgeschlagen* präsentiert. Dies kann eine Unzahl an Ursachen haben. Es ist sicher nicht unüblich, dass für verschiedenste Teile der Buchung unterschiedliche Systeme im Einsatz sind. Diese sind zumeist über Schnittstellen miteinander verbunden und tauschen so Daten aus.

Um die Fehleranalyse durchzuführen, ist es nötig alle Logs, aller beteiligten Systeme zu analysieren, um die Ursache eines Fehlers zu finden. Dieser Prozess kann sehr lange dauern, da verschiedenste Voraussetzungen dazu nötig sind.

- **Zugang zu Logging Information**

  Für eine eingehende Fehleranalyse ist es nötig, Zugang zu sämtlichen Informationen zu erhalten. Häufig sind die einzelnen Systeme verteilt und kein

zentraler Zugriff möglich. Bei anderen Systemen werden kaum oder gar keine Logging-Informationen erzeugt. Diese müssten erst im sogenannten *Debug Mode* laufen, damit ausreichend Information verfügbar ist.

- **Verständnis der Information**

  Besteht Zugriff auf die notwendigen Daten, so müssen diese interpretiert werden. Jedes System hat seine eigene Art zu loggen. Häufig werden wenig sprechende Fehlercodes ausgegeben, die erst in sinnvolle, verständliche Informationen übersetzt werden müssen.

- **Wiederherstellen des Fehlerfalls**

  Die komplexeste Aufgabe ist die Wiederherstellung des Umfeldes in dem ein Fehler aufgetreten ist. Es müssen alle beteiligten Applikationen in den gleichen Zustand versetzt werden, in dem sie zum Zeitpunkt des Fehlers waren.

3.2 Prozessverbesserung

Eine weitere Triebfeder für das Monitoring von Prozessen ist die kontinuierliche Verbesserung. Grundlage für diese Verbesserung können die Daten, die das Monitoring liefert sein.

Prozessverbesserung durch kontinuierliches Prozess Re-Engineering wird in [58] beschrieben. Das Vorgehen teilt sich in fünf verschiedene Teilprozesse: Strategic Decision Process, Re-Engineering Process, Resource Allocation Process, Workflow Management Process und Performance Evaluation Process. Abbildung 3.1 zeigt den Kreislauf der einzelnen Teilprozesse.

Im *Performance Evaluation Process* werden die Daten des *Workflow Management Process* qualitativ und quantitativ ausgewertet. Für Abbildung und Analyse kann ein BPMS verwendet werden, wie unter [57] beschrieben. Die verbreitetste Umsetzung dieser Idee ist in [56] beschrieben. Dabei wird davon ausgegangen, dass der zu evaluierende Prozess komplett in einem WfMS abgebildet ist und durch dieses ausgeführt wird.

Der durchgängige Einsatz von WfMS ist aber in der Realität sehr selten. Der Bedarf an Inputdaten für den Performance Evaluation Process bleibt aber bestehen. Diese Lücke soll das Prozessmonitoring füllen.

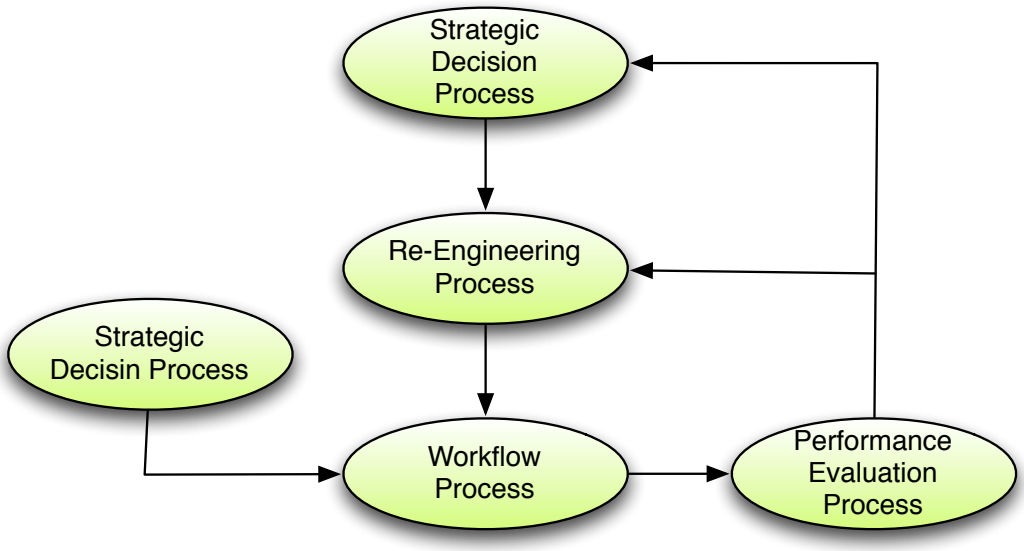

Abbildung 3.1: BPMS Ansatz nach Dimitris Karagiannis

3.3 Operations

Eine weitere Triebfeder für Monitoring ist der Wunsch möglichst in Echtzeit den Status einzelner Teilprozesse zu sehen. Manche Prozesse beinhalten sehr viele unterschiedliche Schritte die voneinander abhängig sind. Häufig haben Prozesse eine maximale Laufzeit.

Für den Prozessverantwortlichen ist aktuelles Feedback über den Zustand seiner Prozesse von hoher Wichtigkeit. Er muss sehen, in welchem Arbeitsschritt der Prozess gerade steckt und wie lange schon. Basierend auf diesen Daten kann er frühzeitig Massnahmen einleiten um einen rechtzeitigen Abschluss des Prozesses zu ermöglichen.

Beispiel Prozess Tagesabschluss des Handelsystems Zum Tagesabschluss müssen in einem Handelssystem verschiedene Arbeitsschritte durchgeführt werden. Wichtig ist, dass der Prozess bis zum nächsten Tag abgeschlossen ist, da sonst die Trader nicht Handeln können. Abbildung 3.2 zeigt diesen Prozess, wie er in einem Energiekonzern implementiert ist.

Der Prozessverantwortliche kann basierend auf seiner Erfahrung und den Daten aus dem Monitoring frühzeitig entsprechende Maßnahmen einleiten, damit der Prozess bis zum nächsten Arbeitstag erfolgreich abgearbeitet werden kann.

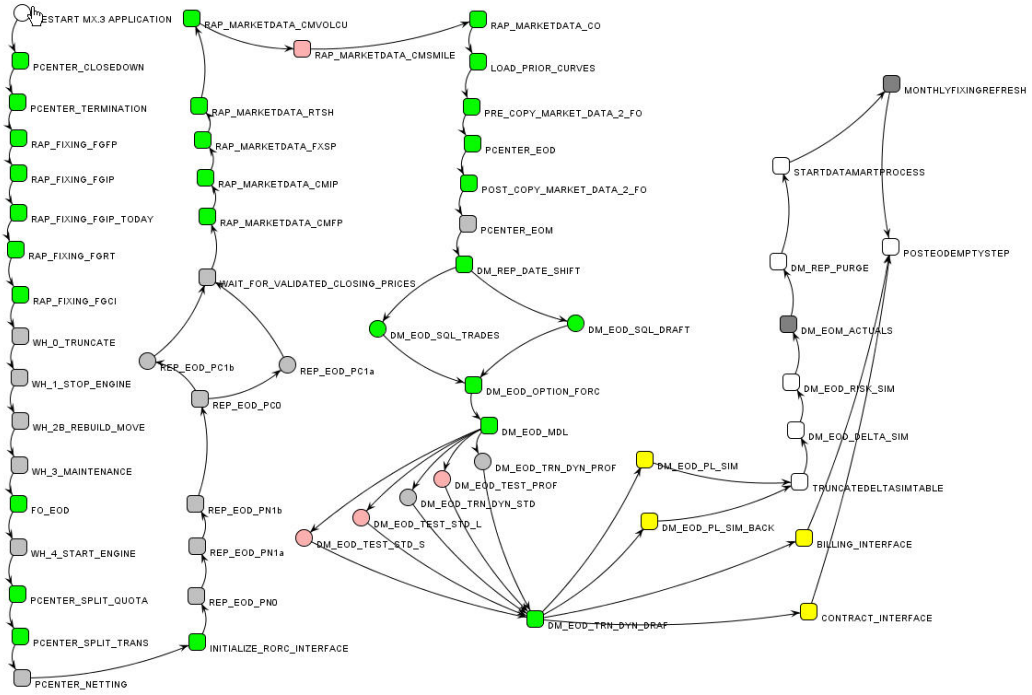

Abbildung 3.2: Prozess Tagesabschluss des Handelsystems

3.4 Entscheidungsunterstützung

Das Forschungsgebiet Decision Support Systems (DSS) ist bereits recht alt. Erste Veröffentlichungen zu diesem Thema reichen zurück bis ins Jahr 1978. [59] In den Neunzigern bekam das Thema einen neuen Schub durch die gestiegene Verbreitung von Business-Intelligence-Systemen. Systeme versuchen durch möglichst ausgefeilte Algorithmen geschäftsrelevante Daten und Trends aus einem großen Datentopf zu gewinnen. Diese Datentöpfe werden als Datawarehouse bezeichnet.

BI-Systeme haben aber den Nachteil, dass sie nicht in Echtzeit, sondern nur basierend auf historischen Daten Informationen liefern können. [39]

Im Dunstkreis von CEP beziehungsweise BAM wurde der Begriff *Real Time BI* entwickelt. Ziel dabei ist es, durch Monitoring und (nahezu) Echtzeit-Verarbeitung der Monitoringinformationen jederzeit den Zustand einzelner Prozesse oder Unternehmensbereiche ablesen zu können, um so qualifizierte Entscheidungen zu treffen.

Continental Airlines nutzt derartige Systeme in unterschiedlichen Bereichen. [38] Einer dieser Bereiche ist *Flight and Passenger Operations*.

Die realtime Daten über den aktuellen Zustand des Flugnetzes von United Airlines werden in Dashboards dargestellt. Neben Informationen über Flüge, sind für Continental Airlines vor allem Informationen über ihre High Value Customers von Interesse. Kommt ein Flug etwa verspätet an und der Anschlussflug ist an einem entfernten Gate, so liefert das System entsprechendes Feedback und ein Assistent am Flughafen kann diesen speziellen Passagieren Hilfestellung anbieten.

Ebenso wird auch Information über die Auslastung der einzelnen Mitarbeiter und die Anzahl der abzufertigenden Flüge abgebildet. Diese Information kann dann entsprechend genutzt werden, um zusätzliche Ressourcen freizugeben.

3.5 Compliance

Neben internen Treibern, wie Prozessverbesserung oder Operations gibt es noch weitere Treiber für Monitoring. Ein wichtiger Treiber ist der Themenbereich Compliance. Das sicher bekannteste Regulativ in diesem Bereich ist der Sarbanes Oxley Act (SOX). Andere Standards sind beispielsweise GxP. Manche dieser Treiber sind extern. Sie wirken von außen auf ein unternehmen und zwingen es gewisse Anforderungen an Prozesskontrolle zu erfüllen.

3.5.1 Sarbanes Oxley Act

Eines der bekanntesten Regulative ist der SOX. Er ist verpflichtend für alle Unternehmen die an amerikanischen Börsen notieren. Zusätzlich erstreckt es sich auch auf deren Tochterunternehmen und Dienstleister. [62]

Ziel des SOX ist es die Corporate Governance zu verbessern und die Transparenz der Buchführung sicherzustellen um Skandale wie Enron oder Worldcom zu verhindern.

Der für die IT relevante Teil ist die Sektion 404 des Sarbanes Oxley Act. Er verpflichtet die Unternehmen alle Prozesse, die zur Finanzgebarung und zur Berichtserstellung beitragen, einem entsprechenden Kontrollsystem zu unterwerfen. Entsprechend dieser Anforderungen sind sie gezwungen gegenüber einem Auditor die Vollständigkeit und Zuverlässigkeit ihres Internen Kontrollsystems nachzuweisen. Des weiteren müssen sie auch sicherstellen, dass die Prozesse (auch die ihrer Dienstleister) keine Schwachstellen aufweisen und nachvollziehbar sind.

3.5.2 Six Sigma

Die Six Sigma Strategie wurde ursprünglich von Motorola in den USA ca. im Jahr 1985 entwickelt. Ziel war es die Qualität der Prozesse deutlich zu verbessern. Grund dafür war die gestiegene Konkurrenz, vor allem aus Japan. Grundidee von Six Sigma ist es die Qualität und die Veränderungen klar bewertbar zu machen. [60] Dazu wurde die Maßzahl DPMO eingeführt. Sie gibt an wie viele Fehler pro einer Million Versuche entstehen.

Der Six Sigma Ansatz kennt 6 Sigma Level. Je höher das level umso geringer ist die Fehleranzahl pro Million Versuche. Sigma Level 1 erlaubt 691.482 Fehler je Million Ausführungen. In Sigma Level 4 sind nur noch 6.120 und Sigma Level 6 lediglich 3,4 Fehler je Million Ausführungen zulässig.

Six Sigma mag zwar aus der Produktion stammen, doch ist die Anwendung nicht auf Herstellungsprozesse beschränkt. Auch administrative Prozesse können abgebildet werden. [60].

Six Sigma hat ein einheitliches Vorgehen definiert, das unter dem Akronym DMAIC bekannt ist. DMAIC bedeutet Define, Measure, Analyse, Improve, Control.

1. **Define** In der ersten Phase des Six Sigma Kreislaufs werden grundsätzlichen Ziele definiert.

2. **Measure** In der nächsten Phase werden grundlegende Parameter identifiziert und die Rohdaten erhoben.

3. **Analyse** In dieser Phase werden die Daten analysiert, Schwachstellen aufgedeckt und Verbesserungspotentiale identifiziert.

4. **Improve** Nach der Identifikation der Verbesserungspotentiale werden gewisse Anpassungen vorgenommen.

5. **Control** Im letzten Schritt des Kreislaufs werden neue Monitoringmechanismen definiert und der Prozess damit weiter überwacht.

Speziell die Bereiche Measure, Analyse und Control benötigen entsprechende Unterstützung seitens einer stabilen Monitoringlösung.

3.5.3 GxP

GxP ist eine Sammlung verschiedener Best Practice Verfahren und in der Pharma Industrie weit verbreitet. G steht in diesem Fall für *Good* und P für *Practices*. Das x ist ein Platzhalter für unterschiedlichste Anwendungsgebiete. Es gibt zum Beispiel GMP für Manufacuring, GAP für Auditing, GEP für Engineering oder GITP für IT.

Diese Richtlinien sind grundsätzlich nicht verpflichtend, doch haben sie sich als Quasi-Standard etabliert und werden vor allem in Pharma-Unternehmen eingesetzt und häufig von ihren Dienstleistern auch eingefordert.

Der Nachweis der Einhaltung dieser Best Practices kann ebenfalls durch gezieltes Prozessmonitoring erfolgen.

Kapitel 4

Problembereiche im Monitoring

In diesem Kapitel werden die typischen Problemstellungen die das Monitoring von Prozessen in verteilten Architekturen mit sich bringt besprochen. Es beginnt zunächst auf Ebene der Daten, danach wird auf das Problem des Kontexts eingegangen. Im Anschluss daran werden noch die Probleme die bei Wiederverwendung von Services auftreten, sowie Probleme der Echtzeitanalyse erläutert.

4.1 Quelldaten für Monitoring

Das erste Problem dem ein Monitoringsystem gegenübersteht ist die benötigten Grunddaten zu erfassen. Die Probleme dabei sind ähnlich denen bei Business Intelligence Lösungen.

4.1.1 Datenquellen verteilt

Das erste Problem ist, dass die Quelldaten in einem Unternehmen sehr verteilt sind. Unterschiedlichste Systeme legen ihre Nutzdaten an unterschiedlichen Orten ab. Typische Vertreter sind:

- **Workstations** Quelldaten, die dem Monitoring dienen, liegen im schlechtesten Fall auf den einzelnen Workstations der Mitarbeiter. Dies können Office-Dokumente, oder Ergebnisse von Spezialprogrammen sein. Diese Daten sind am schwierigsten zu sammeln und auszuwerten.

- **Fileserver** Häufig werden Daten auch auf zentralen Fileservern abgelegt. Auch diese Daten sind schwierig zu erheben. Die Hauptprobleme sind, dass die Daten sehr unstrukturiert vorliegen und keinerlei Aussage über ihre Qualität gemacht werden kann.

- **Datenbanken** Viele Anwendungen nutzen Datenbankmanagement-Systeme um ihre Nutzdaten abzulegen. Dies hat zwar den Vorteil, dass die Daten strukturiert vorliegen und der Zugriff zentral möglich ist, jedoch ergibt sich durch die häufig sehr große Anzahl an verschiedenen Datenbanken in einem Unternehmen eine weitere Steigerung der Komplexität.

Mit der Verteilung der Daten geht noch ein weiteres Problem einher: Das Wissen um diese Daten, vor allem ihrer Aussagekraft und wie ein automatisierter Zugriff möglich wäre, ist ebenfalls sehr verteilt. Nicht selten gibt es in großen Unternehmen nur ein oder zwei Personen, die Bescheid wissen, wie auf diese Daten zugegriffen werden kann. Selbst einfache Probleme, wie benutzte Passwörter oder Servernamen sind oft nur schwer zu bekommen, da das Wissen genauso verteilt ist wie die Daten selbst.

Neben der physischen Verteilung der Daten bringt die Serviceorientierung noch weitere Herausforderungen mit sich. Da jedes Service atomar ist und für einen Prozess mitunter sehr viele unterschiedliche Services benötigt werden, ist die Anzahl an einzelnen Datenquellen ebenfalls sehr hoch.

4.1.2 Unterschiedliche Trägerformate

Das nächste Problem bei der Beschaffung der Quelldaten für das Monitoring ist das unterschiedliche Trägerformat der Datenquellen.

Viele Applikationen protokollieren wertvolle Informationen über ihren Zustand. Dies ist eine reichhaltige Quelle für Monitoringdaten. Leider gibt es aber überhaupt keinen Standard wie diese Information abgelegt wird. Verbreitete Varianten sind:

- **Logfiles** sind mit Sicherheit die verbreitetste Methode, wie Applikationen über ihren Zustand Auskunft geben. Der Apache Webserver [9] logt beispielsweise alle Zugriffe in ein eigenes Logfile. Das so genannte *AccessLog*. Ebenso logt er alle Probleme in ein weiteres Logfile, das *ErrorLog*.

- **Datenbanken** werden auch zur Ablage von Logging Daten genutzt. Der Robo Server von Kapow [10] legt beispielsweise Informationen zu Start, Stop und Problemen in einer Datenbank ab.

- **Nachrichten auf Message Bus** werden eher selten genutzt um Statusinformationen zu publizieren. Solche Lösungen findet man am ehesten im Bereich von EAI-Schnittstellen.

4.1.3 Unterschiedlicher Detailierungsgrad

Das nächste Problem bei der Schaffung der Grundlagen für das Monitoring ist, dass ebenso wie bei Trägerformat und Ablageort, kein Standard existiert, welche Informationen zumindest geloggt werden müssen. Wieviel Information eine Anwendung über ihren Zustand publiziert obliegt ihr selbst. Oft gibt es nicht einmal eine Richtlinie bei den Herstellern, es obliegt jedem Entwickler selbst wieviel Information er logt.

Am häufigsten werden Fehlermeldungen ausgegeben und in ein Logfile geschrieben, aber Start oder Ende einer Operation sind schon deutlich seltener zu finden.

Manche Applikationen können in unterschiedlichen Modi laufen. So gibt es häufig die Möglichkeit Applikationen im *Debug Mode* laufen zu lassen um zusätzliche Informationen zu erhalten.

4.2 Kontext

Ein großes Problemfeld im Bereich des Prozessmonitorings von Services ist das des Kontextes. Typischerweise werden Services so gestaltet, dass sie wiederverwendbar sind. Das impliziert aber auch, dass sie in unterschiedlichen Domänen zum Einsatz kommen können. Soll ein gesamter Prozess überwacht werden, ist es nötig die einzelnen Services in einem gemeinsamen Kontext zu betrachten. Diesen Kontext gibt der Prozess, in dem sie genutzt werden vor

Es kann bei einem Ergebnis eines Service nicht so einfach bestimmt werden, ob es korrekt oder als Fehler anzusehen ist. Erst durch Hinzunahme einer zusätzliche Betrachtungsebene, des Kontextes, kann entschieden werden ob die Antwort eines Service korrekt oder fehlerhaft ist. Ein Beispiel soll diesen Umstand verdeutlichen:

Beispiel

Abbildung 4.1 zeigt zwei Prozesse, die größtenteils die gleichen Services nutzen.

Der erste Prozess heißt *Intraday Risk Assesment* er wird regelmäßig mehrmals pro Tag durchlaufen. Er prüft sämtliche Portfolios aller Händler eines Trading Desks, damit kein Händler sein Risikolimit überschreitet. Dazu nutzt er drei Services:

1. **Service A1** bereitet Portfoliodaten für Currency Trader auf
2. **Service A2** bereitet Portfoliodaten für Commodity Trader auf

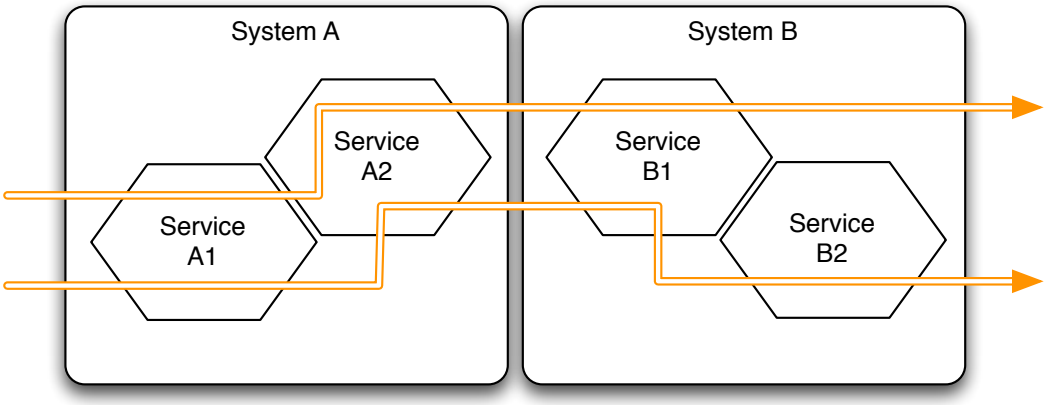

Abbildung 4.1: Services in unterschiedlichem Kontext

3. **Service B1** berechnet das Risikolevel eines Portfolios.

Je nach allgemeinem Risikolevel, Trader und Art des Portfolios sind unterschiedliche Grenzwerte definiert. Bei Abweichungen davon können entsprechende Schritte setzt werden.

Der Prozess 2 ist ein Reporting Prozess. Hier werden die gleichen Daten benötigt, wie im Prozess 1, jedoch werden hier die Daten lediglich für den täglichen Report zusammengefasst, der vom Service B2 erstellt wird. In diesem Prozess ist kein Alerting nötig.

Jeder dieser Prozesse nutzt die Services in einem unterschiedlichen Kontext. Entsprechend diesem Kontext sind die Ergebnisse der einzelnen Service unterschiedlich zu bewerten.

4.3 Orchestrierung

Einer der Grundgedanken der Serviceorientierung ist der der Orchestrierung. Darunter versteht man die Kombination von existierenden Services, zu neuen größeren Services, die einem speziellen Zweck dienen. Ähnlich einem Baukastensystem können Services miteinander verbunden werden.

Werden einzelne Services zu neuen Services kombiniert, so steigt automatisch die Komplexität der Fehlerbehandlung. Wenn eines der Services einen Fehler produziert, kann das gesamte neu erstellte Service gestört sein.

Abbildung 4.2 zeigt ein Beispiel. Auf der untersten Ebene sind neun einzelne Services zu sehen. Diese wurden zu zu drei neuen Services orchestriert. Das Service

A besteht wiederum aus diesen drei neu erstellten Services.

Passiert ein Fehler auf der untersten Ebene, so muss dieser Fehler an die darüber gelegenen Ebenen, also an die neu erstellten Services weitergegeben werden.

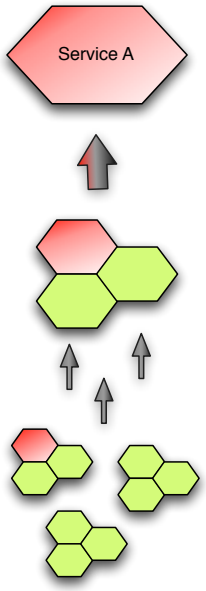

Abbildung 4.2: Fehler in zusammengesetztem Service

4.3.1 Abgrenzung zu Choreography

Im Zusammenhang mit der Orchestrierung wird häufig auch der Begriff der Choreography genannt. An dieser Stelle soll eine Abgrenzung gemacht und die Unterschiede herausgearbeitet werden.

Abbildung 4.3 zeigt den Unterschied zwischen Choreography und Composition. Der Begriff Choreography beschreibt die Erstellung von neuen Services durch die Kombination aus mehreren bestehenden einzelnen Services. Im Gegensatz dazu beschreibt der Begriff Choreography, wie einzelne Services genutzt werden um eine gewisse Tätigkeit zu verrichten.

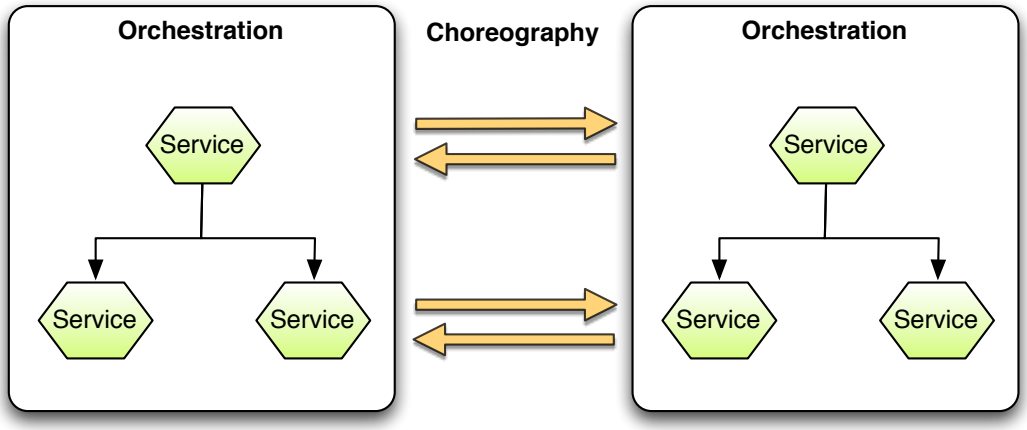

Abbildung 4.3: Choreography vs. Composition

4.4 Echtzeitanalyse

Ein weiteres Problem beim Monitoring ist, dass die Events möglichst in Echtzeit verarbeitet werden sollen. Dies ist nötig da eine der Erwartungen an ein gutes Monitoringsystem ist, Probleme sofort anzuzeigen.

Betrachtet man das Beispiel der United Airlines, die erkennen wollen welche ihrer wertvollsten Kunden unter Umständen Unterstützung beim Umsteigen benötigt, so wird schnell klar, dass die Verarbeitung der Events schnell passieren muss. Ist die Verarbeitung der Events zu langsam, so kann nicht rechtzeitig Unterstützung geboten werden. Darunter wird der Service leiden und der Kunde unter Umständen zu einer Konkurrenzairline wechseln.

Gerade der Anspruch an Echtzeitverarbeitung bringt aber einige technische Herausforderungen an die Infrastruktur und vor allem an die Verarbeitungskomponente mit sich.

Zunächst muss der schnelle Durchsatz von Nachrichten garantiert sein. Es ist zu erwarten, dass die Anzahl an Events sehr groß wird.

Betrachtet man zum Beispiel folgendes Szenario aus dem Bereich Fraud Detection bei Kreditkarten.

Beispiel Eine Kreditkarte wird kopiert, um damit Zahlungen durchzuführen. Dass dieses Szenario sehr relevant ist, zeigen die Zahlen der APACS für 2008. Der Gesamtschaden der 2008 in Großbritannien durch kopierte Kreditkarten entstan-

den ist betrug 169,8 Millionen Pfund. [11]

Card Fraud Type - on UK issued credit and debit cards	2005	2006	2007	2008
Phone, internet and mail order fraud (Card-not-present fraud)	£183.2m	£212.7m	£290.5m	£328.4m
Counterfeit (skimmed/cloned)fraud	£96.8m	£98.6m	£144.3m	£169.8m
Fraud on lost of stolen cards	£68.5m	£68.5m	£56.2m	£54.1m
Card ID theft	£31.9m	£31.9m	£34.1m	£47.4m
Mail non-receipt	£15.4m	£10.2m	£10.2m	£10.2m

Abbildung 4.4: Kreditkartenbetrug in Großbritannien 2005-2009

Um zu verhindern, dass mit kopierten Kreditkarten Zahlungen durchgeführt werden, könnte bei jeder Zahlung der Ort der Zahlung ausgewertet werden. Wenn zwei Zahlungen zur gleichen Zeit an unterschiedlichen Orten stattfinden, ist dies ein eindeutiges Indiz für Betrug.

Betrachtet man aber die Gesamtanzahl aller Kreditkartentransaktionen so ist die Datenflut, die verarbeitet werden muss um Betrug zu erkennen kaum vorstellbar. Trotzdem sollte diese Überprüfung sehr schnell gehen, da Kunden nicht 20 Minuten an der Kasse auf die Ergebnisse warten können.

Kapitel 5

Anforderungen an Prozessmonitoring

Nach der Motivation für Monitoring und einer eingehenden Analyse von Problembereichen des Monitorings sollen nun die grundsätzlichen Anforderungen an ein Prozessmonitoring-System für serviceorientierte Architekturen definiert werden. Zunächst werden generelle Anforderungen, wie Verarbeitung von Datenströmen oder Unabhängigkeit vom Quellformat beschrieben. Im Anschluss daran werden die Anforderungen an das Herzstück des Prozessmonitorings, der Verarbeitungskomponente definiert. Im letzten Schritt werden unterschiedliche am Markt verbreitete Monitoringsysteme klassifiziert und vorgestellt.

5.1 Generelle Anforderungen

In diesem Kapitel werden die generellen Anforderungen an ein Prozessmonitoring-System definiert.

5.1.1 Unabhängigkeit vom Quellformat

Eine der bestehenden Problemstellungen im Monitornig ist, dass die Quelldaten sehr unterschiedlich vorliegen. Sowohl die Datenformate, als auch die Ablageorte und Trägerformate sind wie in Abschnitt 4.1 bereits erläutert über viele verschiedene Systeme verteilt.

Aus diesem Grund muss ein Monitoringsystem von den Datenformaten und Trägermedien unabhängig sein. Dies ist ein zentrales Kriterium, weil der Anspruch des Monitorings gesamter Prozesse besteht. Es kann hier nicht davon ausgegangen

werden, dass alle am Prozess beteiligten Services und Applikationen von Haus aus passende Daten liefern.

5.1.2 Nahe Echtzeit

Eine weitere Anforderung an Prozessmonitoring ist, dass es so schnell wie möglich, am besten in Echtzeit funktionieren muss, wie das bereits in 4.4 vorgestellte Beispiel anschaulich darstellt. Die Prüfung ob eine Kreditkarte geklont ist oder nicht, ist nur dann beim Zahlungsvorgang möglich, wenn der zeitliche Aufwand dieses Prüfschrittes in einem akzeptablen Bereich bleibt.

5.1.3 Keine Datenspeicherung

In [74] definieren die Autoren acht Regeln für realtime stream processing. Eine davon ist *Keep the Data Moving*. Operationen zur Persistierung der Daten sind kostspielig in Bezug auf die Zeit. Es wird empfohlen, die Verarbeitung der Daten direkt durchzuführen und auf eine Speicherung dabei zu verzichten.

Entsprechend dieser Überlegungen ist eine weitere Anforderung an ein Monitoringsystem, dass es nicht auf fixen Datenbasis aufbauen darf. Systeme, wie zum Beispiel ein Datawarehouses, die für die Generierung von Informationen statische Daten benötigen sind tendenziell zu schwerfällig und zu langsam.

5.1.4 Datenströme verarbeiten

Wie im Abschnitt 5.1.3 definiert, darf die Verarbeitungskomponente nicht auf einem Datenpool basieren. Vielmehr ist es nötig, dass die Verarbeitung von Datenströmen möglich ist.

Publizieren alle Services bei jeder Ausführung Start- und Endzeitpunkt, sowie das Ergebnis ihrer Operation, so entsteht dadurch ein kontinuierlicher Strom an Daten.

Beispiel

Durch das Monitoringsystem soll der Prozess der Risikobewertung überwacht werden. Dazu ist es nötig, die Daten sämtlicher offener Positionen aller Händler eines Trading Desks auszuwerten. Diese Aufgabe stellt aber ein Moving Target dar, da Händler schnell Positionen öffnen und wieder schließen können.

Dieser Prozess kann nur überwacht werden, wenn es möglich ist auf einen kontinuierlichen Datenstrom Operationen durchzuführen.

An diesem Beispiel erkennt man, warum es nötig ist kontinuierliche Datenströme verarbeiten zu können.

5.1.5 Abblidung/Zuordnung Kontext

Wie bereits im Abschnitt 4.2 ausgeführt ist die Abbildung des Kontext eine sehr komplexe Aufgabe. Trotzdem ist es nötig, Services in ihrem Anwendungskontext zu monitoren. Anders ist es nicht möglich qualifizierte Aussagen über ihre Outputs zu geben.

Monitoring der Infrastruktur Ebene und der Applikationsebene lassen sich leicht umsetzen. Auf der ersten Ebene können Parameter, wie Verfügbarkeit, oder Antwortzeit gemessen werden. Auf der Applikationsebene ist es dann möglich, Antworten des Service direkt zu überwachen. Liefert das Service eine Antwort? Ist diese Antwort eine Fehlermeldung? Dies sind Fragen, die sich ohne Kenntnis des Kontextes auswerten und konkreten *OK* oder *Nicht OK* Meldungen zuordnen lassen.

Monitoring auf der Business Ebene ist der Kontext aber wichtig. Liefert ein Service das Ergebnis 500 zurück, so kann das je nach Kontext eine Antwort sein, die innerhalb der für das Business akzeptablen Schranken liegt. In einem Anderen Kontext kann die Antwort 500 aber schon äußerst problematisch sein und entsprechende Stellen müssen sofort informiert werden. [51]

5.1.6 Prozessinstanz Monitoring

Eine erweiterte Anforderung an das Prozessmonitoring ist das Monitoring von Prozessinstanzen. Eine Prozessinstanz ist ein konkreter Durchlauf eines Prozesses.

Es gibt durchaus Prozesse, die im Minutentakt durchgeführt werden. Meldet das Prozessmonitoring nun einen Fehler ist es natürlich von Interesse an welcher Stelle der Fehler aufgetreten ist, doch in diesem Fall ist noch von viel größerem Interesse in welcher Prozessinstanz der Fehler aufgetreten ist. Die Inputs und Outputs jedes einzelnen Service in dem konkreten Durchlauf, der einen Fehler hervorgerufen hat sind für die Ursachenforschung von größter Wichtigkeit. Nur wenn alle Parameter der gesamten Prozessinstanz bekannt sind, kann ein vollständiges Bild des Fehlers gezeichnet werden.

5.1.7 Historisierung

Die Datenspeicherung zum Zwecke der Verarbeitung der Events für das Monitoring wurde im Punkt 5.1.3 ausgeschlossen. Eine andere Form der Datenspeicherung ist aber trotzdem notwendig.

Meldet das Monitoring eine Auffälligkeit, so sind die Daten, die zu dieser Meldung geführt haben von Interesse. Um später einmal nachvollziehen zu können, welche Daten zu einer bestimmten Meldung geführt haben, ist es notwendig diese Daten zu speichern. Selbstverständlich muss die Architektur es erlauben, die Datenspeicherung von der Verarbeitung der Events soweit zu trennen, dass die Verarbeitung nicht beeinträchtigt wird.

5.1.8 Grafisches Frontend

Eine besonders wichtige Anforderung an ein Prozessmonitoring-System ist die graphische Benutzeroberfläche. Ein Benutzer soll den Prozess sehen können. Dies erleichtert das Verständnis der Zusammenhänge.

5.1.8.1 Ampeln

Neben der grafischen Darstellung eines gesamten Prozesses ist es auch wichtig dem Benutzer ein visuelles Feedback über den Zustand eines Prozesses und seiner einzelnen Schritte zu geben. Dies ist eine Grundanforderung an ein Prozessmonitoring Frontend.

5.1.8.2 Drill Down

Zeichnet man einen Prozess auf und modelliert man jedes Detail bis auf die Ebene einzelner technischer Services und Datenflüsse, so wird der Prozess schnell sehr groß und man verliert als Nutzer die Übersicht. Um diesem Problem beizukommen ist ein so genannter *Drill Down* wünschenswert.

Unter Drill Down versteht man die Möglichkeit in einen Prozess hineinzuzoomen. Diese Funktionalität kann zum Beispiel abgebildet werden, indem in der Benutzerschnittstelle Prozessschritte als Subprozesse definiert werden. Ein Doppelklick auf diesen Prozessschritt öffnet dann den Prozess der dahinter liegt.

5.1.9 Regelbasiert

Eine der zentralsten Anforderungen an das Monitoringsystem ist die Möglichkeit Complex Events durch eine Regelsprache zu definieren. Typischerweise werden Monitoringanforderungen als Regel ausgedrückt.

Beispiel

Eine typische Anforderung wie sie von Business formuliert werden könnte wäre zum Beispiel: Wenn Trader X im Commodity-Portfolio das Risikolevel von 200 übersteigt, dann muss sofort sein Vorgesetzter benachrichtigt werden.

Um solche Anforderungen abbilden zu können ist eine Regelsprache nötig. Die Verarbeitungskomponente hat dann die Aufgabe möglichst schnell alle Regeln auf sämtliche Events im Datenstrom anzuwenden und gegebenenfalls weitere Aktionen auszulösen.

5.2 Anforderungen an Regel-Engine

Wie aus Punkt 5.1.9 hervorgeht ist Hauptaufgabe die Verarbeitungskomponente Regeln möglichst schnell zu interpretieren. Für diese Aufgabe wurden eigene Regelengines entworfen. Für die Anwendung im Prozessmonitoring gibt es außer den bereits angeführten Anforderungen wie zum Beispiel Geschwindigkeit noch weitere Herausforderungen für die Regelengine.

5.2.1 Abbildung von Prozessfluss

In [76] werden generelle Workflow Patterns beschrieben, wie sie in der Modellierung von Prozessen immer wieder vorkommen. An dieser Stelle werden die wichtigsten Control Flow Patterns vorgestellt, da zumindest diese durch Regeln und GUI abbildbar sein müssen.

5.2.1.1 Sequence

Aktivitäten werden in einer bestimmten Reihenfolge nacheinander abgearbeitet.

Beispiel

Die drei Aktivitäten A,B,C müssen in der Reihenfolge A vor B und B vor C abgebildet werden. (Abbildung 5.1)

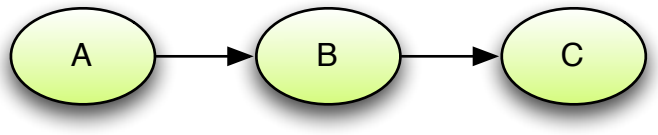

Abbildung 5.1: Sequence

5.2.1.2 Split

Einer Aktivität folgen zwei, oder mehrere andere Aktivitäten nach.

Beispiel

Der Aktivität A folgen sowohl Aktivität B, als auch C nach. (Abbildung 5.2)

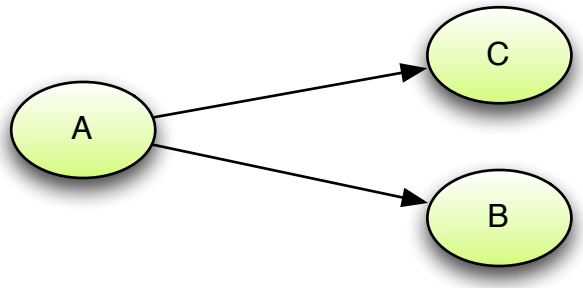

Abbildung 5.2: Split

5.2.1.3 Merge

Zwei, oder mehrere Aktivitäten folgen genau einer Aktivität nach

Beispiel

Den Aktivitäten A und B folgt Aktivität C. (Abbildung 5.3)

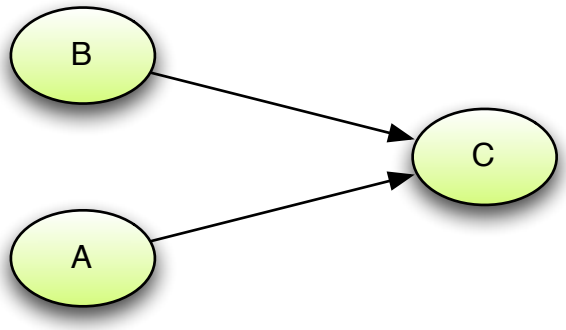

Abbildung 5.3: Merge

5.2.2 Verarbeitung von Ausnahmen

Neben der Analyse von Prozessflüssen beschäftigte sich die Arbeitsgruppe von Prof. Aalst auch mit Ausnahmen (Exceptions) in Workflows. In [71] wurde die Behandlung von unvorhergesehenen Ausnahmen in Prozessen wissenschaftlich aufgearbeitet.

Die Abbildung dieser Ausnahmen ist für ein Prozessmonitoring-System natürlich ein integraler Bestandteil. Darum werden diese Ausnahmen hier einzeln vorgestellt.

5.2.2.1 Work Item Failure

Der einfachste Fall einer Exception ist eine direkte Fehlermeldung eines Service. Solche Fehlermeldungen müssen von der Rules Engine aus dem Datenstrom der Events herausgefiltert werden können. Ein Beispiel für eine solche Exception wäre, wenn einem Service unerwartet Inputparameter, wie zum Beispiel ein Datum in einem falschen Datumsformat übergeben wurde. Das Service würde dann eine Fehlermeldung, wie *Error: Date format unknown* zurückgeben.

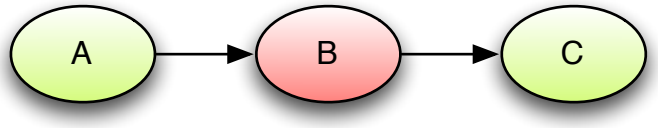

Abbildung 5.4: Work Item Failure

5.2.2.2 Deadline Expiry

Nicht nur gesamte Prozesse haben maximale Laufzeiten, die nicht überschritten werden dürfen (vgl. Beispiel Tagesabschluss in Abschnitt 3.3), sondern auch einzelne Arbeitsschritte. Auch diese Fälle sollten durch Regeln abbildbar sein, damit sie entsprechend überwacht werden können.

Der Typ *Deadline Expiry* beschreibt, wenn ein Service einen gewissen definierten Zeitraum keine Antwort gibt. Die kann zum Beispiel vorkommen, wenn ein Serviceaufruf ein Deadlock im Service verursacht hat und das Service nicht mehr weiterarbeiten kann.

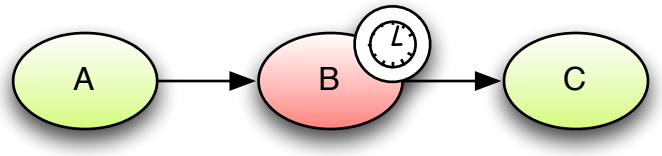

Abbildung 5.5: Deadline Expiry

5.2.2.3 Resource Unavailablility

Eine weitere Exception auf die Rücksicht genommen werden muss ist *Resource Unavailability*. Darunter versteht man die Ausnahme, wenn eine erwartete Ressource zum Zeitpunkt des Aufrufs nicht verfügbar ist.

Ein Beispiel dafür ist zum Beispiel ein Webserver, von dem Daten geholt werden müssen. Ist dieser zum Zeitpunkt des Zugriffs offline, so wird das eine *Resource Unavailability Exception* hervorrufen.

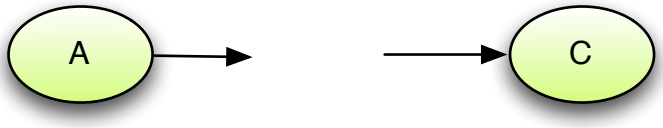

Abbildung 5.6: Resource Unavailability

5.2.2.4 External Trigger

Externe Events können auf Arbeitsschritte in einem Prozess wirken. Sie werden verwendet um anzuzeigen, dass ein bestimmtes Ereignis eingetreten ist, das in diesem Arbeitsschritt behandelt werden soll. Solche externe, also nicht zum eigentlichen Prozess gehörige, Ereignisse könnten zwar schon Design des Arbeitsschrittes oder des Service berücksichtigt werden, doch ist häufig nicht klar ob und zu welchem Zeitpunkt ein solches Event eintreten wird. Deshalb werden solche Events durch das Exceptionhandling verarbeitet.

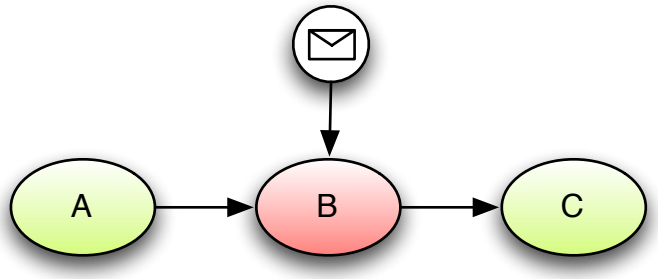

Abbildung 5.7: External Trigger

5.2.2.5 Contstraint Violation

Für Ergebnisse gewisser Arbeitsschritte ist es notwendig Constraints zu definieren. Ein Beispiel dafür wurde schon in Abschnitt 5.1.5 vorgestellt. In diesem Abschnitt wurde ausgeführt, dass Ergebnisse von von Serviceoperationen nur innerhalb ihres Businesskontextes bewertet werden können. Der Kontext ist nur ein Teil der Bewertung, der zweite ist der Constraint selbst.

Beispiel

Ein Service berechnet das Risikolevel eines Portfolios. Hier ist der Kontext die *Intraday Risikoananlyse*. Der Constraint muss dann aber eine fix vorgegebene Zahl sein, die das Level nicht überschreiten darf, also z.B. 500.

Solche Constraints müssen durch Regeln abbildbar sein, damit Monitoring auf Businessebene möglich ist.

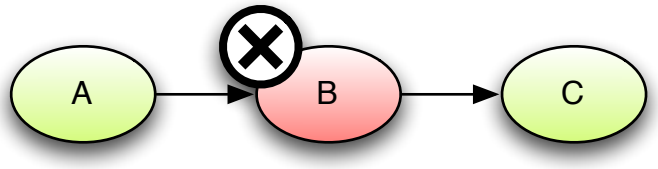

Abbildung 5.8: Constraint Violation

5.2.3 Filtern von Daten

Um viele der oben angeführten Exception Types in Regeln abbilden zu können, muss die Regelsprache über extensive Filtermöglichkeiten verfügen. Sollen beispielsweise Constraint Violations überwacht werden, so müssen aus dem Datenstrom genau die Events gefiltert werden die dem *Service*, dem *Kontext* entsprechen und deren *Constraint* verletzt wird.

Abgeleitet von der SQL sollen Regeln für die Verarbeitung von Events vergleichbare Funktionalität bieten.

- **Filtern von Text**

  Um aus allen Events im Datenstrom die passenden herausfiltern zu können, ist gezielte Suche nach Texten und Textfragmenten nötig. Neben der Suche nach exakt übereinstimmenden Texten, ist auch eine Filterung unter Zuhilfenahme von Wildcards notwendig, um auch weniger genau definierte Muster oder Gruppen filtern zu können.

- **Filtern von Zahlen und Zahlenbereichen** Vor allem für die Prüfung von Constraint Violations ist der Vergleich von Zahlen und Zahlenbereichen nötig. Größer als ($>$), kleiner als ($<$) und gleich ($=$) müssen als Operatoren vorhanden sein, um entsprechendes Filtering zu ermöglichen.

5.2.4 Zeitbezug

Eine besondere Anforderung an die Regelsprache ist der Zeitbezug. Da Events immer zu bestimmten Zeitpunkten auftreten und diese auch für das Monitoring von besonderer Wichtigkeit sind, ist es nötig Zeitpunkte und Zeitfenster in die Abfragen einzubauen.

- **Zeitpunkt** Manchmal ist es nötig, dass gewisse Zeitpunkte eingehalten werden. Zum Beispiel: Keine Position darf nach 19 Uhr noch offen sein in einem bestimmten Portfolio.

- **Zeitfenster** Andere Anforderungen benötigen die Abbildung von Zeitfenstern in der Regelsprache. Zum Beispiel: Der Durchschnittskurs der letzten 10 Minuten darf nicht unter $100 liegen.

5.2.5 Aggregation von Events

Eine weitere wichtige Anforderung an eine Rules Engine ist die Unterstützung der Aggregation von Events. Beliebige Events müssen zu complex events kom-

binierbar sein. Für die Kombination sind boolsche Operatoren geeignet, da mit ihrer Hilfe eine Events flexibel kombiniert werden können.

So können Szenarios, wie zum Beispiel folgendes abgebildet werden: Wenn Event A = 500 ist *und* der Kontext K *oder* S ist, so muss ein Email an riskalert@compamy.com geschickt werden.

5.3 Marktüberblick Monitoringsysteme

Im folgenden Abschnitt sollen die verbreitetsten Monitoringtools und Systeme klassifiziert und einzeln vorgestellt. Die Klassifikation erfolgt basierend auf der Anwendungsdomäne für die die einzelnen Tools entwickelt worden sind.

5.3.1 Infrastruktur Monitoring

In den vergangenen Jahren ist die Größe und Verteilung von IT Infrastrukturen stark gestiegen. Gleichzeitig sind die Anforderungen an Verfügbarkeit und Belastbarkeit der Infrastruktur enorm gestiegen. Selbst kleinste Ausfälle können schon den vorübergehenden Stillstand im Unternehmen bedeuten.

Dementsprechend hoch ist auch der Bedarf an zentraler Überwachung und Verwaltung durch den Betrieb. Für diese Anwendung wurden diverse Tools entwickelt. Die wichtigsten Vertreter seien hier vorgestellt.

5.3.1.1 Nagios

Nagios ist ein open source Werkzeug zum Monitoring von IT Infrastrukturen. Es beherrscht alle gängigen Netzwerkprotokolle, sowie das Monitoring typischer Hostinformationen wie Speicherauslastung oder Prozessorlast. Weiters bietet Nagios durch ein Plugin-System die Möglichkeit es flexibel zu erweitern. Informationen können gruppiert werden um die Übersicht im Webfrontend zu erhöhen. Neben dem Webfrontend bietet Nagios als zusätzliches Feature verschiedene Alertingmechanismen an.

5.3.1.2 HP OpenView

HP OpenView ist eine der verbreitetsten kommerziellen Lösungen im Bereich Monitoring. Mittlerweile ging das Produkt in mehreren weiteren Anwendungen auf, die zusammen das HP Operations Center bilden.

Das HP Operations Center besteht aus verschiedenen Produkten, die entsprechend der Anforderungen kombiniert werden können. Grundbaustein dieser

Monitoringlösung ist der HP Operations Manager. Er teilt sich in drei einzelne Bausteine.

1. **Operations Mananger Agents** Die Agents haben die Aufgabe den Server mit passenden Monitoringinformationen zu versorgen. Zusätzlich besitzen sie Filter um gewisse Events zu korrelieren oder doppelte zu vermeiden um so das Datenvolumen für die Übertragung und den Server möglichst gering zu halten.

2. **Operations Manager Server** Aufgabe des Operations Manager Servers ist es die Monitoringinformation der Agents aufzubereiten und für spätere Analysen abzulegen. Neben diesen Aufgaben übernimmt er Alerting und die Verwaltung von Monitoring-Policies.

3. **Operations Manager Console** Die Operations Manager Console ist das grafische Frontend für den Anwender. Es ist in zwei verschiedenen Ausführungen verfügbar. In der ersten fügt es sich als Microsoft Management Console vollständig in die Microsoft Windows Server Administrative Tools ein. In der zweiten Ausführung gibt es die Console auch als Webinterface. Durch die Console wird der Operations Manager Server gesteuert und konfiguriert.

Quellen: [42], [41]

5.3.1.3 IBM Tivoli Monitor

IBM Tivoli Monitoring ist ein Bestandteil des Tivoli System Management Frameworks. Hauptaufgabe von Tivoli Monitoring ist das Monitoring von Betriebssystemen, Datenbanken, Servern. Dazu nutzt IBM Tivoli Monitoring ebenfalls eine dreischichtige Architektur.

1. **Agent** Der Agent sammelt die benötigte Monitoringinformation und gibt sie an den IBM Tivoli Server weiter.

2. **Server** Der Server nimmt die Monitoringinformationen der Agents entgegen und bereitet sie auf.

3. **Client** Der Client ist eine Java GUI Anwendung, die für den Nutzer die Monitoringinformation darstellt.

Quellen: [44], [45]

5.3.1.4 CA Unicenter Network and System Monitoring (NSM)

Ebenso, wie die beiden zuvor vorgestellten Produkte bietet CA Unicenter NSM die Möglichkeiten Netzwerk, Server und Applikationen auf Infrastrukturebene zu überwachen. Wie Tivoli Monitoring und HP Operations Manager gibt es das Konzept der Agents. CA liefert zudem ein Agent Developer Toolkit um weitere Systeme anbinden zu können.

Im Unterschied zu HP Operations Manager Agents gibt es aber auf Ebene der Agents noch kein Filtering. Für das Filtering gibt es eine eigene Software Komponente, das DSM.

Quellen: [49], [40]

5.3.2 Grid Monitoring

Das Monitoring von Gridcomputing Architekturen ist eine Spezialdisziplin im Monitoring, da die verteilte Architektur eines Grid spezielle Herausforderungen an das Monitoring stellt. Die architektonische Nähe von Gridcomputing und serviceorientierten Architekturen macht es interessant den Markt bestehender Gridcomputing Monitoringsysteme zu betrachten.

An dieser Stelle werden die verbreitetsten Ansätze und Lösungen kurz vorgestellt, um für das Design einer Monitoringlösung für serviceorientierte Architekturen Ideen zu sammeln.

Das Global Grid Forum [12], hat für das Monitoring von Grids eine (Grid Monitoring Architecture, GMA) veröffentlicht. [75] Es beschreibt die drei Hauptkomponenten die für das Monitoring von Grids notwendig sind.

1. **Directory Service** Aufgabe des Directory Service ist es Information zur Eventpublikation für Producer und Consumer zu verwalten.

2. **Producer** Der Producer ist Quelle der Events. Er publiziert seine Statusinformationen in Form von Events.

3. **Consumer** Der Consumer ist ein Abnehmer dieser Events. Er nutzt diese Events für seine Aufgaben im Montoring.

5.3.2.1 R-GMA

R-GMA ist die Abkürzung für Relational Grid Monitoring Architecture. [43] Es ist eine Implementierung der GMA und wird beispielsweise vom europäischen Datagrid Projekt verwendet. Der bekannteste Nutzer des europäischen Data Grid ist der LHC des CERN, der es für die Datenanalyse seiner Experimente nutzt. [13]

Die Grundidee von R-GMA ist, es wie ein großes Datawarehouse wirken zu lassen. Deshalb basiert es auf einem relationalen Datenmodell. Als Sprache in der Producer und Consumer Informationen austauschen wurde SQL ausgewählt.

R-GMA unterscheidet beim Datenaustausch drei verschiedene Formen:

1. Historical Data: In diesem Fall wird ein Set an historischen Daten zurückgegeben.

2. Latest State: Beim Latest State wird bloß der aktuelle Zustand wiedergegeben.

3. Coninuous Stream: Hier wird der Latest State und alle zukünftigen Daten zurückgegeben.

Zur Abbildung der verschiedenen Formen des Datenaustauschs und der Nutzung von SQL als Sprache wird die GMA um zwei Elemente erweitert.

Das erste Element ist das Schema. Dieses Schema liefert das grundlegende Datenmodell. Das andere Element sind Agents. Ihre Aufgabe ist es Publisher und Consumer bei der Ausführung gewisser Tätigkeiten, wie zum Beispiel continuous queries zu unterstützen.

Ein weiteres Feature von R-GMA ist, dass es in MDS 5.3.2.3 integriert werden kann.

5.3.2.2 MonALISA

MonALISA ist ein Akronym und bedeutet MONitoring Agents using a Large Integrated Services Architecture. [70] Die Architektur von MonALISA besteht aus fünf Bestandteilen:

1. **Data Collection Engine** Die Data Collection Engine hat die Aufgabe Monitoringdaten zu sammeln. Sie arbeitet grundsätzlich mit dem SNMP (Simple Network Monitoring Protocol, RFC-1157) kann aber auch über einen Interface Mechanismus weitere Monitoringtools einbinden. Die Engine läuft multithreaded um so möglichst performant sehr viel Information sammeln zu können.

2. **Data Storage** Die gesammelten Daten werden lokal in eine relationalen Datenbank gespeichert. Diese Aufgabe übernimmt die zweite Komponente, das Data Storage. Neben der Speicherung übernimmt dieser Bestandteil auch noch die Indexierung und das Komprimieren alter Monitoringdaten um eine dauerhafte Leistungsfähigkeit sicherzustellen.

3. **Registration and Discovery** Registration and Discovery ist der dritte Baustein von MonALISA. Mittels des Protokolls JINI können sich Monitoringservices registrieren und gegenseitig finden. In den Services wird Information über die Gruppenzugehörigkeit gespeichert. Bei übereinstimmender Gruppenzugehörigkeit können die Services untereinander Daten austauschen und sich so updaten und replizieren. Auf diese Weise entstehen Netzwerke von Services.

4. **Predicates, Filters and Alarms** Sowohl realtime, als auch historische Daten können abgefragt werden. Der Mechanismus dazu sind Prädikate. Sie basieren auf Regular Expressions und ermöglichen den gezielten Datenzugriff. Mit diesem Mechanismus werden auch Filter definiert. Für Abfragen historischer Daten werden die Prädikate in SQL Statements übersetzt. Der gleiche Mechanismus wird auch für Alarme (Alerting) genutzt.

5. **Graphical Clients** Der letzte Teil von MonALISA ist das graphische Frontend. MonALISA wird standardmäßig mit einem Java GUI ausgeliefert. Hauptaufgabe ist die Konfiguration der einzelnen Monitoringservices und die Anzeige der Monitoringergebnisse.

5.3.2.3 Globus MDS

Das Globus Toolkit ist ein open source Werkzeug zum Erstellen von Grids. [14] Die zentrale Komponente zur Überwachung dieser Grids ist das Monitoring und Discovery System, MDS.

Das MDS basiert auf Webservices. Grundsätzlich besteht das System aus vier verschiedenen Komponenten.

1. **Index Service** Aufgabe des Index Service ist es Daten zu sammeln und zu indizieren um sie für etwaige Abfragen bereit zu haben. Als Abfragemechanismen werden sowohl Queries- als auch Subscriptions, also continuous queries unterstützt.

2. **Trigger Service** Aufgabe des Trigger Service ist ebenfalls die Datensammlung, doch bietet er die Möglichkeit basierend auf den gesammelten Daten Aktionen zu setzten.

3. **Aggregator Framework** Mithilfe des Aggregator Frameworks können Index und Trigger Services erstellt werden. Es bietet unterschiedliche Mechanismen der Datensammlung an. Einerseits können Daten in XML Format verarbeitet werden, zum anderen bietet das Framework Interfaces zu anderen Monitoring-Tools an.

4. **User Interface** Auch MDS bietet eine grafische Benutzeroberfläche an. Hauptaugenmerk wurde hier auf die Anzeige der Monitoringinformationen gelegt.

Ein Service zur Archivierung der Daten ist derzeit noch nicht vorhanden, aber für spätere Releases geplant.

5.3.3 Business Activity Monitoring

Business Activity Monitoring (BAM) nach der Definition von Gartner **provides real-time access to critical business performance indicators to improve the speed and effectiveness of business operations** [66]

Aus zwei Gründen ist die Analyse von Business Activity Monitoringsystemen interessant. Erstens, sie bilden ebenfalls wie das hier angestrebte Prozessmonitoring real-time Daten ab und zweitens, im Kern von BAM-Systemen werden oft Complex Event Processing Technologie eingesetzt.

Deshalb werden hier zwei typische Vertreter von BAM Software kurz vorgestellt.

5.3.3.1 Progress Apama

Progress vertreibt unter dem Namen Apama eine Business-Activity-Monitoringlösung. Der Name stammt von dem 2005 übernommenen gleichnamigen Unternehmen.

Apama besteht aus mehreren Grundbausteinen. [52]

1. **Correlation Engine** Die Correlation Engine ist das Herzstück von Apama. Sie ist eine in C++ geschriebene Rules Engine, die continuous queries auf Datenströmen ermöglicht. Sie hat die Aufgabe die Regeln, die mittels der Event Programming Language beschrieben wurden zu verarbeiten.

2. **Event Data Management** Event Data Management speichert die Events in eine relationale Datenbank, um eine Historisierung der Daten zu ermöglichen. Basierend auf diesen Daten können Back-Tests durchgeführt werden und Regeln geprüft werden.

3. **Event Programming Language (EPL)** Die Event Programming Language ist eine an Java erinnernde Sprache names MonitorSkript. Dabei handelt es sich dabei um eine prozedurale Sprache, im Gegensatz zu deklarativen Sprachen wie SQL. Mittels MonitorSkript werden Events und ihr Handling definiert. [65]

4. **EPL Tools** Um Benutzern die Arbeit mit der EPL zu erleichtern wurde der Apama Event Modeler entwickelt. Der Apama Event Modeler ist ein auf dem Eclipse Framework basierendes high-level Tool mit dem Monitors und Smart Blocks in Workflows arrangiert und kombiniert werden können. Monitors sind in EPL geschriebene kleinste Bausteine. Sie erkennen Daten und können Sie weitergeben. Smart Blocks sind wiederverwendbare Blöcke, die Business Logik abbilden.

5. **Real-Time Dashboards** Progress bietet Dashboards sowohl als Webfrontend als auch als Fat-Client an. Diese Dashboards werden mit dem Dashboard Studio im Rahmen des Erstellungsprozesses erstellt und geben ein visuelles Feedback.

Eine optionale Alerting Engine bietet die Möglichkeit Email, SMS oder sonstige Benachrichtigungsmechanismen zu definieren.

5.3.3.2 TIBCO Business Factor

TIBCO geht bei seiner Business-Activity-Monitoringlösung einen anderen Weg als Progress. Business Factor nutzt keine Complex Event Processing Engine für die Aggregation von Events, sondern eine In-Memory Datenbank. Ähnlich wie bei anderen Herstellern ist auch Tibcos Lösung komponentenbasiert. [53]

1. **Tibco Designer** Mit Hilfe des Designers werden Monitoring-Szenarios entwickelt. Der Designer basiert auf dem Tibco Designer, wie er von Tibcos EAI Lösung, Business Works genutzt wird. Herauszustreichen ist, dass es sich dabei um einen kompletten grafischen Editor handelt. Es ist nicht nötig Programmcode von Hand zu schreiben.

2. **Tibco Administrator** Der Adminstrator ist die Adminstrationskomponente in der Suite. Auch er wird in Tibcos EAI Lösung genutzt. Es handelt sich dabei um ein webbasiertes Tool mit dem Server verwaltet und Deployments gemanagt werden.

3. **Tibco Business Works** Business Works ist die EAI Lösung von Tibco. Durch die enorme Vielfältigkeit der Integrationsmöglichkeiten von Business Works können eine sehr große Zahl an Datenquellen angeschlossen werden.

4. **Tibco Business Factor Aggregator** Aufgabe des Aggregators ist es die verschiedenen Events zu aggregieren. Er nutzt dabei im Unterschied zu anderen Herstellern keine Complex Event Processing Technologie, sondern

setzt auf eine In-Memory-Datenbank. Das ist eine Datenbank wie zum Beispiel HSQLDB, die ausschließlich im Arbeitsspeicher läuft und deshalb keine Festplattenzugriffe für die Datenbankoperationen benötigt. Damit wird eines der größten Bottlenecks von klassischen Datenbanken beseitigt.

5. **Tibco Business Factor Server** Im Business Factor Server, werden die designten Modelle gespeichert, aber auch Alerting Regeln verwaltet.

6. **Tibco Business BAM Desktop** Der BAM Desktop ist das grafische Frontend der Lösung. Er ermöglicht die grafische Darstellung von Monitoringinformationen. Er ist komplett anpassbar und ermöglicht so personalisierte Ansichten für den Nutzer.

Zu erwähnen ist noch, dass Tibco unter dem Namen Business Events eine Complex Event Processing Lösung vertreibt.

Kapitel 6

CEP als Basis für Prozessmonitoring

In den Kapiteln 4 und 5 wurden Problembereiche und Anforderungen des Monitorings vorgestellt. In diesem Kapitel werden diese der Definition für Complex Event Processing von David Luckham gegenübergestellt. Durch diese Gegenüberstellung wird dargestellt, warum Complex Event Processing Technologie als Grundlage für Prozessmonitoring in serviceorientierten Architekturen in Frage kommt und so der Ansatz für eine Monitoringlösung herausgearbeitet.

6.1 Gründe für Complex Event Processing als Basis

David Luckham beschreibt Complex Event Processing als *The goal of CEP is to enable the information contained in the events flowing through all of the layers of the enterprise IT infrastructure to be discovered, understood in terms of its impact on high level management goals and business processes, and acted upon in real time.* [8]

- **Events flowing through all layers of the enterprise IT**

  Im Punkt 2.2.3 wurde bereits dargelegt, dass Monitoring auf verschiedenen Ebenen notwendig ist. Von Bestandteilen der Infrastruktur, wie etwa dem Netzwerk bis zur Überwachung von Business Rules, wie zum Beispiel Schranken für Risikolevels muss sich Prozessmonitoring erstrecken. Die Definition von Complex Event Processing erstreckt sich auf diese Bereiche.

- **Discovered**

Eine Grundvoraussetzung für CEP ist das Auffinden der Events. Das selbe gilt auch für Monitoring. Wie im Abschnitt 5.1.1 bereits ausgeführt ist auch hier das Auffinden der Informationen ein notwendiger Schritt dazu.

- **Understood in terms of high level management goals and business processes**

  Eine der wichtigsten Übereinstimmungen zwischen CEP und Prozessmonitoring ist der Fokus auf Prozesse und höherwertige Business Anforderungen. Prozessmonitoring erfordert ab der dritten Ebene (Business Layer) Kenntnis der geschäftlichen Zusammenhänge um Monitoring durchführen zu können. Dieser Umstand wurde in Abschnitt 5.1.5 erläutert.

- **Acted upon in real time**

  Geschwindigkeit in der Verarbeitung ist auch beim Prozessmonitoring eine Grundvoraussetzung. Wie im Punkt Echtzeit 5.1.2 bereits ausgeführt.

Anhand dieser Gegenüberstellung lässt sich gut erkennen, dass Complex Event Processing eine aussichtsreiche Technologie darstellt mit der ein Prozessmonitoring-System entwickelt werden kann.

Wie oben ausgeführt berührt die Definition von Complex Event Processing bereits sehr viele der Herausforderungen von Prozessmonitoring in serviceorientierten Architekturen. Neben diesen Punkten gibt es aber noch weitere Punkte, die Complex Event Processing für das Monitoring serviceorientierter Architekturen interessant machen.

- **Complex Events** Ein bereits betrachteter Bereich ist der der Orchestrierung. Services können zu umfangreicheren Services kombiniert werden (vgl. dazu Abschnitt 4.3). CEP bietet durch die Möglichkeit Events zu umfangreicheren Complex Events zu kombinieren eine ähnliche Funktionalität. Dies ist ein weiterer Grund warum sich CEP als Basis für ein Prozessmonitoring in serviceorientierten Architekturen empfiehlt.

- **Rules** Grundlage von Complex Event Technologie ist immer eine Sprache, mit der Events beschrieben werden können. Häufig geschieht das in Form einer Regelsprache. Genau diese Sprache ist aber auch für Monitoring interessant. Speziell im Business Layer ist es notwendig recht komplexe Sachverhalte abzubilden. Dazu ist eine Regelsprache gut geeignet.

Kapitel 7

Konzeption

In diesem Kapitel wird die grundsätzliche Konzeption eines Monitoringsystems für Prozessmonitoring in serviceorientierten Architekturen beschrieben. Dazu werden zunächst verwandte Konzepte vorgestellt und basierend darauf eine eigene Architektur für ein System zum Monitoring vor Prozessen in serviceorientierten Architekturen entwickelt.

7.1 Architektur

7.1.1 Serviceorientierte Architektur

Für die Umsetzung einer servierorientierten Architektur gibt es unterschiedliche Konzepte. Eines der verbreitetsten entspricht dem von Sun Microsystems. [69] Dieses wird im folgenden Kapitel kurz vorgestellt. Im Anschluss daran wird kurz untersucht ob es als Architektur für Prozessmonitoring geeignet ist.

7.1.1.1 Überblick

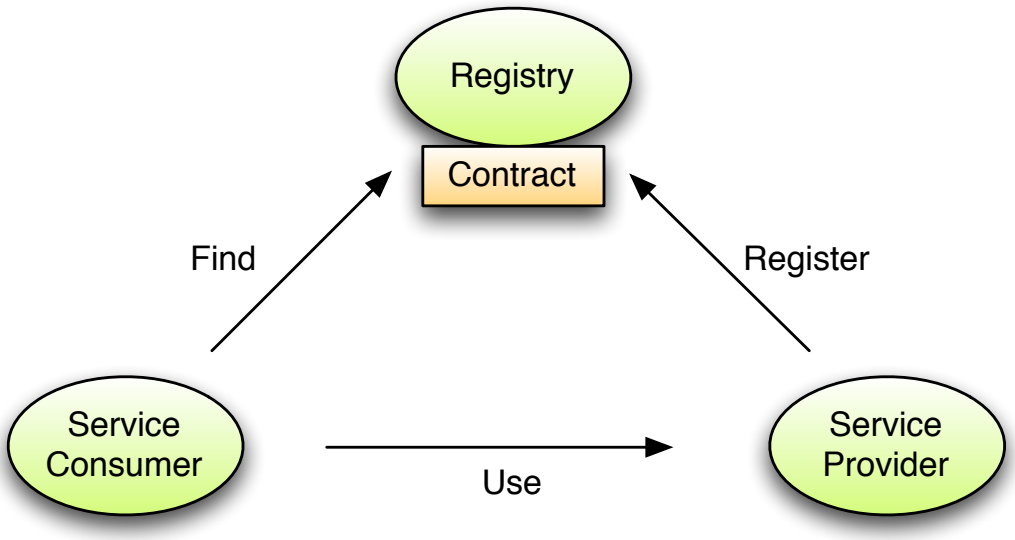

Abbildung 7.1: Serviceorientierte Architektur nach Sun Microsystems

7.1.1.2 Bestandteile der SOA

Die serviceorientierte Architektur besteht aus vier Hauptbestandteilen.

1. **Service Provider**

   Der Service Provider ist der Anbieter eines Service. Er definiert Aussehen und Schnittstelle des Service und stellt es zur Verfügung. Für jedes Service, das er anbietet schreibt er einen Service Contract. Damit wird das Service von ihm bei der Service Registry registriert.

2. **Service Contract**

   Der Service Contract ist eine standardisierte, formale Beschreibung des Service. Er enthält eine Beschreibung wie der Dienst des Service Providers zu nutzen ist. Darin werden zum Beispiel Parameter, wie Ein- und Ausgabewerte oder die Adresse des Service definiert.

3. **Service Registry**

   Die Service Registry ist ein Verzeichnis, in dem ähnlich einem Telefonbuch Services registriert sind. Es ist Schnittstelle zwischen potentiellem Service

Konsumenten und Service Anbieter. Konsumenten können das Verzeichnis der Registry nach passenden Services durchsuchen und die Service Contracts einsehen.

4. **Service Consumer**

   Der Service Consumer ist der Nutzer des Service. Er kann basierend auf dem Service Contract das Service Nutzen.

7.1.1.3 Anwendbarkeit für Monitoring

Theoretisch wäre es möglich das Monitoring-System als serviceorientierte Architektur aufzubauen. Im Rahmen dieser Arbeit wurde aber darauf verzichtet, da dafür eine Service Registry und Service Contracts nötig gewesen wären. Dies hätte einen zusätzlichen Overhead erzeugt, der über eine prototypische Implementierung hinausgegangen wäre und dabei aber keinen Mehrwert erzeugt hätte.

7.1.2 EAI Architektur

Eine weitere verbreitete Architektur ist die EAI- oder Service-Bus-Architektur. EAI bedeutet Enterprise Application Integration. Grundlage dieser Architektur ist ein Enterprise Service Bus mit dem unterschiedliche Applikationen verbunden werden können. Er ist die zentrale Drehscheibe für den Datenverkehr zwischen verschiedenen Anwendungen.

In der EAI Architektur findet der Datenaustausch über einzelne Nachrichten (Messages) statt, die zwischen den Applikationen ausgetauscht werden. Deshalb spricht man in diesem Fall auch häufig von *Message Oriented Middleware*.

7.1.2.1 Überblick

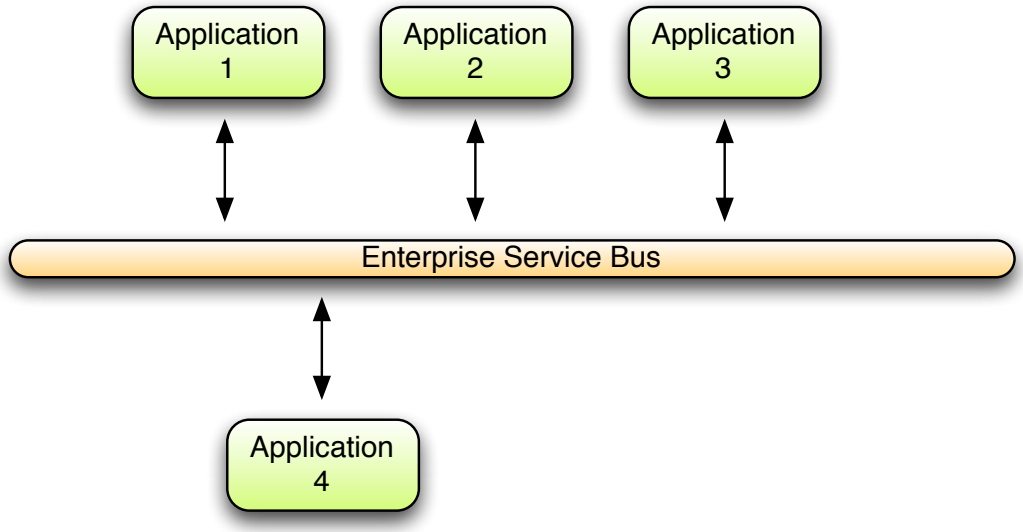

Abbildung 7.2: EAI Architektur

7.1.2.2 Bestandteile EAI Architektur

Die EAI Architektur ist sehr einfach aufgebaut. Sie besteht im wesentlichen aus zwei Hauptkomponenten.

1. **Enterprise Service Bus (ESB)**

   Grundidee eines ESB ist es die vielen Punkt zu Punkt Verbindungen, die zwischen unterschiedlichen Applikationen existieren abzulösen und durch ein einheitliches Übertragungsinterface zu ersetzten. Dieses Interface ist der sogenannte ESB. Seine Aufgabe ist es den zuverlässigen Datentransfer zwischen verschiedenen Applikationen sicherzustellen. Als Basis dafür dienen Messaging Technologien wie zum Beispiel JMS oder Rendezvous.

2. **Applikation**

   Der zweite Baustein sind die Applikationen selbst. Sie werden mit Hilfe von Middleware an den ESB angeschlossen, wo sie Nachrichten versenden und empfangen können.

7.1.2.3 Eignung für Monitoring

Der Messaging Ansatz der EAI Architektur ist vielversprechend. Monitoringinformationen können von den Services auf einen ESB publiziert werden. Dort kann dann die Verarbeitungskomponente diese entgegennehmen und verarbeiten.

Durch diese Architektur können mehrere Anforderungen an Monitoring auf einmal erfüllt werden:

1. **Geschwindigkeit**

   Aktuelle Systeme bieten sehr hohe Übertragungsraten und Volumen an. Der open source JMS Server Apache ActiveMQ kann beispielsweise 20.000 Nachrichten pro Sekunde verarbeiten. [15] Durch den Einsatz eines Messaging-Systems ist eine schnelle und zuverlässige Datenübertragung möglich.

2. **Erweiterbarkeit**

   Durch den Einsatz der EAI-Architektur können beliebig viele Datenquellen Monitoringdaten liefern. Jedes Service kann seine Daten auf den ESB publizieren. Es gibt keine direkte Kopplung zwischen Datenquelle und Datenverarbeitung.

   Soll ein weiteres Service ins Monitoring aufgenommen werden, so ist auf Seite der Verarbeitungskomponente, die die Nachrichten auswertet, keine Anpassung nötig. Sie *horcht* lediglich auf den ESB und nimmt alle Nachrichten entgegen, die dort publiziert werden.

3. **Keine Datenspeicherung (in Verarbeitungskomponente)**

   Durch den Einsatz eine ESB ist es auch möglich die Verarbeitungskomponente von der Historisierungskomponente vollständig zu trennen. Um dies zu ermöglichen, muss der ESB so konfiguriert sein, dass die Monitoring-Informationen nach der Verarbeitung erhalten bleiben. Sie können im ESB vorgehalten werden, bis sie von der Historisierungskomponente verarbeitet worden sind.

7.1.3 Grid Monitoring Architecture

Das Open Grid Forum [16] definierte 2002 eine Referenz-Architektur für das Monitoring von Grids. [75] Da Grids ebenso wie serviceorientierte Architekturen sehr verteilt sind lohnt es sich diese Architektur zu betrachten.

7.1.3.1 Überblick

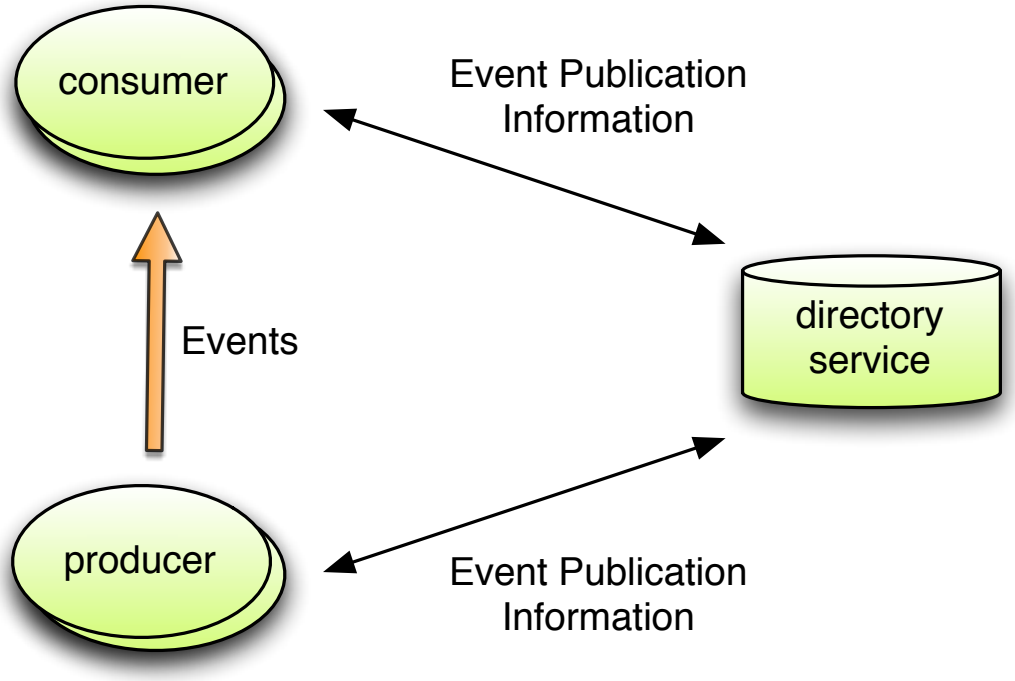

Abbildung 7.3: Grid Monitoring Architektur

7.1.3.2 Bestandteile der Grid Monitoring Architecture

Die Grid Monitoring Architecture besteht aus drei Teilen und erinnert in ihrer Gestalt stark an die in 7.1.1 vorgestellte SOA Architektur. Auch hier gibt es Producer, Consumer und eine Registry, die hier Directory Service genannt wird.

- **Producer**

  Ein Producer ist eine Quelle der Monitoringinformationen. Er emittiert Monitoringinformation, die von einem Consumer verarbeitet werden können. Der Standard definiert zwei Arten der Datenlieferung. Zum einen als kontinuierlicher Datenstrom und zum anderen auf Request-Basis.

- **Consumer**

  Der Consumer stellt den Gegenpart des Producers dar. Er verarbeitet die Monitoringinformationen die Producer entsenden. Dabei kann er gezielt An-

fragen an einen Producer senden, oder einfach einen Datenstrom abonnieren.

- **Directory Service**

  Aufgabe des Directory Service ist es Producern und Consumern eine Plattform zu bieten, wo sie sich finden können. Die Funktionsweise des Directory Service entspricht dem der Service Registry: Producer können sich registrieren und geben an welche Daten sie in welcher Form versenden können. Consumer können das Directory Service durchsuchen um zu sehen, welche Daten angeboten werden.

7.1.3.3 Anwendbarkeit für Monitoring

Die grundsätzliche Architektur der Grid Monitoring Architecture ist geeignet für Service Monitoring. Services können als Producer aufgefasst werden und die Verarbeitungskomponente als Consumer.

Auf das Directory Service kann verzichtet werden, da der hier vorgestellte Ansatz von Prozessmonitoring gewisse Gegebenheiten voraussetzt.

- **Verarbeitungskomponente ist verfügbar**

  Der Consumer ist an einer bestimmten Stelle durchgehend verfügbar. Ein *Kommen und Gehen* von Komponenten, wie in einem Grid Grundannahme wird hier explizit ausgenommen.

- **Eventformat vereinheitlicht**

  Durch die einheitliche Gestaltung des Event Formats ist es nicht nötig, dass ein Dienst Informationen über Datenformate hält und die Rolle eines Vermittlers einnimmt.

7.1.4 Architektur Prozessmonitoring-System für serviceorientierte Architekturen

Im folgenden Kapitel wird die grundlegende Architektur eines Systems für Prozessmonitoring in serviceorientierten Architekturen vorgestellt. Architektonisch ist sie an die EAI- und an die Grid Monitoring Architektur angelehnt. Aus der EAI Architektur wird der Service Bus übernommen. Er ist die zentrale Datenpipeline wo alle Monitoringevents durchlaufen. Die Grid Monitoring Architektur spendet das Producer/Consumer Konzept. Einzelne Services können als Producer, die Rules Engine und der Data Store als Consumer angesehen werden.

Die hier konzipierte Architektur besteht aus sieben grundlegenden Elementen, die im folgenden einzeln beschrieben werden.

7.1.4.1 Überblick

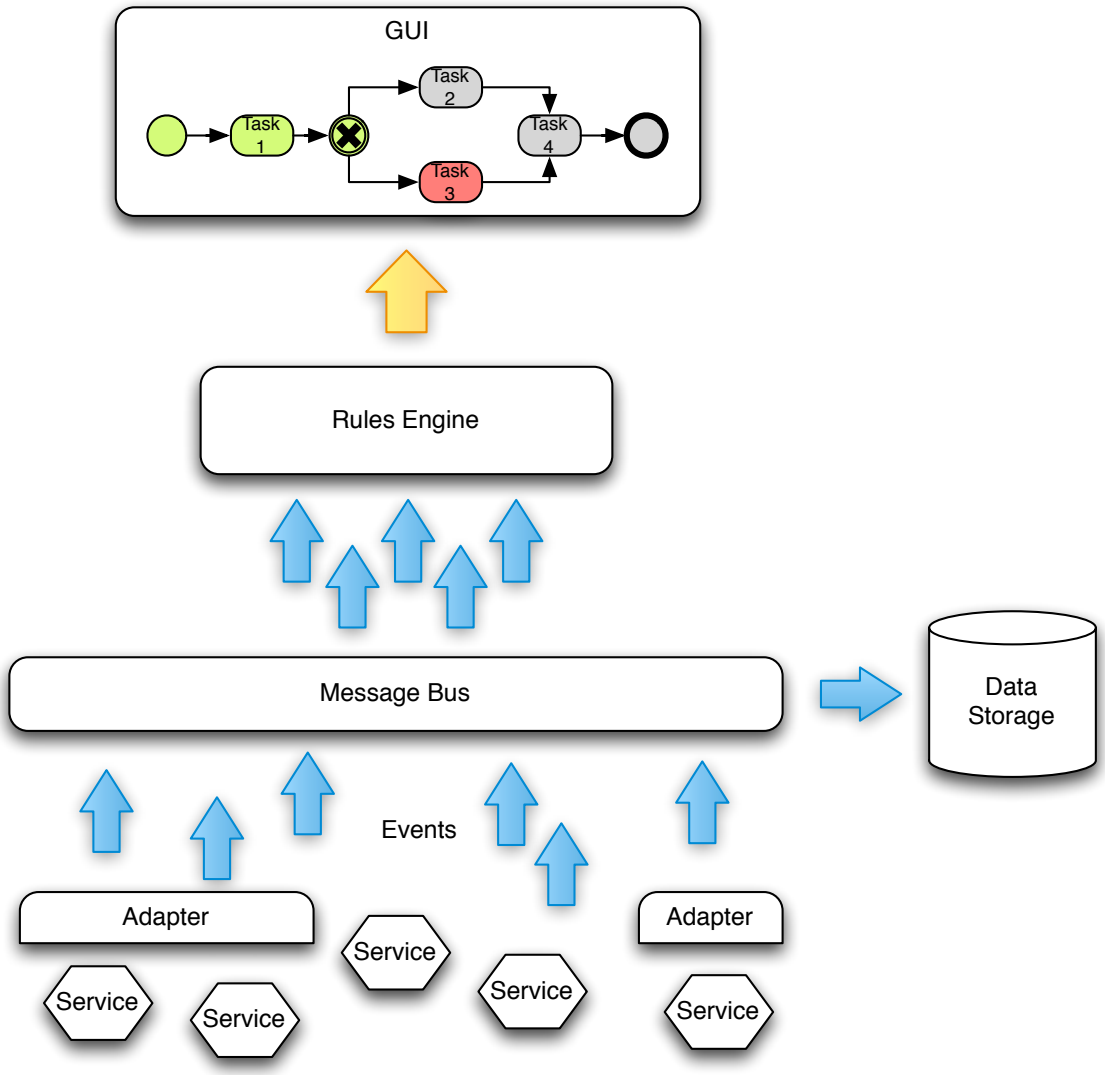

Abbildung 7.4: Architektur Prozessmonitoring-System

7.1.4.2 Service

Erster Grundbaustein eines Monitoringsystems für serviceorientierte Architekturen ist das Service selbst. In dem hier entwickelten Ansatz liefern die einzelnen Services die Grundlage des Monitorings in Form von Events.

Es wird an dieser Stelle keine Entscheidung über das Design der Services getroffen. Ganz im Gegenteil, die angestrebte Lösung ist unabhängig von den eigentlichen Services. Einzige Bedingung an ein Service ist, dass es Output produziert. Zwei Arten von Output können hier unterschieden werden.

1. **Fehlermeldungen**

   Um verfolgen zu können ob ein Service korrekt arbeitet, ist diese Information von großer Wichtigkeit. Liefert ein Service im Fehlerfall überhaupt keinen Output, so kann dies nicht überwacht werden.

2. **Ergebnisse einer erfolgreichen Serviceoperation**

   Speziell wenn geschäftsrelevante Vorgänge überwacht werden sollen, ist es nötig auch die Ergebnisse der Serviceoperationen auszuwerten. Deshalb muss ein Service auch diese Outputs anbieten.

Weitere Anforderungen an das Design, oder die Realisierung der Services gibt es nicht.

7.1.4.3 Adapter

Der zweite Baustein der Architektur sind Adapter. Durch dieses Konzept wird ein Problem, das in verteilten Architekturen auftritt gelöst. Es handelt sich dabei um das Problem der unterschiedlichen Träger der Monitoringinformationen, wie es in 4.1.2 vorgestellt wurde.

Der Ansatz dabei ist simpel. Aufgabe eines Adapters ist es sämtliche monitoring-relevante Information eines Service in ein einheitliches Format zu bringen und als Event zu publizieren.

7.1.4.4 Message Bus

Ein weiterer Teil der Architektur ist der Message Bus. Aufgabe des Message Bus ist es einen zentralen Backbone bereit zu stellen, der schnell und hoch verfügbar Events der einzelnen Services zur Verarbeitungskomponente und zum Datastore transportiert.

Durch den Einsatz eines Message Bus können drei der zuvor behandelten Probleme gelöst werden.

1. **Verteilung der Datenquellen**

   In 4.1 wurde erörtert, wie und warum Quelldaten verteilt sind. Durch den Einsatz eines Bussystems, an das die einzelnen Datenquellen ihre Nachrichten schicken gibt es eine unternehmensweit einheitliche Art des Datentransports. Dies erleichtert das hinzufügen und auslesen von Events.

2. **Verlässlichkeit**

   Speziell beim Monitoring ist es wichtig, dass keine Nachrichten verlorengehen. Nur wenn dies sichergestellt ist, ist es möglich lückenlos Prozesse zu überwachen. Aktuelle Implementierungen von Message Bussen bieten genau das. Tibco EMS bietet die Option des *reliable messaging* [17], ebenso wie beispielsweise Sun 's Implementierung des JMS Protokolls [18], andere Hersteller bieten vergleichbares an.

3. **Geschwindigtkeit**

   Eine weitere wichtige Anforderung ist die Geschwindigkeit des Datentransports. Siehe dazu Abschnitt 5.1.2. Aktuelle Messaging-Systeme erreichen bereits erstaunliche Durchsatzraten. FioranoMQ erreicht beispielsweise Raten von über 90.000 Nachrichten pro Sekunde. [19] [20]

Typische Vertreter sind unter anderen Tibco EMS [17], Apache ActiveMQ [21], Sonic ESB [22] oder IBM WebSphere MQ, besser bekannt unter MQ Series [23]

7.1.4.5 Datenspeicher

Weitere Anforderungen an ein Monitoringsystem sind Nachvollziehbarkeit und die Bereitstellung der Daten für Analysezwecke. Aus diesem Grund ist es nötig Daten zu historisieren. Im Kapitel 5.1.7 wurde dies bereits erörtert.

Diese Anforderung wird durch den Einsatz eines Datastore erfüllt. Dieser kann durch eine einfache relationale Datenbank umgesetzt werden. Die Komplexität dabei ist nicht sehr hoch, da alle Events die gleiche Struktur haben und so abgelegt werden. Die Geschwindigkeit des Datastore ist ebenfalls keine Anforderung höchster Priorität, da die Daten nicht in near realtime in der Datenbank verfügbar sein müssen. Es reicht, wenn sie nach und nach vom Message Bus abgeholt werden und im Datastore abgelegt werden.

Die verbreitetsten Datenbank-Mangementsysteme (DBMS) sind Oracle [24], MS SQL Server [25], IBM DB2 [26] sowie die freien Alternativen MySQL [27] und PostgresSQL [28]

7.1.4.6 Rules Engine

Die Rules Engine ist das Kernstück der Monitoringlösung. Durch den Einsatz einer Rules Engine können viele verschiedene Probleme und Anforderungen gelöst werden. Die Detailanforderungen wurden im Kapitel 5.2 besprochen. Der Einsatz einer Complex Event Processing Engine erfüllt folgende Anforderungen.

- Hohe Geschwindigkeit bei der Verarbeitung der Events
- Abbildung von Business Anforderungen
- Abbildung von Prozessfluss
- Behandlung von Ausnahmen
- Filtern von Events aus dem Datenstrom
- Aggregation von einzelnen Events

Der Markt an Complex Event Processing Engines ist überschaubar. Zu den verbreitetsten Engines zählen Progress Apama [29], Aleri CEP [30], UC4 Decision [31], Tibco Business Events [32]. Im Bereich Opensource-Software existiert nur ein Vertreter, Esper. [33]

7.1.4.7 Grafische Benutzerschnittstelle

Eine der Anforderungen an ein Prozessmonitoring-System ist die übersichtliche visuelle Darstellung der Prozesse und ihres aktuellen Status (vgl. Abschnitt 5.1.8).

Dieses grafische Interface kann entweder browserbasiert, oder als vollständige Applikation umgesetzt werden. Da hier live Updates für aktuellste Statusmeldungen unterstützt werden sollen, wird eine eigenständige Applikation als GUI dienen.

7.2 Datenmodell Event

Der Event ist die Basisinformation für jegliches Monitoring. Für eine schnelle und einfache Verarbeitung ist es nötig ein einheitliches Format für alle Events sämtlicher Services zu finden. Dieses Datenformat muss flexibel genug sein um alle Nachrichten transportieren zu können und dabei so einfach wie möglich gestaltet sein um eine simple Handhabung zu ermöglichen.

Im Rahmen dieser Arbeit wurde ein solches Datenformat entwickelt. Dieses wird en detail in den folgenden Abschnitten erklärt.

7.2.1 Gründe für XML als Datenformat

Für die Übertragung der Daten wurde als Format XML gewählt. Dies hat mehrere Gründe:

1. **Textbasiert**

   XML ist ein textbasiertes Format. Der Vorteil von textbasierten Datenformaten ist, dass sie einfach zu erstellen sind. Es sind keine komplexen Algorithmen nötig um ein Textfile zu erstellen oder auszulesen. Entsprechend einfach ist es für die Services Events zu erzeugen.

2. **Lesbarkeit**

   XML ist eine Sprache die maschinell sehr gut verarbeitet werden kann. Am Markt ist eine Vielzahl von Libraries und Tools erhältlich, die den Umgang mit XML Dokumenten erleichtern. Der entscheidende Vorteil ist aber, dass XML genauso gut von Menschen gelesen werden kann. Jedes Event kann von Menschen ohne spezielle Dekodierung gelesen und verstanden werden.

3. **Verbreitung**

   Neben den genannten Vorteilen muss auch herausgestrichen werden, dass die Datenübertragung im XML Format stark verbreitet ist und sich bereits als Quasi-Standard etabliert hat.

7.2.2 XML Schema des Datenmodells

7.2.2.1 Event XML Schema graphisch

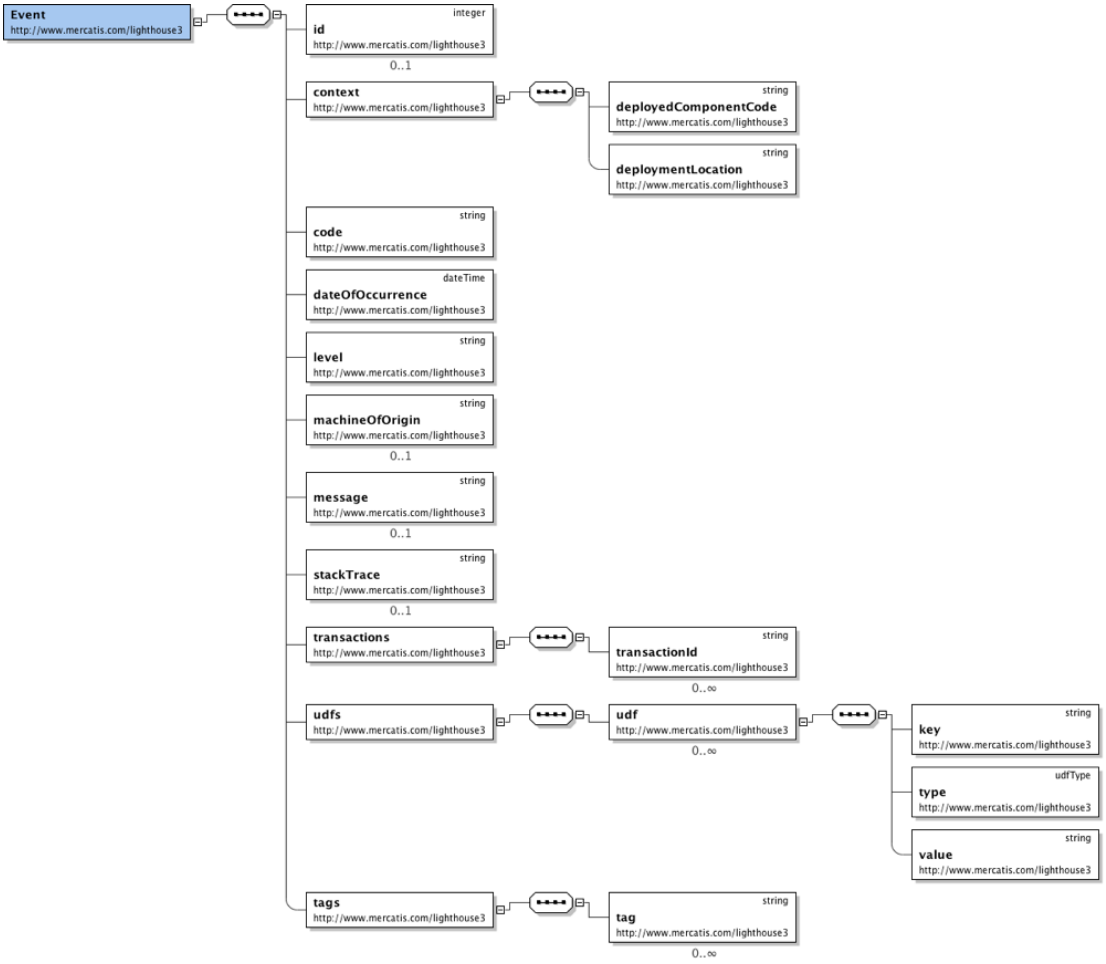

Abbildung 7.5: XML Schema Event

7.2.2.2 Event XML Schema Definition

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified" targetNamespace="http://www.
      mercatis.com/lighthouse3" xmlns:l="http://www.mercatis.com/
      lighthouse3">
3   <xs:simpleType name="udfType">
```

```
4     <xs:restriction base="xs:string">
5       <xs:enumeration value="dateTime"/>
6       <xs:enumeration value="binary"/>
7       <xs:enumeration value="string"/>
8       <xs:enumeration value="float"/>
9       <xs:enumeration value="double"/>
10      <xs:enumeration value="integer"/>
11      <xs:enumeration value="long"/>
12      <xs:enumeration value="boolean"/>
13    </xs:restriction>
14  </xs:simpleType>
15  <xs:element name="Event">
16    <xs:complexType>
17      <xs:sequence>
18        <xs:element ref="l:id" minOccurs="0"/>
19        <xs:element ref="l:context"/>
20        <xs:element ref="l:code"/>
21        <xs:element ref="l:dateOfOccurrence"/>
22        <xs:element ref="l:level"/>
23        <xs:element ref="l:machineOfOrigin" minOccurs="0"/>
24        <xs:element ref="l:message" minOccurs="0"/>
25        <xs:element ref="l:stackTrace" minOccurs="0"/>
26        <xs:element ref="l:transactions"/>
27        <xs:element ref="l:udfs"/>
28        <xs:element ref="l:tags"/>
29      </xs:sequence>
30    </xs:complexType>
31  </xs:element>
32  <xs:element name="id" type="xs:integer"/>
33  <xs:element name="context">
34    <xs:complexType>
35      <xs:sequence>
36        <xs:element ref="l:deployedComponentCode"/>
37        <xs:element ref="l:deploymentLocation"/>
38      </xs:sequence>
39    </xs:complexType>
40  </xs:element>
41  <xs:element name="deployedComponentCode" type="xs:string"/>
42  <xs:element name="deploymentLocation" type="xs:string"/>
43  <xs:element name="code" type="xs:string"/>
44  <xs:element name="dateOfOccurrence" type="xs:dateTime"/>
45  <xs:element name="level" type="xs:string"/>
46  <xs:element name="machineOfOrigin" type="xs:string"/>
47  <xs:element name="message" type="xs:string"/>
48  <xs:element name="stackTrace" type="xs:string"/>
49  <xs:element name="transactions">
50    <xs:complexType>
```

```
51        <xs:sequence>
52          <xs:element minOccurs="0" maxOccurs="unbounded" ref="
                l:transactionId"/>
53        </xs:sequence>
54      </xs:complexType>
55    </xs:element>
56    <xs:element name="transactionId" type="xs:string"/>
57    <xs:element name="udfs">
58      <xs:complexType>
59        <xs:sequence>
60          <xs:element minOccurs="0" maxOccurs="unbounded" ref="l:udf"/
                >
61        </xs:sequence>
62      </xs:complexType>
63    </xs:element>
64    <xs:element name="udf">
65      <xs:complexType>
66        <xs:sequence>
67          <xs:element ref="l:key"/>
68          <xs:element ref="l:type"/>
69          <xs:element ref="l:value"/>
70        </xs:sequence>
71      </xs:complexType>
72    </xs:element>
73    <xs:element name="key" type="xs:string"/>
74    <xs:element name="type" type="l:udfType"/>
75    <xs:element name="value" type="xs:string"/>
76    <xs:element name="tags">
77      <xs:complexType>
78        <xs:sequence>
79          <xs:element minOccurs="0" maxOccurs="unbounded" ref="l:tag"/
                >
80        </xs:sequence>
81      </xs:complexType>
82    </xs:element>
83    <xs:element name="tag" type="xs:string"/>
84 </xs:schema>
```

7.2.3 Event

Ein Event ist ein komplexer Datentyp, der aus verschiedenen anderen Datenfeldern besteht. Die Aufgaben der einzelnen Datenfelder werden in den folgenden Absätzen erläutert.

7.2.3.1 id

Jedes Event hat ein eindeutige ID. Dies ist eine möglichst zufällige, große Zahlenfolge, die automatisch erzeugt wird. Ziel ist es jedes Event absolut eindeutig identifizieren zu können.

7.2.3.2 context

Das Feld Kontext ist wieder ein komplexes Feld. Es besteht aus zwei weiteren Datenfeldern. Aufgabe des Context-Felds ist es die Quelle des Events zu identifizieren. Dazu gibt es zwei Felder:

1. **deployedComponentCode**

   Jedes Service muss in einer Umgebung deployt sein und dort laufen um seine Dienste anzubieten. Häufig ist es so, dass mehrere Services in einer Umgebung laufen. Um zu unterscheiden welches Service das Event erzeugt hat, wird in diesem Feld der Name des Service gespeichert.

2. **deploymentLocation**

   Nicht immer ist der Name des Service eindeutig. Oftmals laufen gleichnamige Services in verschiedenen Umgebungen, wie zum Beispiel einer Test-, einer Integrations- und einer Produktiv Umgebung. Solche Szenarios werden durch dieses Feld abgebildet.

7.2.3.3 code

Der Code ist eine frei wählbare Kurzbezeichnung des Events. Beispiele dafür sind zum Beispiel **Filesystem_OK**, oder **FX_Intraday_Risk_Level**. Sinn des Codes ist es Events möglichst einfach zu beschreiben.

7.2.3.4 dateOfOccurrence

Jedes Event tritt zu einem bestimmten Zeitpunkt auf. Dieser Zeitpunkt wird in dem Feld **dateOfOcurrence** gespeichert.

7.2.3.5 level

Nicht alle Events sind automatisch kritische Events. Um kritische Events von unkritischen Events unterscheiden zu können, wird jedem Event ein Level zugeordnet. Sechs verschiedene Level werden hier genutzt. Dabei wurde weitestgehend das Konzept der Loglevel, wie es in Java genutzt wird [34] übernommen.

1. FATAL (höchste Priorität)
2. ERROR
3. WARNING
4. INFO
5. DETAIL
6. DEBUG (geringste Priorität)

Von FATAL bis FINEST geht die Abstufung der einzelnen Level. FATAL ist die höchste Prioritätsstufe. Events mit diesem Level sind typischerweise Meldungen, die den Prozessfluss unterbrechen, oder ein Business Ziel verfehlen lassen. Am anderen Ende der Skala befindet sich das Level DEBUG. Es ist gedacht um zusätzliche Informationen, die für die Fehlerverfolgung oder ähnliche Tätigkeiten hilfreich sind, zu identifizieren.

7.2.3.6 machineOfOrigin

Das Feld **machineOfOrigin** bezeichnet die Maschine wo die Nachricht entstanden ist. Im Gegensatz zum Feld **deploymentLocation**, das frei für Kontextinformation genutzt werden kann, ist hier der physische Servernamen des Rechners anzuwenden.

Hintergrund dafür ist, dass in größeren Installationen technische Services zumeist geclustert werden um die Ausfallsicherheit und die Geschwindigkeit zu erhöhen. Um Probleme, die in geclusterten Services auftreten dem korrekten Node zuordnen zu können, ist es nötig den exakten Namen des Servers zu kennen.

7.2.3.7 message

Im Feld **message** ist die eigentliche Meldung gespeichert. Dieses Feld kann frei genutzt werden um natürlichsprachlich das Event zu beschreiben.

7.2.3.8 stackTrace

Der **Stacktrace** ist für technische Fehlermeldungen gedacht. Er zeigt die Abfolge der Aufrufe im Programmcode durch die ein Fehler durchgereicht wurde um so die Wurzel des Problems erkennen zu können.

7.2.3.9 transactions

Um Prozessinstanzen abbilden zu können, ist es nötig die einzelnen Nachrichten zu kennzeichnen. Durch eine kontextuelle Kennzeichnung, wie etwa durch eine Transaction-ID oder durch eine Artikelnummer, ist es möglich Events einem bestimmten Prozessdurchlauf, also einer Prozessinstanz zuzuordnen. Da nicht alle Systeme die gleichen Transaktionscodes erzeugen können ist es hier möglich mehrere dieser Transaction-IDs zu speichern.

Soll zum Beispiel eine Bestellung in einem Online-Shop verfolgt werden ist es möglich im ersten Schritt die Session-ID der Websession, eine Kunden-ID und die Artikelnummer als Transaction-IDs zu speichern. In anderen Services, die etwa das Billing durchführen ist dann vielleicht nur mehr die Kunden-ID verfügbar. Die Schnittmenge der Transaction-IDs erlaubt es aber dann noch immer den Prozessfluss in einer konkreten Instanz nachzuvollziehen.

Dieser Ansatz ist selbstverständlich ein *best effort* Ansatz. Es ist nur möglich Transaktionen nachzuvollziehen, solange es zwischen Systemen gemeinsame Transaktions-Codes gibt.

7.2.3.10 udfs

Ein ähnliches Konzept, wie *transactions* bieten die **User Defined Fields (UDF)** . Unter dem Konten *udfs* ist es ebenso wie unter dem Konten *transactions* möglich eine Liste von unterschiedlichen *UDFs* anzulegen.

In einem solchen User Defined Field können beliebige Informationen typisiert abgelegt werden. Ziel davon ist es Nutzdaten, Beschreibung und Typ so zu trennen, dass eine spätere Auswertung so leicht wie möglich gemacht wird.

Ein UDF besteht aus drei einzelnen Feldern.

1. **key**

   Im Feld *key* wird der Name des User Defined Fields gespeichert.

2. **type**

   Im Typ wird der Datentyp des UDFs festgehalten. Unterstützt werden nur einfache Datentypen, die auch in Datentypen von relationalen Datenbanken übersetzt werden können, damit Auswertungen erleichtert werden. Unterstützt werden die einfachen Datentypen: dateTime, string, float, double, integer und binary.

3. **value**

Im Feld *value* werden dann die eigentlichen Nutzdaten gespeichert. Dies kann, entsprechend der Typdefinition ein Datum (dateTime), ein Text (string) oder sogar ein Excel File (binary) sein.

7.2.3.11 tags

Unter dem Konten **tags** können beliebig viele einzelne *tag* Felder angefügt werden. Grundidee dieser Felder ist es eine weitere Möglichkeit zu bieten Nachrichten logischen Einheiten zuzuordnen. Grundsätzlich wäre dies auch durch den Einsatz von *UDFs* möglich, doch dabei ist ein zusätzlicher Aufwand durch die Typisierung der Felder notwendig. Ist keine Typisierung nötig, so können *tags* genutzt werden um Nachrichten zu kategorisieren.

Selbstverständlich ist es nicht nötig immer alle Felder zu nutzen. Zwingend vorgeschrieben sind lediglich folgende vier Felder:

1. code
2. context (deployedComponentCode, deploymentLocation)
3. dateOfOccurrence
4. setLevel

7.2.4 Datenbank Schema des Datenmodells

Zur Historisierung der Daten ist es nötig die Events in einer relationalen Datenbank zu speichern. In den folgenden Abschnitten wird das Datenbank Schema definiert, das für die Speicherung dieser Events entwickelt wurde.

7.2.4.1 ER Diagramm

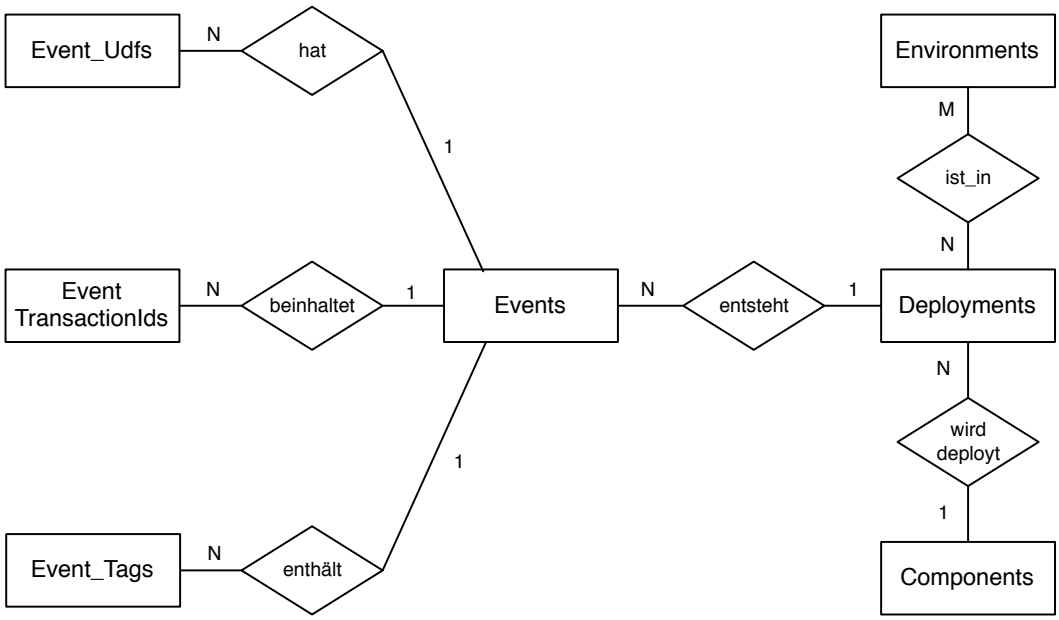

Abbildung 7.6: ER Diagram Datastore

7.2.4.2 Entitäten

- **EVENTS**

  Im Table *Events* werden alle Events gespeichert. Ein Event kann beliebig viele UDFs, Transaction IDs und Tags beinhalten.

Spaltenname	Beschreibung
EVT_ID	Eindeutige ID eines Events
EVT_CONTEXT_ID	Eindeutige ID des Deployments
CODE	Kurzbezeichnung des Events
LEVEL	Priorität des Events (vgl. 7.2.3.5)
DATE_OF_OCCURRENCE	Zeitpunkt zu dem das Event erzeugt wurde
MACHINE	Physische Maschine auf der das Event entstand
MESSAGE	Textuelle detailierte Beschreibung des Events
STACK_TRACE	Für technische Services, Stacktrace der Fehlermeldung.

- **EVENTS_TAGS**

  Hier werden die Tags des Events gespeichert. Einem Event können mehrere Tags zugeordnet sein.

Spaltenname	Beschreibung
EVT_ID	Eindeutige ID eines Events
TAG	Ein Tag, das diesem Event zugeordnet ist

- **EVENTS_TRANSACTION_IDS** Jedem Event können mehrere Transaction-IDs zugeordnet werden. Diese werden in dieser Tabelle gespeichert.

Spaltenname	Beschreibung
EVT_ID	Eindeutige ID eines Events
TRANSACTION_ID	Transaction-ID, die diesem Event zugeordnet ist

- **EVENTS_UDFS**

  Jedem Event können mehrere User Defined Fields (UDF) zugeordnet werden. Diese sind typisiert und je nach übergebenen Typ werden diese in der entsprechenden Spalte abgelegt.

Spaltenname	Beschreibung
EVT_ID	Eindeutige ID eines Events
BOOLEAN_VAL	Boolean Wert eines UDFs
INTEGER_VAL	Integer Wert eines UDFs
LONG_VAL	Long Wert eines UDFs
FLOAT_VAL	Float Wert eines UDFs
DOUBLE_VAL	Double Wert eines UDFs
DATE_VAL	Zeit- und Datumswert eines UDFs
BINARY_VAL	Binärdaten eines UDFs
STRING_VAL	Datums- und Zeitwert eines UDFs
UDF	Name des UDFs

- **DEPLOYMENTS**

  Ein Deployment stellt ein in einer bestimmten Umgebung laufendes Service dar. Es kann mehrere Events erzeugen. Daraus geht hervor, dass ein Event immer nur in einem Deployment entstehen kann. Deployments können in unterschiedlichen Environments laufen.

Spaltenname	Beschreibung
STC_ID	Eindeutige ID eines Deployments
DEPLOYED_COMPONENT_ID	Eindeutige ID einer Software Komponente
LOCATION	Logische Umgebung des Deployments
DESCRIPTION	Beschreibung des Deployments
CONTACT	Kontaktdaten des Verantwortlichen
CONTACT_EMAIL	Email Adresse des Verantwortlichen

- **COMPONENTS**

  Eine Software Komponente ist ein Stück Software, das wenn es deployt wird Events erzeugen kann. Es kann beliebig oft deployt werden. Diese Deployments werden in der Relation *DEPLOYMENTS* gespeichert. Die Grunddaten der Komponente werden hier gespeichert.

Spaltenname	Beschreibung
COM_ID	Eindeutige ID einer Software Komponente
COM_CODE	Kurzbezeichnung einer Software Komponente
COM_PARENT_ID	Eindeutige ID einer Software Komponente zur Abbildung von hierarchischen Beziehungen
LONGNAME	Name der Software Komponente
DESCRIPTION	Beschreibung der Software Komponente
VERSION	Version der Software Komponente
CONTACT	Kontaktdaten des Verantwortlichen
CONTACT_EMAIL	Email Adresse des Verantwortlichen
COPYRIGHT	Copyright Daten der Software Komponente

- **ENVIRONMENTS**

  Ein Environment stellt eine logische Einheit dar, in dem Deployments zusammengefasst werden. Entsprechend können mehrere Deployments in einem Environment gemacht werden, genauso wie mehrere Environments das gleiche Deployment enthalten kann.

Spaltenname	Beschreibung
STC_ID	Eindeutige ID eines Environments
ENV_CODE	Kurzbezeichnung eines Environments
ENV_PARENT_ID	Eindeutige ID eines Environments, zur Abbildung von hierarchischen Beziehungen
LONGNAME	Name des Environments
DESCRIPTION	Beschreibung des Environments
CONTACT	Kontaktdaten des Verantwortlichen
CONTACT_EMAIL	Email Adresse des Verantwortlichen

Kapitel 8

Prototypische Implementierung

Um zu prüfen wie weit sich die in den vorhergehenden Kapiteln definierten Lösungsideen und Konzepte umsetzen lassen, wird ein Prototyp einer solchen Lösung implementiert. Die Hauptbestandteile werden in den folgenden Kapiteln beschrieben.

Als Programmiersprache wurde Java gewählt, da hier bereits Vorwissen vorhanden war und viele Bestandteile der Zielarchitektur ebenfalls in Java geschrieben worden sind.

8.1 Message Bus

Einer der Bestandteile, die in der Architekturkonzeption definiert wurden ist der Message Bus. Er ist das Backbone auf dem sämtliche Events transportiert werden. Für den Prototypen wurde ActiveMQ als Message Bus ausgewählt. Dies hat mehrere Gründe.

1. ActiveMQ ist in Java geschrieben und fügt sich gut in die restliche Architektur ein.

2. ActiveMQ ist sehr verbreitet und es gibt eine große Anzahl an Artikeln, How-Tos und Tutorials, die Hilfe bieten.

3. ActiveMQ ist für den Prototypen ausreichend schnell. Laut Herstellerangaben können bis zu 20.000 Nachrichten pro Sekunde geschrieben und gelesen werden. [15]

4. Der gewichtigste Grund für die Entscheidung für ActiveMQ ist, dass diese Software unter der Apache 2.0 Lizenz veröffentlicht wurde. Diese Lizenz erlaubt eine freie Verwendung der Software.

8.2 CEP Engine: Esper

Der Kern des Prototypen ist die Complex Event Processing Engine. Als Engine wurde Esper gewählt. Hauptgründe für die Wahl sind zum einen die freie Verfügbarkeit und die Implementierung in Java.

Esper wurde unter der GPL 2.0 veröffentlicht und ist unter http://esper.codehaus.org verfügbar. Durch die Lizenz GPL ist es möglich den Code frei zu nutzen. Die einzige Einschränkung ist, dass direkt gelinkter Code selbst wieder veröffentlicht werden muss.

Neben Esper gibt es nur sehr wenige frei verfügbare Complex Event Processing Engines, wie zum Beispiel Borealis. Borealis ist der Nachfolger von Aurora. Sowohl Borealis, als auch Aurora sind in C++ geschrieben und nur unter Linux zu betreiben. Aus diesen Gründen sind sie nicht geeignet.

8.2.1 Abdeckung Anforderungen

Neben der freien Verfügbarkeit spricht auch der große Funktionsumfang für Esper. Die wichtigsten Anforderungen an eine CEP Engine für den Einsatz im Monitoring werden von Esper abgedeckt, wie die folgenden Punkte zeigen.

8.2.1.1 Geschwindigkeit

Eine der Hauptanforderungen die an eine CEP Engine gestellt wurden erfüllt Esper ausgezeichnet. Laut eigenen Messungen kann Esper 500.000 Nachrichten pro Sekunde verarbeiten. Diese Tests wurden auf handelsüblicher Intel Hardware mit einem 2.0 GHz Prozessor durchgeführt. [35]

8.2.1.2 Mächtige Regelsprache

Esper verfügt über eine sehr mächtige Regelsprache, die auf der bekannten Structured Query Language (SQL) basiert. Diese erlaubt es viele der zuvor definierten Anforderungen abzudecken. Esper nennt diese Sprache Event Processing Language (EPL).

- **Filtern** Die Regelsprache von Esper erlaubt Filtering analog wie SQL.

```
1    select * from TickStream where tickerSymbol = 'GOOG'
```

- **Aggregation** Ebenso wie SQL bietet Esper einige Aggregationsfunktionen, wie zum Beispiel AVG() für die Ermittlung eines Durchschnitts.

```
1    select avg(temperature) from SensorEvent
```

- **Zeitbezug** Ein Feature, das SQL nicht mitbringt ist der Zeitbezug. Esper's EPL bietet dem Nutzer die Möglichkeit Abfragen auf sich fortbewegende Zeitfenster zu machen, wie das folgende Beispiel zeigt.

```
1  select avg(price) from StockTick.win:time(30 sec) where
       symbol='IBM'
```

  Im Beispiel wird der Durchschnittspreis der IBM Aktie der letzten 30 Sekunden ermittelt. Mit jedem zusätzlichen Event wird die Query erneut ausgeführt und der Durchschnittpreis aus allen Events der vergangenen 30 Sekunden berechnet.

- **Abbildung Prozesslogik** Zur Abbildung von Reihenfolgen von Events stellt die EPL die Operatoren ->, *()* sowie *not*, *or*, *and* zur Verfügung die gemeinsam mit dem Pattern - Matching Keyword *every* genutzt werden können.

  Soll zum Beispiel folgende Reihenfolge von Events abgebildet werden *Event A, dann Event B oder Event C*, würde das Pattern folgendermaßen definiert sein.

```
1  select * from pattern [every (A -> (B or C))]
```

8.3 CEP Konfiguration

Die Konfiguration der einzelnen Regeln und der Events, die sie beim Feuern erzeugen wurde in eine eigene Konfigurationsdatei ausgelagert. Der Grund dafür lag in erster Linie in der Vereinfachung der Benutzbarkeit für den Anwender.

Grundsätzlich werden in Esper für die Verarbeitung von Events eigene Eventlistener programmiert. Da aber hier die Funktionsweise der Eventlisteners immer die gleiche wäre, wurde ein konfigurierbarer Eventlistener entwickelt, der standardmäßig ausgeführt wird, sobald eine Regel feuert. Als Konfiguration wird eine Definition eines Events übergeben, das erzeugt werden soll, sobald die definierte Regel feuert.

8.3.1 CEP Konfigurationsdatei

Die Konfigurationsdatei besteht aus zwei grundlegenden Bausteinen, die zu einer CEP-Konfiguration zusammengefasst werden. Diese CEP Konfiguration ist durch die Tags *<cepevent>* und *</cepevent>* umschlossen. Eine Konfigurationsdatei kann mehrere CEP-Konfigurationen enthalten.

- **Regeldefinition** In der Regeldefinition wird die Regel definiert. Sie wird in der Esper eigenen Event Processing Language (EPL) geschrieben, die vollumfänglich unterstützt wird.

- **Eventdefinition** In der Eventdefinition wird der Event beschrieben, der versendet werden soll, sobald eine Regel feuert. Da das erzeugte Event wieder ein Event ist, wie es unter 7.2 definiert wurde, entspricht auch die Konfiguration diesem. Zur Vereinfachung der Konfiguration wurde jedoch auf die Bestandteile verzichtet, die nicht absolut zwingend erforderlich sind. Deshalb besteht die Definition eines Events hier nur aus fünf Teilen.

  1. **code:** Eindeutige Bezeichnung des Events
  2. **level:** Wichtigkeit des Events
  3. **machineOfOrigin:** Rechner auf der das Event erzeugt wurde.
  4. **message:** Beschreibung des Events.
  5. **context:** Bestehend aus *deployedComponentCode* und *deploymentLocation* die die Komponente, der das Event zugeordnet wird definiert.

8.3.2 Beispiel

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>
5       select * from pattern
6       [every start=XMLEvent(code="AddNewEmployee") ->
7       (timer:interval(2 sec) and
8       not done=XMLEvent(code="AddNewEmployeeDone", transactions.
            transactionId[0] = start.transactions.transactionId[0])
9       )]
10    </query>
11    <event>
12      <code>CEP-Filter-Advanced</code>
13      <level>INFO</level>
14      <machineOfOrigin>ftibco99</machineOfOrigin>
15      <message>
16        A done event did not follow a start event within 2sec
17      </message>
18      <context>
19        <deployedComponentCode>CEP-Event 1</deployedComponentCode>
20        <deploymentLocation>CEP Engine</deploymentLocation>
21      </context>
```

```
22      </event>
23    </cepevent>
24 </cepevents>
```

8.3.3 Hotloading von CEP Konfigurationen

Es ist zu erwarten, dass die Konfiguration häufig ändert. Um den Betrieb durch Neustarts nach Anpassungen an der Konfiguration nicht unterbrechen zu müssen, wurde ein Hotloading für die Konfiguration entwickelt.

Darunter versteht man, dass die Konfiguration beim Starten der Applikation geladen wird und ab dann überwacht wird. Sobald sich die Konfiguration ändert, wird sofort die aktualisierte Konfiguration geladen. Tritt beim Laden der CEP Konfiguration ein Fehler, wie zum Beispiel ein Syntax Error in der Definition der Query, so bleibt die bestehende Konfiguration erhalten.

Auf diese Art und Weise können neue CEP Konfigurationen hinzugefügt werden und bestehende verändert oder gelöscht werden. Das Laden doppelter Queries ist unterbunden, da sonst Queries doppelt feuern und dadurch Events doppelt erzeugen würden.

8.4 Event Storage

Als Event Storage wird das freie Datenbank Managementsystem (DBMS) MySQL 5.1 verwendet. Hauptgründe dafür sind die freie Verfügbarkeit, die große Verbreitung und die gute Unterstützung durch Java Persistency Frameworks, wie zum Beispiel Hibernate.

8.4.0.1 SQL Create Statements

Die Tabellendefinitionen die zur Speicherung der Events im Kapitel 7.2.4 definiert. Hier sind die create statements der einzelnen Tabellen so, wie sie im DBMS angelegt wurden.

```
1    create table COMPONENTS (
2        COM_ID bigint not null,
3        COM_CODE varchar(255) not null unique,
4        COM_PARENT_ID bigint,
5        LONGNAME varchar(255),
6        DESCRIPTION varchar(255),
7        VERSION varchar(255),
8        CONTACT varchar(255),
9        CONTACT_EMAIL varchar(255),
10       COPYRIGHT varchar(255),
11       primary key (COM_ID)
```

```
12      ) ENGINE=InnoDB;
13
14      create table DEPLOYMENTS (
15          STC_ID bigint not null,
16          DEP_DEPLOYED_COMPONENT_ID bigint not null,
17          LOCATION varchar(255) not null,
18          DESCRIPTION varchar(255),
19          CONTACT varchar(255),
20          CONTACT_EMAIL varchar(255),
21          primary key (STC_ID)
22      ) ENGINE=InnoDB;
23
24      create table ENVIRONMENTS (
25          STC_ID bigint not null,
26          ENV_CODE varchar(255) not null unique,
27          ENV_PARENT_ID bigint,
28          LONGNAME varchar(255),
29          DESCRIPTION varchar(255),
30          CONTACT varchar(255),
31          CONTACT_EMAIL varchar(255),
32          primary key (STC_ID)
33      ) ENGINE=InnoDB;
34
35      create table ENVIRONMENT_DEPLOYMENT (
36          ENV_ID bigint not null,
37          DEP_ID bigint not null,
38          primary key (ENV_ID, DEP_ID)
39      ) ENGINE=InnoDB;
40
41      create table EVENTS (
42          EVT_ID bigint not null auto_increment,
43          EVT_CONTEXT_ID bigint not null,
44          CODE varchar(255) not null,
45          LEVEL varchar(255) not null,
46          DATE_OF_OCCURRENCE datetime not null,
47          MACHINE varchar(255),
48          MESSAGE longtext,
49          STACK_TRACE longtext,
50          primary key (EVT_ID)
51      ) ENGINE=InnoDB;
52
53      create table EVENT_TAGS (
54          EVT_ID bigint not null,
55          TAG varchar(255) not null,
56          primary key (EVT_ID, TAG)
57      ) ENGINE=InnoDB;
58
```

```
59      create table EVENT_TRANSACTION_IDS (
60          EVT_ID bigint not null,
61          TRANSACTION_ID varchar(255) not null,
62          primary key (EVT_ID, TRANSACTION_ID)
63      ) ENGINE=InnoDB;
64
65      create table EVENT_UDFS (
66          EVT_ID bigint not null,
67          BOOLEAN_VAL bit,
68          INTEGER_VAL integer,
69          LONG_VAL bigint,
70          FLOAT_VAL float,
71          DOUBLE_VAL double precision,
72          DATE_VAL datetime,
73          BINARY_VAL longblob,
74          STRING_VAL varchar(255),
75          UDF varchar(255) not null,
76          primary key (EVT_ID, UDF)
77      ) ENGINE=InnoDB;
78
79
80
81      alter table ENVIRONMENTS
82          add index FKEB63F8C0ADC21E8D (ENV_PARENT_ID),
83          add constraint FKEB63F8C0ADC21E8D
84          foreign key (ENV_PARENT_ID)
85          references ENVIRONMENTS (STC_ID);
86
87      alter table ENVIRONMENT_DEPLOYMENT
88          add index FKF911091C5433E14 (DEP_ID),
89          add constraint FKF911091C5433E14
90          foreign key (DEP_ID)
91          references DEPLOYMENTS (STC_ID);
92
93      alter table ENVIRONMENT_DEPLOYMENT
94          add index FKF9110913448143C (ENV_ID),
95          add constraint FKF9110913448143C
96          foreign key (ENV_ID)
97          references ENVIRONMENTS (STC_ID);
98
99      alter table EVENTS
100         add index EVENTS_EVT_CONTEXT_ID_KEY (EVT_CONTEXT_ID),
101         add constraint EVENTS_EVT_CONTEXT_ID_KEY
102         foreign key (EVT_CONTEXT_ID)
103         references DEPLOYMENTS (STC_ID);
104
105     alter table EVENT_TAGS
```

```
106         add index EVENT_TAGS_EVT_ID_KEY (EVT_ID),
107         add constraint EVENT_TAGS_EVT_ID_KEY
108         foreign key (EVT_ID)
109         references EVENTS (EVT_ID);
110
111     alter table EVENT_TRANSACTION_IDS
112         add index EVENT_TRANSACTION_IDS_EVT_ID_KEY (EVT_ID),
113         add constraint EVENT_TRANSACTION_IDS_EVT_ID_KEY
114         foreign key (EVT_ID)
115         references EVENTS (EVT_ID);
116
117     alter table EVENT_UDFS
118         add index EVENT_UDFS_EVT_ID_KEY (EVT_ID),
119         add constraint EVENT_UDFS_EVT_ID_KEY
120         foreign key (EVT_ID)
121         references EVENTS (EVT_ID);
```

8.5 GUI

Als grafisches Benutzerinterface wurde das Frontend von Lighthouse 3 gewählt. Lighthouse 3 ist eine Monitoringlösung, die von Mercatis entwickelt wird. Hauptgrund dafür ist der Zugang zum Programmcode und die gute Unterstützung durch die Entwickler.

Das Lighthouse 3 GUI bietet die meisten der unter Abschnitt 5.1.8 geforderten Funktionen, wie visuelle Darstellung von Prozessen, visuelles Feedback über Prozesszustand oder Organisation von Prozessen in Subprozessen.

8.5.1 Screenshot

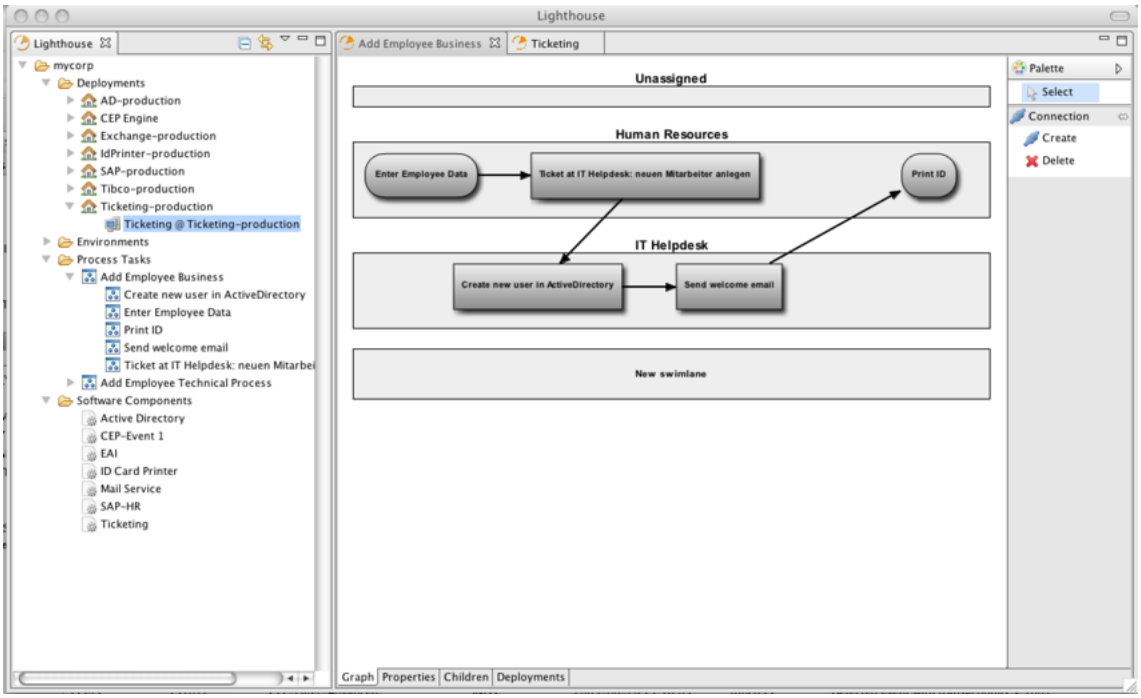

Abbildung 8.1: Screenshot Lighthouse 3 GUI

8.5.2 Aufbau des GUI

Das GUI Gliedert sich in drei Bereiche. Auf der linken Seite ist eine Baumstruktur mit vier Hauptkomponenten: Deployments, Environments, Process Tasks und Software Components. Am rechten Rand sind - sofern man als Benutzer im Prozessbearbeitungsmodus ist - die Malwerkzeuge zur Bearbeitung der Prozesse untergebracht. In der Mitte werden die jeweiligen Nutzdaten verfügbar gemacht.

Von speziellem Interesse sind die Hauptkomponenten an der rechten Seite. Diesen Komponenten liegt ein grundlegendes Konzept zugrunde, dessen Verständnis für die Arbeit mit dem Lighthouse GUI wichtig ist. Aus diesem Grund werden die einzelnen Komponenten hier erklärt.

- **Software Components**

  Eine Software Komponente ist ein Stück Software, das ein (technisches) Service repräsentiert.

- **Deployments**

  Um eine Software Komponente nutzen zu können muss sie in einer Umgebung deployt werden. Selbstverständlich kann eine Software Komponente mehrmals deployt werden, wie etwa ein einer Test- und einer Produktionsumgebung. Diese Deployments werden unter diesem Punkt erfasst.

- **Process Tasks** Unter den Process Tasks können frei Prozesse modelliert werden. Herauszustreichen ist, dass den einzelnen Prozessschritten Deployments zugewiesen werden können. So wird die Verbindung zwischen Prozessschritten und den tatsächlichen technischen Services die sie ermöglichen geschaffen.

- **Environments** Unter dem Punkt Environments können Deployments zu logischen Gruppen zusammengefasst werden. So können für verschiedene Anwender angepasste Sichten auf die Deployments generiert werden.

Zur Herstellung der Ampelfunktionalität ist es möglich auf Prozesstasks, aber auch auf einzelne Deployments oder Environments Status zu definieren. Diese nehmen für die entsprechenden Deployments, oder Gruppen von Deployments die Events entgegen und verändern den Status entsprechend.

Kapitel 9

Test

Um sicherzustellen, dass der Prototyp zuverlässig läuft und die Ergebnisse der Case-Study aussagekräftig sind, muss er eingehend getestet werden. Zwei verschiedene Tests werden durchgeführt. Der erste Test prüft die Qualität des Prototypen. Der zweite Test prüft die Belastbarkeit anhand eines Lasttests.

9.1 Test Setup

9.1.1 Überblick

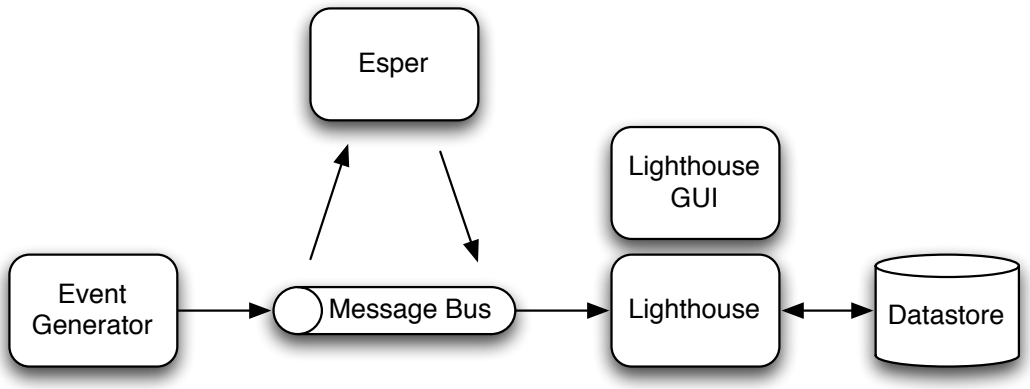

Abbildung 9.1: Architektur des Testsystems

9.1.2 Testsystem

Sämtliche Bestandteile des Testsystems laufen auf der selben Maschine. Dies hat zwei Vorteile: Erstens, es erleichtert das Aufsetzen eines Testsystems. Es ist nicht notwendig eigene dedizierte Maschinen für Testzwecke bereit zu stellen.

Der zweite Vorteil dieses einfachen Setups ist, dass Einflüsse auf die Übertragung der Daten, wie sie in einem Netzwerk auftreten können ausgeschlossen werden. Außerdem ist durch den Verzicht auf Netzwerk-Infrastruktur ein potentielles Bottleneck von vornherein ausgeschlossen.

9.1.2.1 Technische Daten

Rechner	Apple MacBook
Prozessor	2.4 GHz Intel Core 2 Duo
Arbeitsspeicher	4 GB 1067 MHz DDR
L2 Cache	3MB
Betriebssystem	Mac OS X 10.5.8 (Leopard)
JVM	Java 1.5.0_19

9.1.3 Event Generator

Aufgabe des Event Generators ist es möglichst einfach und verlässlich Events zu generieren und die CEP-Engine damit zu versorgen. So sollen Verhältnisse, wie sie in realen Umgebungen entstehen können, simuliert werden. Für diese Aufgabe wurde das verbreitete Java Lasttest-Werkzeug JMeter ausgewählt.

Für den Einsatz von JMeter sprechen mehrere Gründe.

- **Native JMS Unterstützung**

  JMeter wird bereits mit drei verschiedenen JMS Samplern ausgeliefert. Jeweils ein Sampler für JMS-Topic-Publishing und JMS-Topic-Subscribe, sowie ein Sampler sowohl für JMS-Queue-Request und respond. Damit ist es nicht nötig eigene Sampler zu schreiben.

- **Kontrolle des Ablaufflusses**

  In JMeter können beliebig viele Sampler erzeugt werden, die Events versenden können. Zusätzlich können für diese aber auch noch Loops und ähnliche Steuerelemente für die Ablaufsteuerung definiert werden. Damit es ist möglich auf einfache Weise komplexe Event-Datenströme zu generieren.

- **Grafische Benutzerschnittstelle**

JMeter bietet eine grafische Benutzerschnittstelle an. Damit können Testdefinitionen direkt in einem einfachen Java GUI erstellt werden. Vor allem bei komplexeren Abläufen erhöht das die Verständlichkeit der einzelnen Testdefinitionen.

- **Lasttest Integriert**

  JMeter ist ursprünglich für Lasttests entwickelt worden. Es erlaubt die Ausführung der definierten Tests in beliebig vielen Threads, mit beliebig vielen Wiederholungen und auf einen selbst bestimmbaren Zeitraum verteilt zu starten. So kann sehr schnell sehr viel Last auf einem System erzeugt werden.

  Im Gegensatz dazu ist es natürlich auch möglich gleichzeitige Threads, Wiederholungen auf 1 zu setzen. Durch diese Parameter wird aus dem Lasttest ein einfacher Funktionaler Test.

9.2 Testmethode

Um feststellen zu können, ob der erstellte Prototyp fehlerfrei läuft ist es notwendig qualitative Tests durchzuführen. Diese Tests werden als Testfälle definiert und in Testruns durchgeführt. Dabei werden sowohl Positiv-, als auch Negativ-Tests erstellt. Positiv-Tests prüfen ob die definierte Funktionalität korrekt ist, Negativ-Tests prüfen, ob die Fehlerbehandlung ausreichend ist und ob die Software durch gezielte Falscheingaben zum Absturz gebracht werden kann.

9.2.1 Testfall

Ein Testfall besteht dabei immer aus fünf Elementen:

- **ID**

  Jedem Testfall wird eine eindeutige ID zugewiesen. Sinn dieser ID ist eine eindeutige Kennzeichnung eines Testfalls, um später darauf referenzieren zu können.

- **Name**

  Der Name des Testfalls soll den Testfall selbst beschreiben. Schon durch den Namen soll klar werden, was der Test

- **Beschreibung**

Jeder Testfall wird detailliert beschrieben. Der Detailierungsgrad soll dabei so gewählt werden, dass eine Person, die nicht der Entwickler selbst ist, den Testfall selbstständig durchführen kann.

- **Input**

  Werden spezielle Input Daten genutzt, so müssen sie im Feld Input definiert werden, um die Nachvollziehbarkeit und Wiederholbarkeit der Testfälle sicherzustellen.

- **Erwartetes Ergebnis**

  Für jeden Testfall wird eine erwartete Reaktion des Systems definiert. Mit dieser kann das erzielte Ergebnis verglichen und der Erfolg des Tests definiert werden.

9.2.2 Testrun

Die tatsächliche Durchführung eines Testfalls ist der Testrun. Dieser wird vom Tester ausgeführt und dokumentiert. Für die Dokumentation werden hier folgende Felder genutzt:

- **Datum** Im Feld Datum wird der Ausführungszeitpunkt festgehalten. Dies dient der Vergleichbarkeit von verschiedenen Testruns.
- **ID** Die ID ist die Referenz zum Testfall. Sie zeigt an welcher Testfall durchgeführt wurde.
- **Ergebnis** Im Feld Ergebnis wird das tatsächliche Testergebnis festgehalten. Entspricht es dem erwarteten Ergebnis reicht die Meldung *OK*. Weicht es vom erwarteten Ergebnis ab, so muss dieses dokumentiert werden.

9.3 Testfall Beschreibungen

9.3.1 Grundfunktionalität

9.3.1.1 Starten der Applikation

- **ID** GF1
- **Name** Starten der Applikation
- **Beschreibung** Die Applikation wird mit einer gültigen Konfigurationsdatei als Input Parameter gestartet und die Konfiguration wird eingelesen.

- **Input** -
- **Erwartetes Ergebnis** Applikation startet ohne Fehler und gibt Feedback über die eingelesene Konfiguration.

9.3.1.2 Filtern von Events aus dem Datenstrom

- **ID** GF2
- **Name** Filtern von Events aus dem Datenstrom
- **Beschreibung** Die Applikation ist gestartet und empfängt Events. Events mit dem Code *Basic Event 1* werden herausgefiltert und erzeugen ein CEP Event mit dem Code *CEP Event 1*.
- **Input** B.1.1.1
- **Erwartetes Ergebnis** Ein neues Event wird erzeugt und die Applikation gibt Feedback dazu.

9.3.1.3 Ampelfunktion

- **ID** GF3
- **Name** Ampelfunktion
- **Beschreibung** Ein CEP Event löst ein Einfärben eines Prozessschrittes aus im GUI aus.
- **Input** B.1.1.2
- **Erwartetes Ergebnis** Ein Prozessschritt im GUI wird rot.

9.3.2 CEP Query Sprache

9.3.2.1 Abbildung von Vorher-Nachher Beziehungen

- **ID** QL1
- **Name** Abbildung von Vorher-Nachher Beziehungen
- **Beschreibung** Zwei unterschiedliche aufeinanderfolgende Events müssen auf dem Eventstrom herausgefiltert werden und lösen ein CEP Event aus.
- **Input** B.1.2.10

- **Erwartetes Ergebnis** Ein CEP Event wird erzeugt, wenn Event 2 direkt auf Event 1 folgt.

9.3.2.2 Abbildung Zeitfenster

- **ID** QL2
- **Name** Abbildung Zeitfenster
- **Beschreibung** Ein 10 Sekunden Zeitfenster wird aufgespannt und Events werden in zufälliger Frequenz gefeuert. Immer wenn zwei bestimmte Events innerhalb dieses Fensters eintreten, soll ein CEP Event gefeuert werden.
- **Input** B.1.2.11
- **Erwartetes Ergebnis** Ein CEP Event wird erzeugt, sobald zwei Events innerhalb von 10 Sekunden eintreffen.

9.3.2.3 Erkennen fehlender Events

- **ID** QL3
- **Name** Erkennen fehlender Events
- **Beschreibung** Es werden zwei Events in einer Transaktion innerhalb von 10 Sekunden erwartet. Das zweite Event bleibt aus. Beide Events haben die gleiche Transaction-ID über die sie zugeordnet werden können.
- **Input** B.1.2.12
- **Erwartetes Ergebnis** Ein CEP Event wird erzeugt, wenn innerhalb von 10 Sekunden kein passendes Event empfangen wird.

9.3.2.4 Vergleich mit Operatoren

- **ID** QL4
- **Name** Vergleich mit Operatoren
- **Beschreibung** Events mit der Transaction-ID > 1001 soll herausgefiltert werden und jedes Mal ein CEP Event erzeugt werden.
- **Input** B.1.2.13
- **Erwartetes Ergebnis** Bei einem Event mit einer Transaction-ID > 1001 wird ein CEP Event erzeugt.

9.3.3 Hotloader CEP Konfiguration

9.3.3.1 Start mit einer Konfiguration

- **ID** KO1
- **Name** Start mit einer CEP Konfiguration
- **Beschreibung** Starte die Applikation mit einer Konfigurationsdatei, die nur eine CEP Konfiguration enthält
- **Input** B.1.2.1
- **Erwartetes Ergebnis** Applikation startet, fügt die CEP Konfiguration hinzu und gibt Feedback.

9.3.3.2 Start mit mehreren Konfigurationen

- **ID** KO2
- **Name** Start mit zwei Konfigurationen
- **Beschreibung** Starte die Applikation mit einer Konfigurationsdatei, die mehr als eine CEP Konfigurationen enthält
- **Input** B.1.2.2
- **Erwartetes Ergebnis** Applikation startet, fügt alle CEP Konfigurationen hinzu und gibt Feedback.

9.3.3.3 Hinzufügen einer CEP Konfiguration

- **ID** KO3
- **Name** Hinzufügen einer CEP Konfiguration
- **Beschreibung** Öffne die Konfigurationsdatei und füge eine weitere CEP Konfiguration hinzu. Speichere diese Datei.
- **Input** B.1.2.1 -> B.1.2.2
- **Erwartetes Ergebnis** Applikation lädt die veränderte Konfigurationsdatei und gibt Feedback.

9.3.3.4 Löschen einer CEP Konfiguration

- **ID** KO4
- **Name** Löschen einer CEP Konfiguration
- **Beschreibung** Öffne die Konfigurationsdatei und entferne eine CEP Konfiguration. Speichere diese Datei.
- **Input** B.1.2.2 -> B.1.2.1
- **Erwartetes Ergebnis** Applikation lädt die veränderte Konfigurationsdatei und gibt Feedback

9.3.3.5 Doppelte CEP Konfiguration

- **ID** KO5
- **Name** Dopplete CEP Konfiguration
- **Beschreibung** Öffne die Konfigurationsdatei und dupliziere eine der CEP Konfigurationen. Speichere die Datei.
- **Input** B.1.2.3
- **Erwartetes Ergebnis** Applikation lädt alle CEP Konfigurationen außer der doppelten und gibt Feedback zu doppelter Konfiguration.

9.3.3.6 Fehler in XML Format

- **ID** K06
- **Name** Fehler in XML Format
- **Beschreibung** Starte die Applikation mit einer nicht wohlgeformten Konfigurationsdatei.
- **Input** B.1.2.4
- **Erwartetes Ergebnis** Applikation startet, wirft aber entsprechende Fehlermeldung aus.

9.3.3.7 Fehler in Query

- **ID** KO7
- **Name** Fehler in Query
- **Beschreibung** Öffne die Konfigurationsdatei und verändere die Query, sodass sie nicht mehr gültig ist. Speichere die Datei.
- **Input** B.1.2.5
- **Erwartetes Ergebnis** Die Konfigurationsdatei wird eingelesen und alle CEP Konfigurationen verarbeitet. Die CEP Konfiguration mit dem Fehler in der Query wird nicht geladen. Die Applikation zeigt eine Warnung an.

9.3.3.8 Fehler in Eventdefinition

- **ID** KO8
- **Name** Fehler in Eventdefinition
- **Beschreibung** Öffne die Konfigurationsdatei und verändere die Definition des Events so, dass sie nicht mehr gültig ist. Speichere die Datei.
- **Input** B.1.2.6
- **Erwartetes Ergebnis** Applikation lädt Datei und loggt Fehlermeldung zu Eventdefinition.

9.3.3.9 Fehlende Eventdefinition

- **ID** KO9
- **Name** Fehlende Eventdefinition
- **Beschreibung** Öffne die Konfigurationsdatei und lösche aus einer CEP Konfiguration die gesamte Eventdefinition. Speichere die Datei.
- **Input** B.1.2.7
- **Erwartetes Ergebnis** Die Konfigurationsdatei wird eingelesen und alle CEP Konfigurationen verarbeitet. Die CEP Konfiguration mit der fehlenden Eventdefinition wird nicht geladen. Die Applikation zeigt eine entsprechende Fehlermeldung an.

9.3.3.10 Fehlende Query

- **ID** K10
- **Name** Fehlende Query
- **Beschreibung** Öffne die Konfigurationsdatei und lösche aus einer CEP Konfiguration die gesamte Query. Speichere die Datei.
- **Input** B.1.2.8
- **Erwartetes Ergebnis** Die Konfigurationsdatei wird eingelesen und alle CEP Konfigurationen verarbeitet. Die CEP Konfiguration mit der fehlenden Query wird nicht geladen. Die Applikation zeigt eine entsprechende Fehlermeldung an.

9.3.3.11 Auskommentierte CEP Konfiguration

- **ID** KO11
- **Name** Auskommentierte CEP Konfiguration
- **Beschreibung** Öffne die Konfigurationsdatei und kommentiere mit <!– und –> eine CEP Konfiguration aus. Speichere die Datei.
- **Input** B.1.2.9
- **Erwartetes Ergebnis** Die Konfigurationsdatei wird eingelesen und alle CEP Konfigurationen verarbeitet. Die auskommentierte CEP Konfiguration wird nicht geladen. Die Applikation zeigt keine Fehlermeldung an.

9.4 Lasttest

Zur Bestimmung der Leistungsfähigkeit wird ein Lasttest durchgeführt. Dabei stehen zwei Parameter im Mittelpunkt: Erstens, die Geschwindigkeit mit der ankommende Anfragen verarbeitet werden können und zweitens die Stabilität bei großen Datenmengen.

9.4.1 Testaufbau

Die Testumgebung entspricht der unter 9.1 definierten Umgebung. Als Erzeuger der Last dient JMeter, ein frei verfügbares javabasierendes Lasttest-Tool.

Mit JMeter werden Events erzeugt und in eine JMS Queue geschoben. Diese Queue wird von der Applikation überwacht und alle Nachrichten, die dort ankommen werden sofort verarbeitet.

Die Verarbeitung geschieht in drei Schritten. Zuerst werden die Nutzdaten aus der JMS Nachricht ausgelesen. Diese werden dann an die Regelengine weitergereicht. Feuert die Regel so wird im letzten Schritt eine JMS Nachricht, mit der CEP Nachricht im Body erzeugt, die an die Speicherkomponente weitergereicht wird.

9.4.2 Testdurchführung

Die Durchführung der Tests erfolgt in drei Schritten. Zuerst wird die Datenbank geleert um sicherzustellen, dass bei der Auswertung der Daten nur die Daten des aktuellen Testlaufs einbezogen werden.

Im zweiten Schritt werden wird die Konfiguration in JMeter angepasst und der Testlauf gestartet.

Im Anschluss daran werden im letzten Schritt die Daten ausgewertet. Dabei wird mit dem unten definierten SQL Statement der Start- und Endzeitpunkt der Verarbeitung ermittelt. Die Dauer zwischen den beiden Zeitpunkten stellt die gesamte Verarbeitungsdauer dar.

```
1 select max(date_of_occurrence) as max, min(date_of_occurrence) as
    min from events;
```

Zwei verschiedene Testserien werden gestartet. Zuerst wird lediglich ein Thread erzeugt, in dem JMS Nachrichten erzeugt werden. Mit dieser Konfiguration werden mehrere Tests durchgeführt, wobei die Zahl der erzeugten Events mit jedem Lauf gesteigert wird. Der Parameter *ramp-up period* wird dabei auf 1 belassen, was bedeutet, dass die Nachrichten innerhalb einer Sekunde erzeugt werden.

In der zweiten Testserie wird versucht durch den Einsatz mehrerer Threads die Anzahl an gleichzeitig eintreffenden Nachrichten zu erhöhen um so Grenzen der Stabilität der Applikation zu erkennen.

9.4.3 Konfiguration JMeter

JMeter verwaltet seine Konfigurationen in Testplänen. Auch für den Lasttest wurde ein eigener Testplan erstellt. Diese Testplan besteht aus drei Komponenten.

1. **Thread Group Loop**

   In der Thread Group können beliebig viele Threads erzeugt werden, die gleichzeitig Last erzeugen. Zusätzlich kann noch eingestellt werden, wie oft die Schritte innerhalb dieser Thread Group wiederholt werden sollen.

2. **JMS Point-to-Point Sampler**

   Der JMS Sampler erzeugt JMS Nachrichtung und sendet diese an eine Queue. Er benötigt als Input-Parameter lediglich den Namen der Queue und die Adresse des JMS Servers, die als JNDI Parameter übergeben werden können. Zusätzlich muss die ConnectionFactory und die Nachricht, die im Message Body übergeben werden soll, angegeben werden.

3. **Summary Report**

   Der Summary Report gibt Auskunft über die Anzahl der korrekt, sowie der fehlerhaft versendeten Nachrichten. Zusätzlich zeigt er die minimale, maximale und die Gesamtdauer an, die zum Erzeugen und versenden der Nachrichten benötigt wurden.

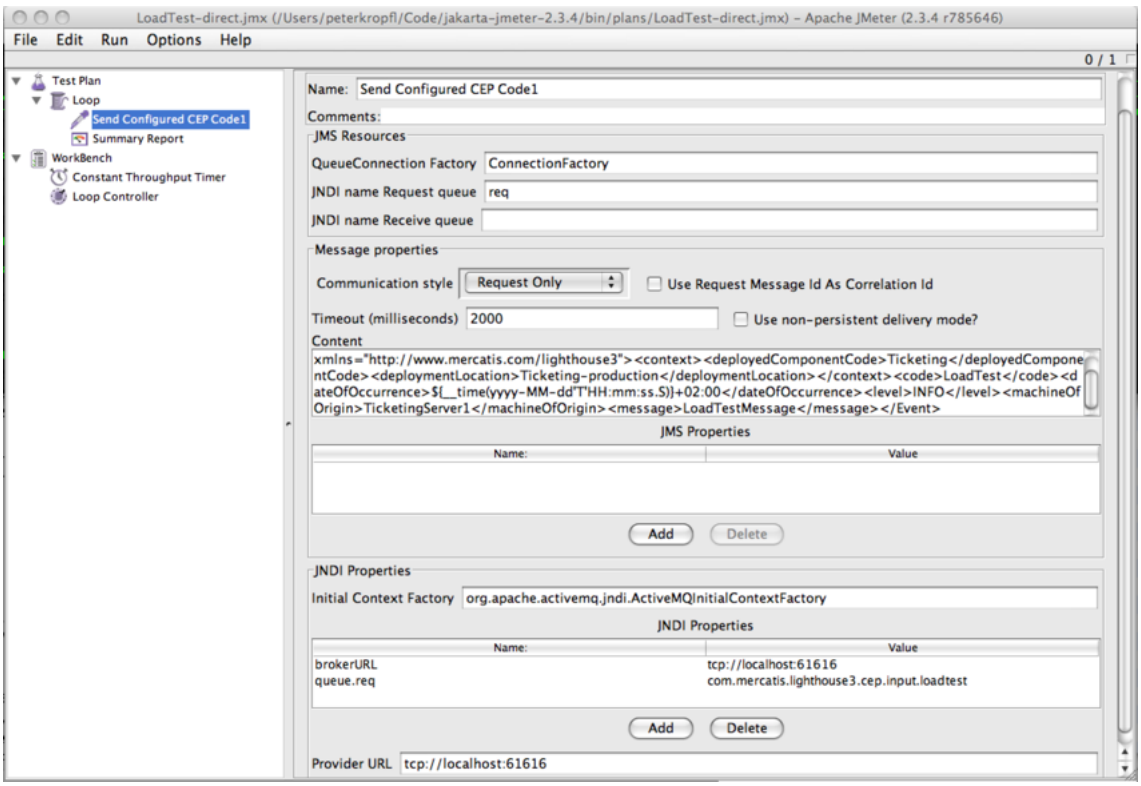

Abbildung 9.2: Screenshot JMeter

9.4.4 Testdaten

9.4.4.1 Input Event

Als Input dient ein einfaches Event, das im Message Body der JMS Nachricht als Text übergeben wird. Das einzige Feld, das außergewöhnlich behandelt wird ist das Feld *dateOfOccurrence*. Hier wird über die JMeter Funktion *time()* das aktuelle Datum und die aktuelle Zeit erzeugt. Wenn die Input-Events ebenfalls in der Datenbank gespeichert werden, so kann durch das Datum ermittelt werden, wie lange das Lasttest-Tool für die Generierung aller Input-Events gebraucht hat.

```
1 <Event xmlns="http://www.mercatis.com/lighthouse3">
2   <context>
3     <deployedComponentCode>Ticketing</deployedComponentCode>
4     <deploymentLocation>Ticketing-production</deploymentLocation>
5   </context>
6   <code>LoadTest</code>
```

```
7   <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
        dateOfOccurrence>
8   <level>INFO</level>
9   <machineOfOrigin>TicketingServer1</machineOfOrigin>
10  <message>LoadTestMessage</message>
11 </Event>
```

9.4.4.2 CEP Konfiguration

Die CEP Konfiguration wurde ebenfalls sehr einfach gestaltet. Das erzeugte CEP Event ist statisch und kurz gehalten. Die Query wurde so gewählt, dass sie bei jedem ankommenden Event feuert. Durch einen Vergleich der Anzahl der Input-Events und der erzeugten CEP Events kann auf die Stabilität der Engine auch unter Last geschlossen werden.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>.
4     <query>select * from XMLEvent where code='LoadTest'</query>
5     <event>
6         <code>LoadTestEvent</code>
7         <level>INFO</level>
8         <machineOfOrigin>mymac</machineOfOrigin>
9         <message>Loadtest Event Message</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16 </cepevents>
```

9.4.5 Ergebnisse Lasttest

9.4.5.1 Zusammenfassung

Ohne jegliche vorhergehende Optimierung erreichte der Prototyp bereit sehr respektable Verarbeitungsraten. Selbst unter Belastungen von 50.000 Nachrichten fällt die Verarbeitungsrate mit sequenzieller Verarbeitung der Nachrichten nicht unter 7680 Events pro Minute.

Die Stabilität der gesamten Umgebung ist ebenfalls ausgezeichnet. Bei keinem, der getesteten Szenarios traten Fehler, wie beispielsweise fehlende Events auf.

Zu beobachten ist, dass erst ab 500 Events pro Sekunde die Verarbeitungsrate messbar abnimmt, wobei die Geschwindigkeit mit der die Verarbeitungsgeschwindigkeit abfällt nicht proportional zur Menge verläuft. Ab ca. 10.000 Events verlangsamt sich der Abfall deutlich und selbst bei 100.000 Events fällt die Verarbeitungsrate nicht unter 110 Events pro Sekunde.

9.4.5.2 Detail Ergebnisse

Events	Dauer in s	Rate Events/s
100	1	100,00
100	1	100,00
200	1	200,00
300	1	300,00
400	1	400,00
500	1	500,00
600	2	300,00
700	2	350,00
800	3	266,67
900	4	225,00
500	1	500,00
1000	4	250,00
1500	6	250,00
2000	8	250,00
3000	12	250,00
4000	17	235,29
5000	21	238,10
6000	28	214,29
7000	35	200,00
8000	43	186,05
9000	53	169,81
10000	61	163,93
11000	70	157,14
12000	77	155,84
13000	84	154,76
14000	92	152,17
15000	101	148,51
20000	163	122,70
25000	204	122,55
30000	233	128,76
35000	276	126,81
40000	335	119,40
45000	385	116,88
50000	388	128,87
100000	841	114,81

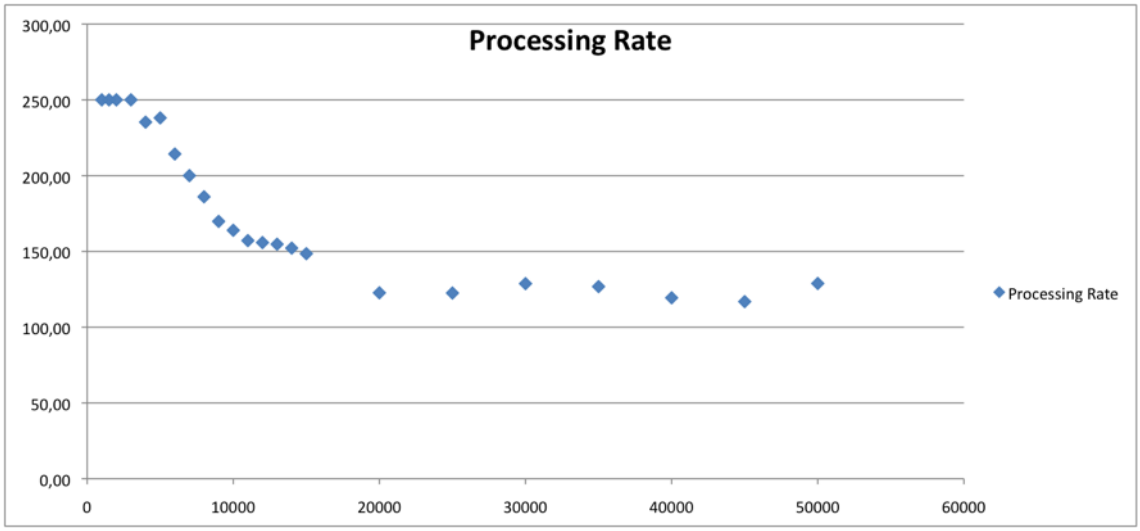

Abbildung 9.3: Verarbeitungsrate

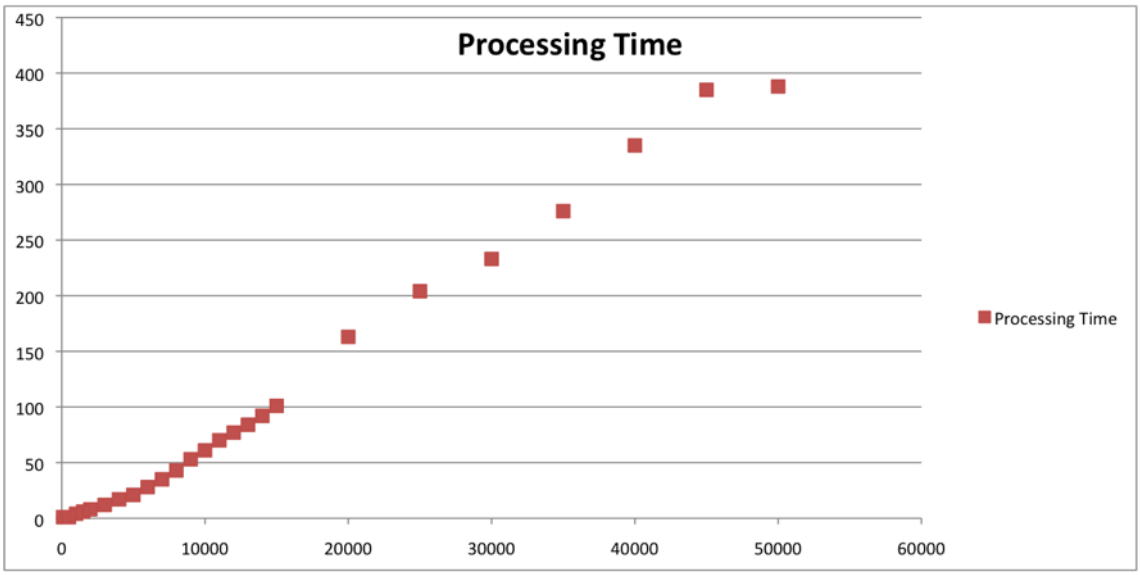

Abbildung 9.4: Verarbeitungsdauer

Kapitel 10

Case Study

In dieser Case-Study soll der Prototyp in einer realistischen Umgebung getestet werden. Da der Ansatz aber tiefe Eingriffe in bestehende Infrastrukturen voraussetzen würde und dies nicht in einer produktiven IT Landschaft möglich war, wurden verschiedene Szenarios nachgestellt um so möglichst gut typische Anforderungen abbilden zu können. Diese wurden analog zu den Lasttests mittels des Testwerkzeugs JMeter implementiert.

10.1 Business Case

10.1.1 Prozessbeschreibung

Der Business Case ist ein Prozess, der die Bearbeitung eines Trades im Bereich Energiehandel abbildet. Durchschnittlich werden 400 Trades pro Tag durch die Händler im Asset-Trading initiiert.

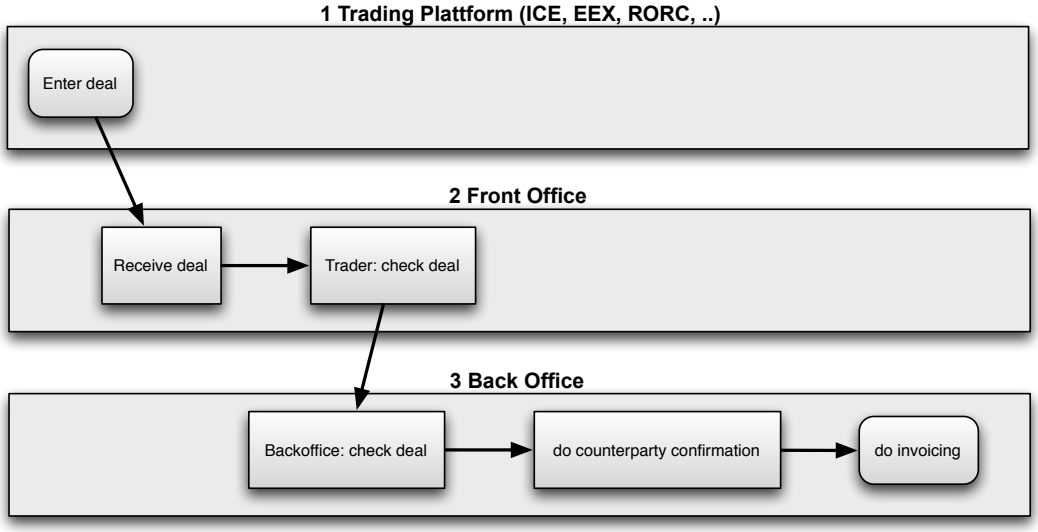

Abbildung 10.1: Geschäftsprozess: Abwicklung Energiehandel

Der Prozess spannt sich über drei Swimlanes, die unterschiedliche Bereiche repräsentieren. Dabei handelt es sich um die Bereiche Frontoffice und Backoffice, die für die Bearbeitung von Deals verantwortlich sind. Beide Bereiche sind innerhalb des Unternehmens angesiedelt. Die oberste Swimlane repräsentiert eine der unterschiedlichen Strombörsen.

Ein Trader erstellt auf der Handelsplattform einer dieser Börsen einen Deal. Dieser Deal wird über spezielle Schnittstellen in das Handelssystem des Unternehmens übertragen. Dort muss der Deal zunächst vom Trader selbst validiert werden. Sobald der Trader die Korrektheit des Deals bestätigt, wird er an das Backoffice zu weiteren Bearbeitung weitergeleitet.

Im Backoffice wird jeder Deal ein weiteres Mal überprüft. Sofern alle Prüfungen in Ordnung abgeschlossen sind, wird er zur Confirmation weitergeleitet. In diesem Prozessschritt wird jeder Deal mit dem Handelspartner abgeglichen. Somit ist sichergestellt, dass beide Handelspartner handelseins sind und die Verrechnung und Energielieferung kann stattfinden.

10.1.2 Abhängigkeiten zu weiteren Prozessen

Der Prozess *Abwicklung Energiehandel* hat Abhängigkeiten zu dem Folgeprozess *Tagesabschluss*. Im Rahmen des Tagesabschluss-Prozesses wird die Gesamtrisiko-Berechnung durchgeführt. Basis dafür sind vollständig bearbeitete Deals. Kann diese Risikoanalyse nur unvollständig durchgeführt werden, so ergibt sich daraus

ein hoher Risikolevel und die Händler können am folgenden Tag nur innerhalb von sehr engen Bahnen handeln, was sich negativ auf deren Performance auswirkt.

Aus diesem Grund ist es notwendig, dass der Prozess der Bearbeitung der Deals schnell und zügig durchlaufen werden kann. Möglichst frühzeitig müssen Probleme erkannt und behandelt werden können.

10.1.3 Typische Probleme

Ein Problem im Prozess *Abwicklung Energiehandel* ist, dass Deals "verlorengehen". Dies kann unterschiedlichste Gründe haben und an verschiedenen Stellen innerhalb des Prozesses bemerkt werden. Durch die komplexe Architektur der Handelssysteme ist es aber eine langwierige und komplizierte Suche, die nur von Experten durchgeführt werden kann.

Die Gründe für verlorene Deals können vielfältig sein. Manchmal treten Fehler am Deal-Interface zu eine Börse auf, ein anderes Mal hat ein Trader auf der Handelsplattform einer Börse einen Custom-Deal erstellt, der so von den eigenen nachgelagerten Systemen nicht verarbeitet werden kann.

Ein weiteres Problem ist, dass es an den Schnittstellen zwischen den Abteilungen im Bereich Trading und im Bereich Backoffice zu Reibungsverlusten kommt und so gewisse Abläufe im Prozess zu lange dauern. Ein Beispiel dafür sind Ansuchen um Ergänzung der Dealinformationen des Backoffice bei Tradern.

10.1.4 Ziele des Monitorings

Durch das Monitoring dieses Prozesses soll die Qualität dieses Prozesses sichergestellt und überprüfbar gemacht werden. Ziel ist es den Prozess schneller und sicherer zu machen. Dies soll über die folgenden vier Punkte geschehen.

- **Aufzeigen von Systemfehlern**

  Beispiel: Probleme beim Import eines Deal durch das RORC-Deal-Interface werden nicht mehr nur in das Logfile auf einem Cluster-Node des Handelssystems geschrieben, sondern sofort bei ihrem Auftreten vom Monitoring erkannt und im GUI dargestellt.

- **Aufzeigen von Bedienungsfehlern**

  Beispiel: Wenn ein Trader auf der Handelsplattform einer Börse einen Deal erstellt und nicht ausreichend Informationen eingibt, damit das Backoffice mit seinen Tools automatisch die Validierung durchführen kann, so wird der fehlerhafte Deal direkt im Monitoring an der korrekten Position in der

Prozesskette angezeigt, inklusive der verfügbaren Informationen zu diesem Deal.

- **Einhalten von maximalen Durchlaufzeiten**

  Beispiel: Wenn die Validierung eines Deals durch das Backoffice länger als die erlaubte Maximaldauer braucht, so wird dies im Monitoring angezeigt und eine entsprechende Warnung für diesen speziellen Deal ausgegeben.

- **Erkennen von *"verlorenen"* Deals**

  Beispiel: Wird ein Deal von einer Counterparty nicht konfirmiert, so wird er in der automatischen Weiterverarbeitung aller Deals ignoriert. Bleibt die Weiterverarbeitung eines Deals aus, so soll dies nach einer gewissen Zeitdauer im Monitoring entsprechend kenntlich gemacht werden.

10.2 Services im Prozess

Der Prozess wird von verschiedenen Services unterstützt, die kurz vorgestellt werden.

- **Market Interface**

  Über das Market Interface Service sind die unterschiedlichen Handelsplattformen der einzelnen Energiehandelsplätze wie RORC, ICE oder EEX angeschlossen. An dieser Schnittstelle gelangen die Deals, die ein Trader auf der Handelsplattform erstellt in das Unternehmen. Jeder Deal, der über diese Schnittstelle in das Unternehmen gelangt, wird mit seiner Deal-ID geloggt.

- **Trader Confirmation Service**

  Das Trader Confirmation Service zeigt dem Trader alle Deals an, die er auf den verschieden Handelsplattformen erstellt hat und die in das Handelssystem des Unternehmens gelangt sind. Hier muss der Trader die Korrektheit seiner Deals bestätigen, bevor sie weiterverarbeitet werden können. Sobald ein Trader einen seiner Deals positiv bestätigt, liefert das Trader Confirmation Service Deal-ID, Zeitstempel den Namen des Traders zurück.

- **Portfolio Risk Analyser**

  Der Portfolio Risk Analyser aggregiert Deals eines Portfolios und errechnet das Risikolevel und das gesamte Exposure des Porfolios.

- **Backoffice Deal Validator**

  Der Backoffice Deal Validator ist ein Service der den Deal auf Vollständigkeit prüft. Dazu zählt beispielsweise die Prüfung ob alle notwendigen Daten vorhanden sind, oder ob mit der Counterparty ein Rahmenvertrag zum Handel besteht. Der Deal Validator versendet eine OK Meldung gemeinsam mit der Deal-ID sobald die Prüfung erfolgreich war.

- **Electronic Confirmation Service**

  Das Electronic Confirmation Service übernimmt die automatische Bestätigung eines Deals. Dazu schickt es die Basisdaten eines Deals wie Deal-ID, Preis, Volumen und Laufzeit an die Counterparty und wartet auf eine Antwort der Counterparty mit den selben Basisinformationen. Sobald die Counterparty eine derartige Bestätigung schickt, liefert das Confirmation Service eine OK Meldung und die Basisinformationen eines Deals zurück.

10.3 Abbildung im GUI

Die zuvor definierten Prozesse und Services werden in Lighthouse abgebildet. Dazu wird für jedes Service eine Software Komponente angelegt. Zu beachten ist dabei, dass durch die Baumstruktur auch zusammengesetzte Services abgebildet werden können.

Diese einzelnen Software Komponenten werden auf verschiedenen Systemen deployt. Dies wird im ersten Punkt *Deployments* abgebildet.

Im Menüpunkt *Process Tasks* wird der zuvor beschriebene Prozess abgebildet und den einzelnen Arbeitsschritten, die zugehörigen Deployments zugeordnet. So kann danach durch die Definition von *Status* für jeden der einzelnen Arbeitsschritte eine Ampel definiert werden, die auf den Statusmeldungen der einzelnen Deployments basiert.

Im Bereich *Environments* können Deployments beliebig gruppiert werden. In diesem Beispiel wurden sie nach Einheiten im Organigramm gruppiert.

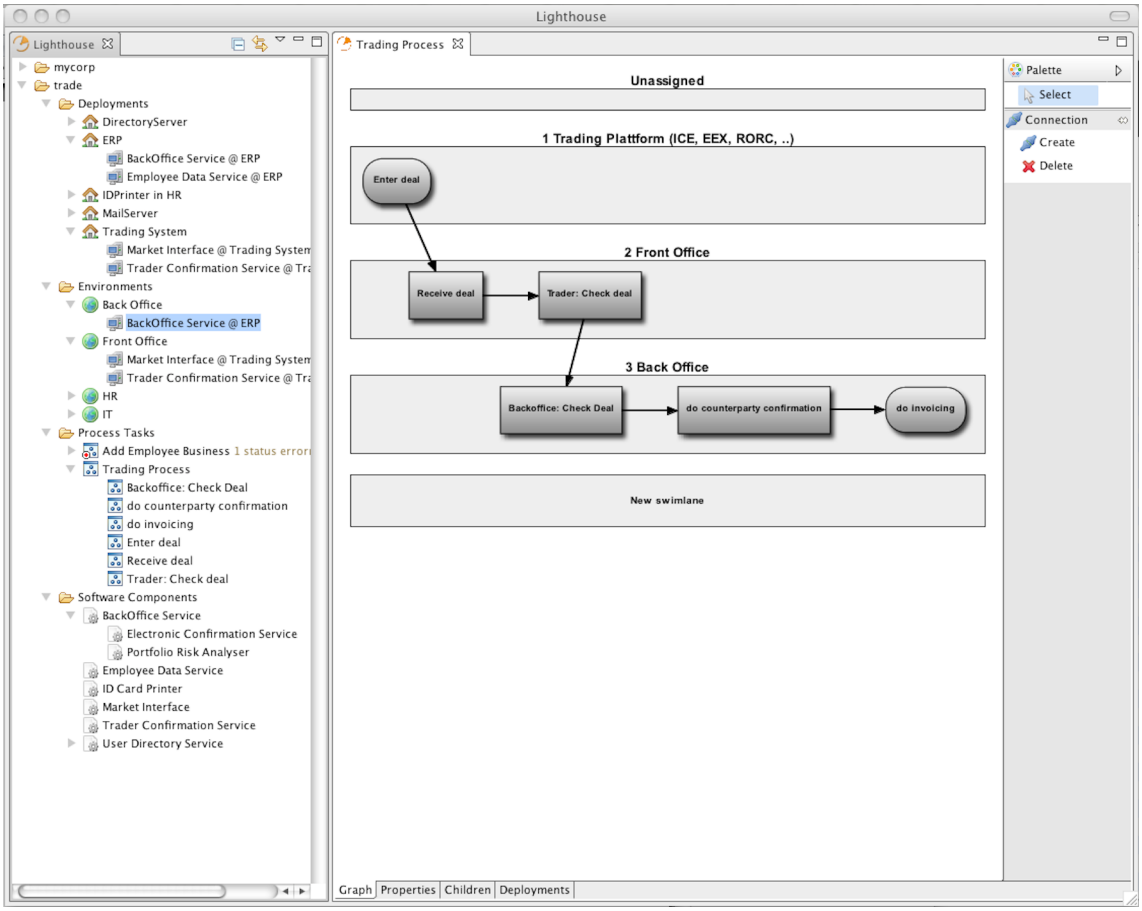

Abbildung 10.2: Screenshot Abbildung Prozess und Services im GUI

10.4 Szenarios

Um in der Case-Study reale Einsatzbedingungen und Anforderungen simulieren zu können, werden unterschiedliche Szenarios definiert. Diese Szenarios werden mit Hilfe von JMeter Testplänen implementiert. Im folgenden Teil werden die einzelnen Szenarios kurz erörtert, sowie die Abbildung durch Events und Regeln gemeinsam mit der Abbildung im GUI definiert.

10.4.1 Szenario 1: Alles OK

Im ersten Szenario wird ein perfekter Prozessablauf abgebildet. Ein Trader erstellt einen neuen Deal auf der Handelsplattform der Börse RORC. Dieser Deal gelangt

über das Market Interface ins System, wird sowohl im Frontoffice, als auch im Backoffice kontrolliert und der Handelspartner bestätigt den Deal.

Anhand dieses Szenarios wird das grundsätzliche Zusammenspiel von Events, Regeln und GUI gezeigt.

10.4.1.1 Events

Dieses Szenario besteht aus fünf Events, jedes Service liefert eine *OK-Meldung* nach erfolgreicher Beendigung der Arbeit.

Diese Events werden in JMeter definiert und nacheinander versendet. Zwischen den Events ist eine *wait activity* die verhindert, dass alle Events auf einmal versendet werden.

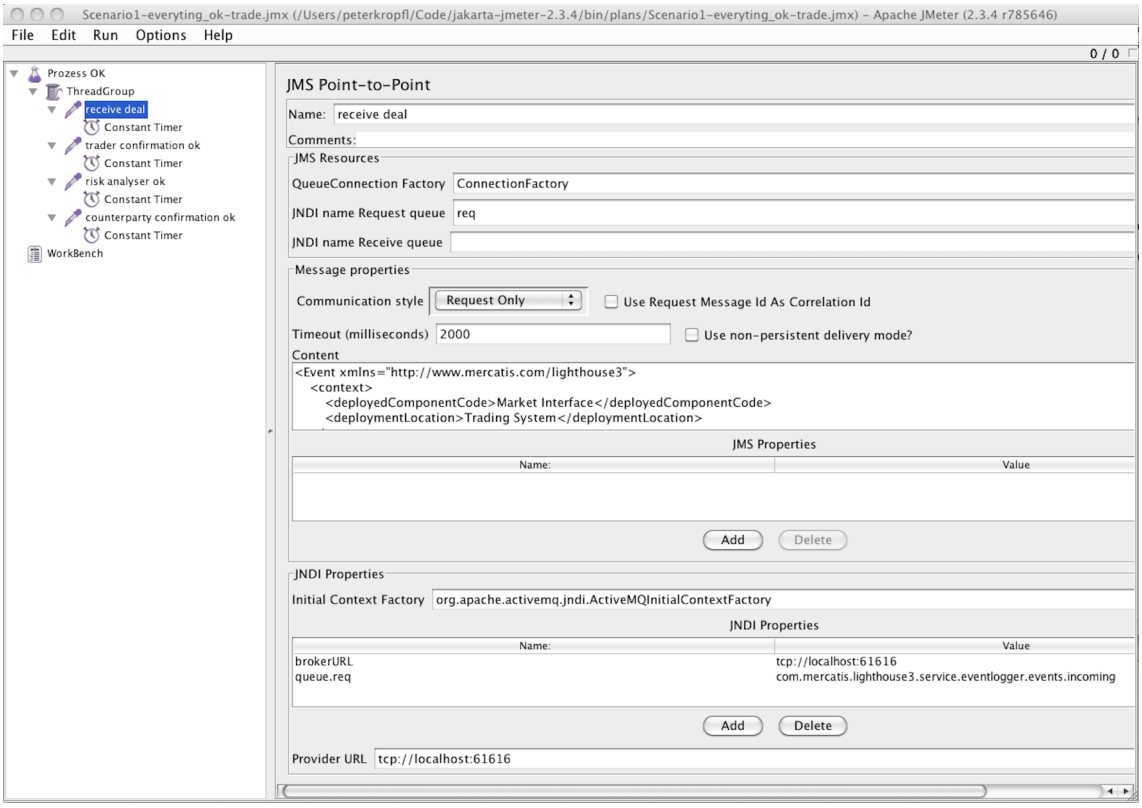

Abbildung 10.3: Szenario 1 Abbildung in JMeter

- **Receive Deal OK**

Sobald ein neuer Deal durch das Market Interface erfolgreich entgegengenommen wurde, versendet es folgendes Event.

- **code:** NewDealOK
- **message:** New Deal with DealID RO-123 received from RORC.
- **level:** INFO
- **deployedComponentCode:** Market Interfaces
- **deployementLocation:** Trading System
- **udf:** key=dealID, value=RO-123
- **udf:** key=market, value=RORC

Quellcode: B.2.1.1

- **Trader: Check Deal OK**

  Mithilfe des Trader Confirmation Service werden die einzelnen Deals durch den Trader bestätigt. Nach der erfolgreichen Bestätigung wird ein Event versendet, das den Erfolg anzeigt und die wichtigsten Parameter enthält.

  - **code:** TraderConfirmationOK
  - **message:** Trader kroepfl_p confirmed Deal with dealID: RO-123
  - **level:** INFO
  - **deployedComponentCode:** Trader Confirmation Service
  - **deploymentLocation:** Trading System
  - **udf:** key=dealID, value=RO-123
  - **udf:** key=trader, value=kroepfl_p

  Quellcode: B.2.1.2

- **Portfolio Risk Analyser: Deal OK**

  Im Backoffice wird der Deal durch den Portfolio Risk Analyser geprüft. Nach erfolgreicher Prüfung versendert der Portfolio Risk Analyser folgendes Event.

  - **code:** PortfolioRiskAnalyser
  - **message:** Portfolio risk check of deal RO-123 for Green Portfolio A was successful. Risklevel Medium; Exposure 2.324.230,12

  - **level:** INFO
  - **deployedComponentCode:** Portfolio Risk Analyser
  - **deploymentLocation:** BackOfficeSystem
  - **udf:** key=dealID, value=RO-123
  - **udf:** key=portfolio, value=Green Portfolio A
  - **udf:** key=trader, value=kroepfl_p
  - **udf:** key=risklevel, value=medium
  - **udf:** key=exposure, value=2.324.230,12

  Quellcode: B.2.1.3

- **Backoffice Deal Validation OK**

  Neben der Risikobewertung wird im Backoffice der Deal auf Vollständigkeit geprüft. Ist die Prüfung erfolgreich wird das folgende Event versendet. Es enthält zur Identifikation die Deal-ID.

  - **code:** BODealValidationOK
  - **message:** Backoffice deal validation for deal RO-123 was successful
  - **level:** INFO
  - **deployedComponentCode:** Backoffice Deal Validator
  - **deploymentLocation:** BackOfficeSystem
  - **udf:** key=dealID, value=RO-123

  Quellcode: B.2.1.4

- **Counterparty Confirmation**

  Bei erfolgreicher Bestätigung des Deals durch die Counterparty wird ein Event versendet, das die Basisdaten des Deals enthält.

  - **code:** CounterpartyConfirmationOK
  - **message:** Counterparty e.on trading ltd confirmed dealID RO-123
  - **level:** INFO
  - **deployedComponentCode:** Electronic Confirmation Service
  - **deploymentLocation:** BackOfficeSystem

- **udf:** key=counterparty, value=e.on trading ltd
- **udf:** key=duns, value=1234567
- **udf:** key=commodity, value=CO2 Cert.
- **udf:** key=price, value=100.134,20
- **udf:** key=volume, value=250
- **udf:** key=delivery, values=2010-03-01

Quellcode: B.2.1.5

10.4.1.2 CEP Regeln

Nicht alle Events können direkt im GUI herausgefiltert werden. Das zusammengesetzte Service *Backoffice Service* besteht aus zwei verschiedenen Services. Mithilfe der ersten beiden Regeln kann die CEP Engine aus den Events der darunter liegenden Services Events für das User Directory Service herausfiltern.

Um aber explizit zu zeigen, dass das Backoffice Service für einen bestimmten Deal ausgeführt wurde, ist es nötig auch einen zusätzlichen Vergleich der in den UDFs übergebenen Deal-IDs durchzuführen.

Diese Anforderung wurde in der dritten Regel abgebildet. Es werden die Deal-ID aus dem Event *Portfolio Risk Analyser: Deal OK* und dem Event *Backoffice Deal Validation OK* selektiert, wenn in einem Zeitfenster von 60 Sekunden beide Events eintreffen und beide die gleiche User-ID in ihren UDF Feldern tragen.

Die Selektion der User-ID und des Events ist für eine spätere Weiterverarbeitung des Events nötig.

```
1 select * from XMLEvent where code='PortfolioRiskAnalyserOK'
2 select * from XMLEvent where code='BODealValidationOK'
3 select portfolio.udfs.udf[1].value as portfolioID, validator.* as p
4         from
5           XMLEvent(code="PortfolioRiskAnalyserOK").win:time(60 sec)
                as portfolio,
6           XMLEvent(code="BODealValidationOK").win:time(60 sec) as
                validator
7           where
8           validator.udfs.udf[1].value = portfolio.udfs.udf[1].value
9 select * from XMLEvent where code='CounterpartyConfirmationOK'
```

Die fünfte Regel Filtert aus allen Events, die das erfolgreiche Prüfen eines Deals in einem Portfolio anzeigen, alle Events heraus, die die Prüfung für eines der Green Portfolios durchgeführt haben. Dies geschieht mittels des **regexp** Operators, der Stringmatching mittels Regular Expressions ermöglicht.

```
1 select * from XMLEvent where code='PortfolioRiskAnalyserOK' and udfs
    .udf[1].value regexp 'Green Portfolio*'
```

10.4.1.3 Abbildung GUI

Die einzelnen Arbeitsschritte färben sich grün, sobald die Events zu den zugeordneten Services eintreffen.

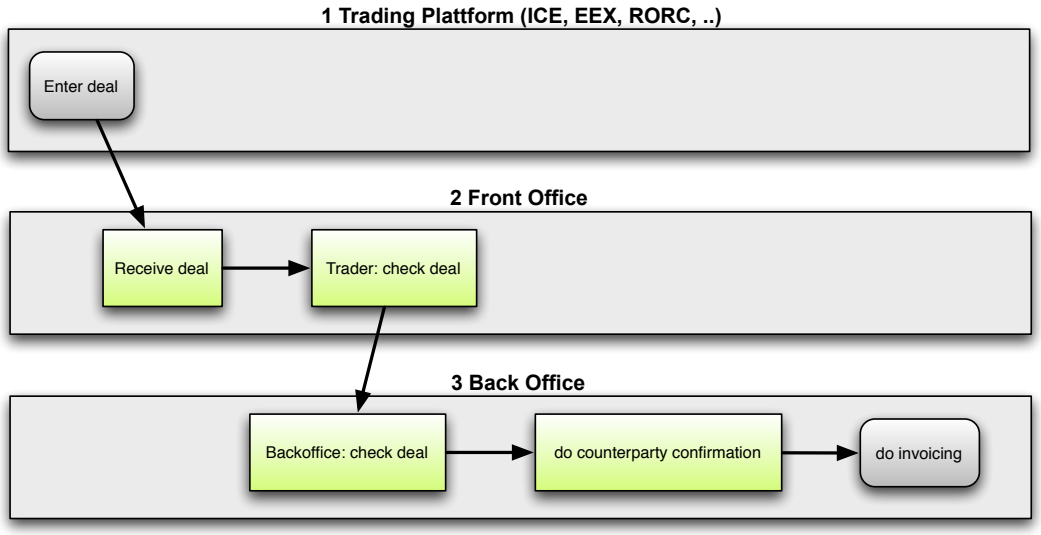

Abbildung 10.4: Szenario 1: Abbildung im GUI

10.4.2 Szenario 2: Fehlermeldung des Market Interface

Im zweiten Szenario wird vom Market Interface ein Fehler gemeldet. Es liefert ein Event zurück, das als Level *ERROR* und als Code *NewDealError* enthält.

In diesem Szenario wird gezeigt, wie Fehler von Services innerhalb des Prozesses gefiltert und visualisiert werden können. Dazu werden zwei unterschiedliche Events genutzt: Erstens, das Event *Receive Deal OK*, das bereits in Szenario 1 beschrieben wurde. Es wird mehrmals verschickt. Mittels des JMeter eigenen Zufallsgenerators wird ab und zu ein weiteres Event verschickt, das eine Fehlermeldung des Market Interface simuliert.

10.4.2.1 Events

- **Receive Deal Error** Sobald ein Deal nicht korrekt verarbeitet werden kann, wird folgendes Event erzeugt

- **code:** NewDealError
- **message:** Could not parse deal body from deal RO-111
- **level:** ERROR
- **deployedComponentCode:** Market Interface
- **deployementLocation:** Trading System
- **udf:** key=dealID, value=RO-111
- **udf:** key=market, value=RORC

Quellcode: B.2.1.6

10.4.2.2 CEP Regeln

```
1 select * from XMLEvent where context.deployedComponentCode='Market
    Interface'
2   and level='ERROR'
3   and code != 'CEP:MarketInterfaceError'
```

Die CEP Regel filtert alle Events, die vom Market Interface versendet werden und das Level ERROR haben. Um die Events, die durch die CEP Engine erzeugt wurden nicht erneut zu filtern, werden diese explizit ausgenommen.

10.4.2.3 Abbildung GUI

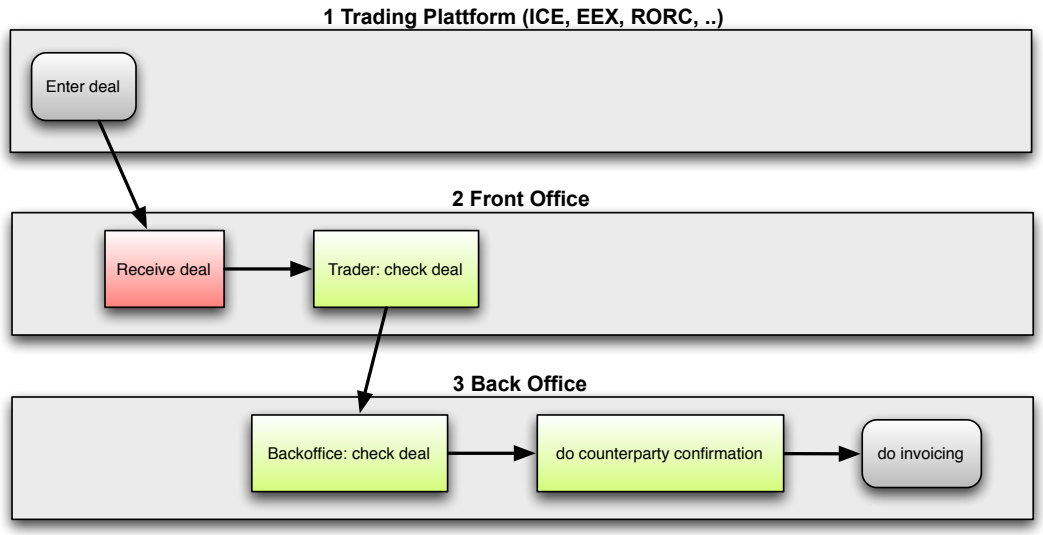

Abbildung 10.5: Szenario 2: Fehler im Market Interface

10.4.3 Szenario 3: Fehlermeldung eines tieferliegenden Service

In diesem Szenario vermeldet ein indirekt genutztes Service einen Fehler. Es handelt sich dabei um das Service Backoffice Deal Validator, das einen Systemfehler bei der Verbindung zur Datenbank meldet.

Im dritten Szenario werden zwei verschiedene Anforderungen an das Monitoring untersucht. Die erste Anforderung ist die Abbildung und Meldung von Systemfehlern. Die zweite Herausforderung, die hier untersucht wird ist die Verarbeitung von Fehlern aus zusammengesetzten Services.

Anhand dieses Szenarios wird ein wichtiger Baustein zum Monitoring von serviceorientierten Architekturen untersucht: Das Monitoring von wiederverwendbaren Services in unterschiedlichen Kontexten.

10.4.3.1 Events

- **Deal Validation Error**

  Das Employee Data Service meldet beim Anlegen einen Fehler zurück.

  - **code:** UniqueKeyContraintViolation
  - **message:** ORA-0001: Unique Key Contraint Violation: DEAL_ID
  - **level:** ERROR
  - **deployedComponentCode:** Deal Validator
  - **deployementLocation:** BackOfficeSystem

  Quellcode: B.2.1.7

10.4.3.2 CEP Regeln

```
1 select * from XMLEvent where context.deployedComponentCode='Deal
    Validator' and level='ERROR'
```

Der Filter selektiert alle Events, die vom Deal Validator Service ausgesendet werden und das Level ERROR haben. Aus dieser Meldung wird in der CEP Engine dann ein neues Event erzeugt, das als context das Backoffice Service kon-

figuriert hat.

10.4.3.3 Abbildung GUI

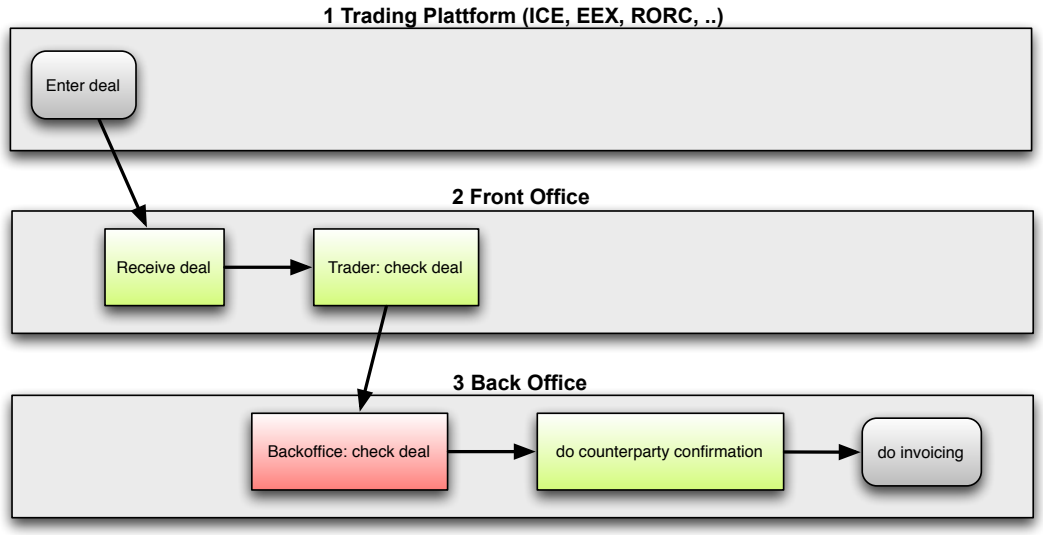

Abbildung 10.6: Szenario 3: Fehler in Composite Service

10.4.4 Szenario 4: Maximale Durchlaufzeit überschritten

Im vierten Szenario wird gezeigt, wie mit den hier entwickelten Ansätzen, Vorher/Nachher-Beziehungen, sowie zeitliche Abhängigkeiten abgebildet werden können.

In diesem Beispiel wird die maximale Durchlaufzeit des Prozesses überschritten, indem ein erwartetes Event in der Prozesskette ausbleibt. Als Event wird ein OK des Portfolio Risk Analysers erwartet, das als Code *PortfolioRiskAnalyserOK* und als UDF ein String Feld mit der Deal-ID des Deals enthält.

10.4.4.1 Events

Für dieses Szenario werden die gleichen Events, wie in Szenario 1 (10.4.1) verwendet. Der einzige Unterschied ist, dass das vorletzte Event mit deutlicher Verspätung versendet wird. Dazu wird in JMeter der *Constant Time Sampler* auf 10 Sekunden gesetzt.

10.4.4.2 CEP Regeln

Zur Abbildung von Verspätungen in Regeln ist es nötig Zeitdauern anzugeben. Dies wird in der EPL mit Hilfe des Timer Schlüsselworts *timer:interval(3 sec)* umgesetzt.

Die folgende Regel Filtert alle Events, die innerhalb von 3 Sekunden dem Event mit dem Code *NewDealOK* folgen. Es werden aber nur jene Events selektiert, die als Code *PortfolioRiskAnalyserOK* und den gleichen Wert als User-ID in den UDFs tragen. Auf diese Weise wird sichergestellt, dass keine Events, die zu unterschiedlichen Prozessinstanzen gehören, das Ergebnis beeinflussen.

```
1 select * from pattern [every start=XMLEvent(code="NewDealOK") ->
2       (timer:interval(3 sec) and not done=XMLEvent(code="
            PortfolioRiskAnalyserOK", udfs.udf[0].value = start.udfs.
            udf[0].value))]
```

10.4.4.3 Abbildung GUI

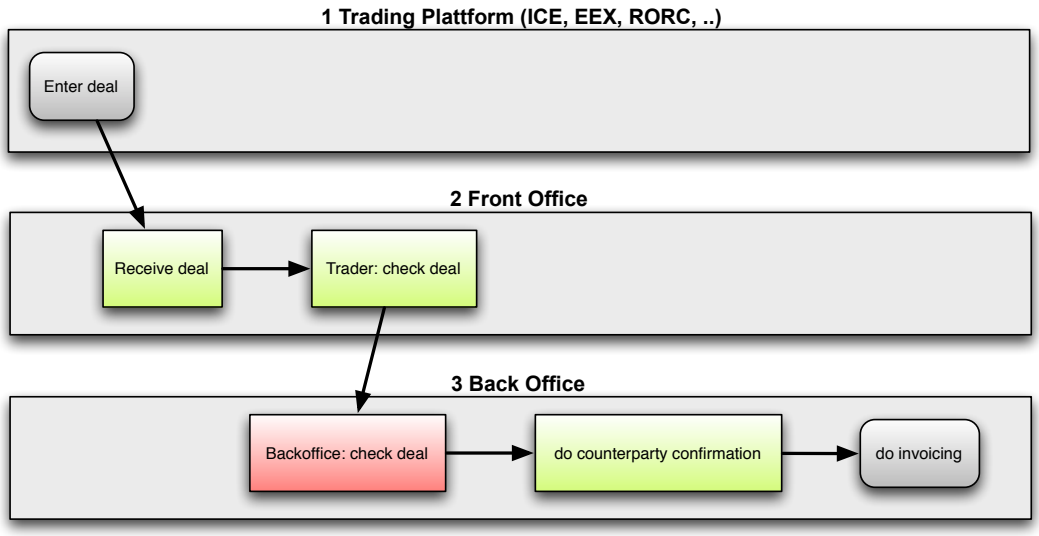

Abbildung 10.7: Szenario 4: Verspätung im Prozess

10.4.5 Szenario 5: Weiterverarbeitung von Basis-Eventdaten aus komplexen Events

Eine typische Anforderung, die auftritt sobald Prozessinstanzen überwacht werden sollen, ist der Zugriff auf Eventparameter, wie etwa User-IDs oder Transak-

tionsnummern. Durch die Zuhilfenahme dieser Parameter ist es möglich einen gemeinsamen Kontext herzustellen, der die Abbildung von Prozessinstanzen ermöglicht.

Im Szenario 1 (10.4.1) wurden zwei verschiedene Events korreliert und so ein neues Event erzeugt. Um die Daten dieser Events in weiteren Anfragen nutzen zu können, ist es nötig sie an einer bestimmten Stelle zu speichern.

Esper bietet dafür die Möglichkeit zusätzliche Streams zu erzeugen und dort selektierte Events einzufügen. Dies geschieht mit Hilfe des *insert into* Statements.

Die vierte Regel aus Szenario 1 kann um dieses Statement erweitert werden um den folgenden Fall abzubilden:

Nach dem Eintreffen eines neuen Deals und der Validierung durch den Trader muss innerhalb von 3 Sekunden das Backoffice den Deal validieren. Ist das nicht der Fall soll gewarnt werden.

Der erste Teil der Anforderung kann bereits durch eine Regel analog zu Szenario 1 abgedeckt werden. Sobald zwei bestimmte Events innerhalb von 60 Sekunden auftreten und zum gleichen Deal gehören, erzeugt die Engine ein neues Event. Um aber auf die grundlegenden Eventdaten weiterhin zugreifen zu können, müssen diese in einem neuen Stream (*MemoryStream*) gepublisht werden.

```
1 insert into MemoryStream
2       select deal.udfs.udf[1].value as dealId, traderConfirmation.*
            as t
3             from
4               XMLEvent(code="NewDealOK").win:time(60 sec) as deal,
5               XMLEvent(code="TraderConfirmationOK").win:time(60
                    sec) as traderConfirmation
6               where
7               deal.udfs.udf[1].value = traderConfirmation.udfs.udf
                    [1].value
```

Dieser Stream kann dann selbst wieder gefiltert werden. So ist es möglich den zweiten Teil der Anforderung abzudecken. Dazu wird folgender Filter definiert: Selektiere alle Events, die im *MemoryStream* mit Code *TraderConfirmationOK* gefunden werden, denen kein Event mit Code *DealvalidationOK* und der gleichen User-ID in den UDFs nachfolgt.

```
1 select evt.* from pattern [every mem=MemoryStream(t.code="
    TraderConfirmationOK")
2         -> (timer:interval(3 sec) and not
3           evt=XMLEvent(code="DealValidationOK", evt.udfs.udf[1].
                value=mem.t.udfs.udf[1].value))]
```

10.5 Durchführung

In diesem Kapitel wird der Test der zuvor beschriebenen Szenarios beschrieben. Analog dem Vorgehen bei den Funktionstests, werden für die einzelnen Szenarios Testfälle definiert. Diese Testfälle enthalten aber keine ausführliche Beschreibung, sondern lediglich eine Kurzbeschreibung mit einer Referenz zu den entsprechenden Passagen im Kapitel 10.4, da sie dort schon extensiv beschrieben wurden.

10.5.1 Technische Umsetzung

Technisch werden die Tests der einzelnen Szenarios wie die bisherigen Tests (vgl. Kapitel 9) aufgesetzt. Die Generierung der einzelnen Events wird mittels JMeter umgesetzt. Die Validierung der Ergebnisse erfolgt über einen Vergleich der (CEP-) Events, die bei den Tests in der Datenbank gespeichert wurden.

10.5.2 Testfälle

10.5.2.1 Test Szenario 1

- **ID** TS1
- **Name** Test Szenario 1
- **Beschreibung** In diesem Test wird das Szenario 1 (10.4.1) 400 Mal nacheinander ausgeführt.
- **Input** JMeter-Plan, der folgende Events 400 Mal nacheinander ausführt: B.2.1.1, B.2.1.3, B.2.1.4, B.2.1.5, B.2.1.5
- **Erwartetes Ergebnis** Drei Ergebnisse werden erwartet:
  1. Alle Events, die mittels JMeter erzeugt wurden sind in der Eventdatenbank zu finden (2000 Events)
  2. Alle CEP-Events, die in der CEP-Konfiguration (B.2.2.1) definiert wurden sind ebenfalls in der Eventdatenbank. (400 CEP:PortfolioRiskAnalyserOK, 400 CEP:BODealValidationOK, 400 CEP:ValidationOK, 400 CEP:GreenportfolioRiskOK)
  3. Alle Arbeitsschritte im GUI sind grün.

10.5.2.2 Test Szenario 2

- **ID** TS2
- **Name** Test Szenario 2
- **Beschreibung** Im zweiten Szenario wird getestet ob Fehlermeldungen korrekt erkannt und behandelt werden.
- **Input** JMeter-Plan, der folgende Events ausführt: B.2.1.1, B.2.1.6. Dabei wird mithilfe des Loop-Controllers das Event *NewDealOK* 300 Mal ausgeführt und des Event *NewDealError* 30 Mal.
- **Erwartetes Ergebnis** Drei Ergebnisse werden erwartet:
  1. Alle Events, die mittels JMeter erzeugt wurden sind in der Eventdatenbank zu finden. (300 *NewDealOK*, 30 *NewDealError*)
  2. Alle CEP-Events, die in der CEP-Konfiguration (B.2.2.2) definiert wurden sind ebenfalls in der Eventdatenbank. (30 *CEP:MarketInterfaceError*)
  3. Alle Arbeitsschritte im GUI sind grün, bis auf den ersten Schritt. Dieser wird rot, sobald ein das Fehler-Event B.2.1.6 auftritt.

10.5.2.3 Test Szenario 3

- **ID** TS3
- **Name** Test Szenario 3
- **Beschreibung** Im dritten Test wird getestet ob Fehler, die tief in zusammengesetzten Services auftreten korrekt erkannt und dargestellt werden.
- **Input** JMeter-Plan der das Event *DealValidationError* (B.2.1.7) versendet.
- **Erwartetes Ergebnis**
  1. Alle Events, die mittels JMeter erzeugt wurden sind in der Eventdatenbank zu finden. (1 DealValidationError)
  2. Alle CEP-Events, die in der CEP-Konfiguration (B.2.2.3) definiert wurden sind ebenfalls in der Eventdatenbank. (1 CEP:DealValidationError)
  3. Alle Arbeitsschritte im GUI sind grün, bis auf den Schritt *Backoffice: check deal*. Dieser wird rot, sobald ein das Fehler-Event B.2.1.6 auftritt.

10.5.2.4 Test Szenario 4

- **ID** TS4
- **Name** Test Szenario 4
- **Beschreibung** Im vierten Szenario wird überprüft ob Verspätungen erkannt werden können. Dabei wird durch die CEP-Regeln definiert, dass das Event *ProtfolioRiskAnalysisOK* (B.2.1.3) innerhalb von drei Sekunden auf das Event *NewDealOK* (B.2.1.3) folgen muss. Im JMeter-Plan wird durch einen Timersampler absichtlich eine Verzögerung eingebaut um die Verspätung zu simulieren.
- **Input** Events: B.2.1.3, B.2.1.3
- **Erwartetes Ergebnis**
  1. Alle Events, die mittels JMeter erzeugt wurden sind in der Eventdatenbank zu finden. (2 Events)
  2. Alle CEP-Events, die in der CEP-Konfiguration (B.2.2.4) definiert wurden sind ebenfalls in der Eventdatenbank. (1 CEP:PortfolioRiskAnalysisLATE)
  3. Alle Arbeitsschritte im GUI sind grün, bis auf den Schritt *Backoffice: check deal*. Dieser wird rot, sobald die Verspätung eintritt.

10.5.2.5 Szenario 5

- **ID** TS5
- **Name** Szenario 5
- **Beschreibung** Im fünften Test wird geprüft, ob es möglich ist Eventdaten aus bereits durch die CEP-Engine verarbeiteten Events weiterzunutzen. Dazu wird das Szenario 1 erweitert. Den bestehenden Konfigurationen und Events wird eine weitere Konfiguration hinzugefügt. Diese prüft, ob innerhalb eines Zeitfensters von 60 Sekunden dem Event *TraderConfirmationOK* das zu der selben Prozessinstanz gehörige Event *DealValidationOK* eintritt. Sollte dieses ausbleiben wird von der vom Monitoring-System das Event *CEP:DealValidationMissing* erzeugt. Die Umsetung erfolgt mithilfe von JMeter. Es werden wieder alle Events aus Szenario 1 nacheinander versendet, jedoch wird auf das Event *Backoffice Deal Validation* (B.2.1.4) verzichtet.

- **Input** Events: B.2.1.1, B.2.1.3, B.2.1.5, B.2.1.5
- **Erwartetes Ergebnis**
  1. Alle Events, die mittels JMeter erzeugt wurden sind in der Eventdatenbank zu finden. (4 Events)
  2. Alle CEP-Events, die in der CEP-Konfiguration (B.2.2.4) definiert wurden sind ebenfalls in der Eventdatenbank. (1 CEP:PortfolioRiskAnalyserOK, 1 CEP: DealValidationLATE)
  3. Alle Arbeitsschritte im GUI sind grün, bis auf den Schritt *Backoffice: check deal*. Dieser wird rot, sobald die Verspätung eintritt.

10.6 Ergebnisse

Sämtliche Szenarios wurden in Testdurchläufen erfolgreich getestet. Es zeigte sich, dass es möglich ist mit dem hier entwickelten Ansatz die Herausforderungen, die an Prozessmonitoring gestellt werden zu meistern.

10.6.1 Abdeckung Anforderungen des Business Case

Die Anforderungen, die in diesem Business-Case an das Monitoring gestellt wurden, konnten erfolgreich abgedeckt werden.

- **Erkennen von Bedienungs- und Systemfehlern**

  Sowohl Bedienungsfehler, als auch Systemfehler erzeugen Anomalien in den Daten. In dem zweiten Szenario wurde künstlich ein Fehler am DealInterface simuliert. Dieser Fehler wurde durch das Monitoring erkannt, verarbeitet und in einem übersichtlichen GUI dargestellt.

  Ebenso wurde gezeigt, dass auch sehr technische Probleme durch die Definition von entsprechenden Regeln für die Verantwortlichen im Business "übersetzt" werden konnten, indem sie dem konkreten Prozessschritt zugeordnet wurden. So kann die Auswirkung dargestellt und die betroffenen Personen kontaktiert werden.

- **Einhalten von maximalen Durchlaufzeiten**

  Durch konsequente Kontrolle der Durchlaufzeiten auf Ebene jeder Prozessinstanz kann sichergestellt werden, dass sofort eingegriffen werden kann, sobald Durchlaufzeiten auch in Teilprozessen überschritten werden. Dadurch können Hotspots in der Prozesskette erkannt und analysiert werden.

Im Szenario 4 wird beispielsweise die Dauer der Ausführung des Teilprozesses *Backoffice Deal Validation* überwacht. Bei Problemen kann sofort eingegriffen werden. Durch die Überwachung auf Prozessinstanz-Ebene ist es zudem möglich direkt zu sehen, welcher Deal die Überschreitung hervorgerufen hat.

- **Erkennen von *"verlorenen Deals"***

  Das Monitoring auf Ebene der Prozessinstanz erlaubt die durchgängige Verfolgung von Deals durch die Handelssysteme. Im fünften Szenario wurde gezeigt, dass es dadurch möglich ist zu prüfen, ob alle Deals sämtliche Schritte der Prozesskette erfolgreich durchlaufen haben. Im konkreten Beispiel bleibt für einen Deal die Validierung aus und diese Prozessinstanz wird nicht weiter ausgeführt. Dies kann durch das Monitoring erkannt und dargestellt werden kann.

10.6.2 Vorteile dieses Monitoring-Ansatzes

In dieser Case-Study werden die vielen Vorteile die dieser Monitoring-Ansatz bringt deutlich. In den folgenden Absätzen werden die wichtigsten zusammengefasst.

- **Integration unterschiedlicher Systeme**

  Ein Vorteil den dieser Monitoring-Ansatz bietet, ist die Integration von unterschiedlichen Systemen. Es ist nicht mehr nötig, jedes einzelne Service und jede Applikation extra zu überwachen. Sämtliche relevante Monitoring-Information ist über eine Schnittstelle verfügbar.

- **Unmittelbares Feedback**

  Ein weiterer wichtiger Vorteil ist das direkte und aktive Feedback. Meldungen werde nicht mehr in versteckten Logfiles abgelegt, die mühevoll von Experten analysiert werden müssen. Die ersten beiden Szenarios zeigen, wie Fehler- und OK-Meldungen sofort analysiert, verarbeitet und dargestellt werden um den Verantwortlichen **live** den Zustand ihrer Prozesse zeigen zu können.

- **Genaue Lokalisation des Problems**

  Ein weiterer großer Vorteil ist die Möglichkeit Probleme exakt zu lokalisieren. Üblicherweise müssen sämtliche beteiligten Systeme untersucht werden, um Fehler zu finden. Je verteilter und komplexer diese Systeme werden,

umso schwieriger ist die Fehlersuche. In den Szenarios wird durch Überwachung auf Prozessebene gezeigt, wie Deals durch die einzelnen Systeme verfolgt werden können. Dadurch ist eine genaue Lokalisation der Ursache eines Problems möglich.

- **Überwachung der Prozesskette**

  Dieser Monitoring-Ansatz bietet die Möglichkeit durch Vorher-Nachher-Beziehungen verschiedene Services und Systeme entlang der Prozesskette miteinander in Beziehung zu setzen. So kann auch das Ausbleiben von Information in der Prozesskette erkannt werden. Dies ist ein großer Vorteil gegenüber herkömmlichen Formen des Monitorings, da dies typischerweise nur innerhalb einer Applikation möglich ist.

- **Sichtbarkeit der Auswirkungen**

  Monitoring von Prozessen bietet den Vorteil, dass Auswirkungen von Problemen sichtbar werden. Es ist dadurch auch für Personen, die mit dem Gesamtprozess nicht im Detail vertraut sind möglich, Aussagen über Auswirkungen für Personen und Systeme zu machen.

- **Abbildung von geschäftsrelevanten Anforderungen**

  Einer der größten Vorteile dieses Ansatzes ist Möglichkeit geschäftsrelevante Anforderungen abzubilden. Szenario 4 zeigt hier zum Beispiel wie geschäftskritische Zeitdauern über verschiedene Systeme hinweg auf Dealebene überwacht werden können. So kann in diesem Beispiel sichergestellt werden, dass kein Deal übersehen wird und die Verarbeitung rechtzeitig beendet werden kann.

Kapitel 11

Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, dass es möglich ist Prozessmonitoring in serviceorientierten Architekturen mit Hilfe von Complex Event Processing durchzuführen. Damit ist es nicht mehr nötig Prozesse in Execution Engines auszuführen um sie überwachen zu können.

Auf diese Art ist es möglich Prozesse, die nicht in BPM Systemen abgebildet sind oder gar nicht in solchen Systemen abgebildet werden können oder sollen, zu monitoren. Hinzu kommt, dass durch den hier entwickelten Ansatz diese Prozesse leichter anpassbar sind, da lediglich der Output der einzelnen Services die Quelle des Monitorings darstellt und so der Grad an Kopplung äußerst gering ist.

Der hier vorgestellte Ansatz zeigt Lösungswege für die typischen Probleme von Monitoring in serviceorientierten Architekturen auf. Zu diesen Problemen zählen etwa die Verteilung der Services, sowie die Verschiedenartigkeit der Daten, die Abbildung von Prozessflüssen oder auch die Wiederverwendung von Services in unterschiedlichen Geschäftskontexten.

Die Überwachung von einzelnen Prozessinstanzen ist nach wie vor ein großes Problem.

In den beiden letzten Szenarios der Case-Study wurde deutlich, dass die Komplexität der Regeln für das Monitoring von Prozessinstanzen sehr groß werden kann. Speziell, wenn viele verschiedene, zusammengesetzte Services am Prozess beteiligt sind, ist es schwierig diese einem gemeinsamen Kontext zuzuordnen.

Zwei Probleme sind dafür ausschlaggebend. Im hier vorgestellten Prototypen ist es nicht möglich Daten eines Events, das durch eine Regel gefiltert wurde direkt an das erzeugte Event anzuhängen. Gerade dies würde aber das Handling von zusammengesetzten Services deutlich erleichtern. Durch die Verwendung von Variablen die sowohl in der Konfiguration der Events, als auch in der Definition

von Regeln gemeinsam genutzt werden können, wäre es sicherlich möglich direkt Daten zu übergeben. Ein anderer Ansatz wäre das gesamte Event, das eine Regel feuern ließ, an das daraus resultierende Event direkt anzuhängen.

Das zweite Problem ist die Zusammenfassung von verschiedenen Services zu einem Prozess. Verschiedene Services, die an einem Prozessfluss beteiligt sind, werden mit sehr großer Wahrscheinlichkeit keine gemeinsame Transaktions-ID haben. Viel wahrscheinlicher ist der Fall, dass unterschiedliche Services gar keine oder unterschiedliche Transaktions-IDs haben. Eindeutige Transaktionen zu erzeugen und über diese einen gemeinsamen Kontext zu erstellen ist eine Aufgabe, die für jeden Prozess einzeln geregelt werden muss. Eine Zuordnung zu einem gemeinsamen Kontext und damit zu einer konkreten Prozessinstanz ohne über Transaktions-IDs den gemeinsamen Kontext aufzuspannen ist ein weiterhin offenes Thema.

Neben diesen offenen Punkten hat sich gezeigt, dass speziell die Regelsprache sehr kompliziert ist und sehr viele Möglichkeiten bietet. Hier wäre eine Vereinfachung mit Sicherheit von großem Vorteil. Möglich wäre zum Beispiel eine abgeleitete Sprache, die für typische Anwendungen bereits vorgefertigte Bausteine anbietet. Eine andere Möglichkeit wäre die Ableitung von Regeln und Regelbestandteilen aus den Prozessmodellen. Vorher/Nachher - Beziehungen könnten beispielsweise direkt aus dem Modell abgeleitet werden.

Ein Punkt der in der prototypischen Implementierung außen vor gelassen wurde ist die Ausfallsicherheit. Da die hier entwickelten Werkzeuge dem Monitoring dienen, sind die Anforderungen an die Verfügbarkeit entsprechend hoch. Für den Einsatz in einem realen Umfeld ist deshalb auch ein zuverlässiges Konzept für die Themen Lastverteilung und Ausfallsicherheit Grundvoraussetzung.

Dieser Punkt und die direkte Parameterübergabe zwischen ausgewerteten und erzeugten Events sind die Bereiche denen in der nächsten Zeit mit Sicherheit am meisten Beachtung geschenkt wird um aus dem Prototypen ein Produkt zu schaffen.

Anhang A

Lebenslauf

A.1 Akademischer Werdegang

1988-1992	Volksschule, Kath. Privatschule mit Öffentlichkeitsrecht Institut Neulandschulen, Wien 10
1992-2000	Gymnasium (neusprachlicher Zweig), Kath. Privatschule mit Öffentlichkeitsrecht Institut Neulandschulen, Wien 10
2001-2006	Bakkalaureat Wirtschaftsinformatik TU-Wien und Universität Wien
2006-2010	Master Wirtschaftsinformatik TU-Wien und Universität Wien, Abschluss Universität Wien

A.2 Beruflicher Werdegang

2000	DSI, Mitarbeit Testteam Bank Austria Wertpapiersysteme
2000-2001	Wiener Sozialdienste, Zivildienst, Basale Förderklassen
2001	Bank Austria, Support Software-Migration
2002-2006	Snowboardlehrer
2006-aktuell	IT Consultant, Mercatis Consulting GmbH

Anhang B

Testdaten

B.1 Testing

B.1.1 Grundfunktionalität

B.1.1.1 GF2

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16 </cepevents>
```

B.1.1.2 GF3

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
```

```
6         <code>Generated CEP Code</code>
7         <level>ERROR</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP error event for a basic event</
              message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16 </cepevents>
```

B.1.2 Hotloader CEP Konfiguration

B.1.2.1 KO1

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16 </cepevents>
```

B.1.2.2 KO2

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
```

```
11              <deployedComponentCode>CEP-Event 1</
                    deployedComponentCode>
12              <deploymentLocation>CEP Engine</deploymentLocation>
13            </context>
14      </event>
15    </cepevent>
16    <cepevent>
17      <query>select * from XMLEvent where code='Basic Event 2'</query>
18      <event>
19          <code>CEP-Filter-Basic-Event-2</code>
20          <level>INFO</level>
21          <machineOfOrigin>ftibco99</machineOfOrigin>
22          <message>Filtered Basic Event 2 from many other events and
                generated a CEP Event</message>
23          <context>
24              <deployedComponentCode>CEP-Event 1</
                    deployedComponentCode>
25              <deploymentLocation>CEP Engine</deploymentLocation>
26            </context>
27      </event>
28    </cepevent>
29 </cepevents>
```

B.1.2.3 KO5

```
 1 <?xml version="1.0" encoding="UTF-8" ?>
 2 <cepevents>
 3   <cepevent>
 4     <query>select * from XMLEvent where code='Basic Event 1'</query>
 5     <event>
 6         <code>Generated CEP Code</code>
 7         <level>INFO</level>
 8         <machineOfOrigin>ftibco99</machineOfOrigin>
 9         <message>Generated a CEP event for a basic event</message>
10         <context>
11             <deployedComponentCode>CEP-Event 1</
                   deployedComponentCode>
12             <deploymentLocation>CEP Engine</deploymentLocation>
13           </context>
14     </event>
15   </cepevent>
16   <cepevent>
17     <query>select * from XMLEvent where code='Basic Event 2'</query>
18     <event>
19         <code>CEP-Filter-Basic-Event-2</code>
20         <level>INFO</level>
21         <machineOfOrigin>ftibco99</machineOfOrigin>
22         <message>Filtered Basic Event 2 from many other events and
```

```
            generated a CEP Event</message>
23        <context>
24            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
25            <deploymentLocation>CEP Engine</deploymentLocation>
26          </context>
27    </event>
28  </cepevent>
29  <cepevent>
30    <query>select * from XMLEvent where code='Basic Event 2'</query>
31    <event>
32      <code>CEP-Filter-Basic-Event-2</code>
33      <level>INFO</level>
34      <machineOfOrigin>ftibco99</machineOfOrigin>
35      <message>Filtered Basic Event 2 from many other events and
            generated a CEP Event</message>
36      <context>
37        <deployedComponentCode>CEP-Event 1</deployedComponentCode>
38        <deploymentLocation>CEP Engine</deploymentLocation>
39      </context>
40    </event>
41  </cepevent>
42 </cepevents>
```

B.1.2.4 KO6

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
```

B.1.2.5 KO7

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>s * from XMLEvent where code='Basic Event 1'</query>
5     <event>
```

```
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16 </cepevents>
```

B.1.2.6 KO8

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7
8     </event>
9   </cepevent>
10 </cepevents>
```

B.1.2.7 KO9

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5
6   </cepevent>
7 </cepevents>
```

B.1.2.8 KO10

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <event>
5         <code>Generated CEP Code</code>
6         <level>INFO</level>
7         <machineOfOrigin>ftibco99</machineOfOrigin>
8         <message>Generated a CEP event for a basic event</message>
9         <context>
10            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
```

```
11              <deploymentLocation>CEP Engine</deploymentLocation>
12            </context>
13      </event>
14    </cepevent>
15 </cepevents>
```

B.1.2.9 KO11

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='Basic Event 1'</query>
5     <event>
6         <code>Generated CEP Code</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Generated a CEP event for a basic event</message>
10        <context>
11            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
12            <deploymentLocation>CEP Engine</deploymentLocation>
13          </context>
14    </event>
15  </cepevent>
16  <!--
17  <cepevent>
18    <query>select * from XMLEvent where code='Basic Event 2'</query>
19    <event>
20        <code>CEP-Filter-Basic-Event-2</code>
21        <level>INFO</level>
22        <machineOfOrigin>ftibco99</machineOfOrigin>
23        <message>Filtered Basic Event 2 from many other events and
              generated a CEP Event</message>
24        <context>
25            <deployedComponentCode>CEP-Event 1</
                  deployedComponentCode>
26            <deploymentLocation>CEP Engine</deploymentLocation>
27          </context>
28    </event>
29  </cepevent>
30  -->
31 </cepevents>
```

B.1.2.10 QL1

```
1 <cepevent>
2   <query>
```

```
3     select start.* as s, done.* as d from pattern [every start=
          XMLEvent(code="AddNewEmployee") -> done=XMLEvent(code="
          AddNewEmployeeDone")]
4   </query>
5   <event>
6     <code>CEP-Filter-Advanced</code>
7     <level>INFO</level>
8     <machineOfOrigin>ftibco99</machineOfOrigin>
9     <message>
10      A done event followed a start event.
11    </message>
12    <context>
13      <deployedComponentCode>CEP-Event 1</deployedComponentCode>
14      <deploymentLocation>CEP Engine</deploymentLocation>
15    </context>
16  </event>
17 </cepevent>
```

B.1.2.11 QL2

```
1 <cepevent>
2   <query>
3     select * from pattern [every start=XMLEvent(code="AddNewEmployee
          ") -> (timer:interval(2 sec) and XMLEvent(code="
          AddNewEmployee"))]
4   </query>
5   <event>
6     <code>CEP-Filter-Advanced</code>
7     <level>INFO</level>
8     <machineOfOrigin>ftibco99</machineOfOrigin>
9     <message>
10      Event with code "AddNewEmployee" was detected 2 times within 2
            seconds
11    </message>
12    <context>
13      <deployedComponentCode>CEP-Event 1</deployedComponentCode>
14      <deploymentLocation>CEP Engine</deploymentLocation>
15    </context>
16  </event>
17 </cepevent>
```

B.1.2.12 QL3

```
1 <cepevent>
2     <query>
3       select * from pattern [every start=XMLEvent(code="
            AddNewEmployee") ->
4       (
```

```
5           timer:interval(2 sec) and not done=XMLEvent(code="
                AddNewEmployeeDone", transactions.transactionId[0] =
                start.transactions.transactionId[0])
6         )]
7       </query>
8       <event>
9         <code>CEP-Filter-Advanced</code>
10        <level>INFO</level>
11        <machineOfOrigin>ftibco99</machineOfOrigin>
12        <message>
13          A done event did not follow a start event within 2sec
14        </message>
15        <context>
16          <deployedComponentCode>CEP-Event 1</deployedComponentCode>
17          <deployedComponentLocation>CEP Engine</deployedComponentLocation>
18        </context>
19      </event>
20    </cepevent>
```

B.1.2.13 QL4

```
1 <cepevent>
2     <query>
3       select * from XMLEvent() where cast(transactions.transactionId
            [0],int) > 1001
4     </query>
5     <event>
6       <code>CEP-Filter-Advanced</code>
7       <level>INFO</level>
8       <machineOfOrigin>ftibco99</machineOfOrigin>
9       <message>
10        Detected event with transactionID > 1001
11      </message>
12      <context>
13        <deployedComponentCode>CEP-Event 1</deployedComponentCode>
14        <deployedComponentLocation>CEP Engine</deployedComponentLocation>
15      </context>
16    </event>
17  </cepevent>
```

B.2 Case Study

B.2.1 Eventdefinitionen

B.2.1.1 New Deal OK

```
1 <Event xmlns="http://www.mercatis.com/lighthouse3">
```

```
2     <context>
3         <deployedComponentCode>Market Interface</
              deployedComponentCode>
4         <deploymentLocation>Trading System</deploymentLocation>
5     </context>
6     <code>NewDealOK</code>
7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
          dateOfOccurrence>
8     <level>INFO</level>
9     <machineOfOrigin>TradingSystem-node1</machineOfOrigin>
10    <message>New Deal with DealID RO-123 received from RORC.</
          message>
11    <udfs>
12        <udf>
13            <key>dealID</key>
14            <value>RO-123</value>
15            <type>string</type>
16        </udf>
17        <udf>
18            <key>market</key>
19            <value>RORC</value>
20            <type>string</type>
21        </udf>
22    </udfs>
23 </Event>
```

B.2.1.2 Trader Confirmation OK

```
1 <Event xmlns="http://www.mercatis.com/lighthouse3">
2     <context>
3         <deployedComponentCode>Trader Confirmation Service</
              deployedComponentCode>
4         <deploymentLocation>Trading System</deploymentLocation>
5     </context>
6     <code>TraderConfirmationOK</code>
7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
          dateOfOccurrence>
8     <level>INFO</level>
9     <machineOfOrigin>TradingSystem-node1</machineOfOrigin>
10    <message>Trader kroepfl_p confirmed Deal with dealID: RO-123</
          message>
11    <udfs>
12        <udf>
13            <key>dealID</key>
14            <value>RO-123</value>
15            <type>string</type>
16        </udf>
17        <udf>
```

```
18              <key>trader</key>
19              <value>kroepfl_p</value>
20              <type>string</type>
21          </udf>
22      </udfs>
23 </Event>
```

B.2.1.3 Portfolio Risk Analysis OK

```
 1 <Event xmlns="http://www.mercatis.com/lighthouse3">
 2     <context>
 3         <deployedComponentCode>Portfolio Risk Analyser</
               deployedComponentCode>
 4         <deploymentLocation>BackOfficeSystem</deploymentLocation>
 5     </context>
 6     <code>PortfolioRiskAnalyserOK</code>
 7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
           dateOfOccurrence>
 8     <level>INFO</level>
 9     <machineOfOrigin>TradingSystem-node1</machineOfOrigin>
10     <message>Portfolio risk check of deal RO-123 for Green Portfolio
           A was successful.</message>
11     <udfs>
12         <udf>
13             <key>dealID</key>
14             <value>RO-123</value>
15             <type>string</type>
16         </udf>
17         <udf>
18             <key>portfolio</key>
19             <value>Green Portfolio A</value>
20             <type>string</type>
21         </udf>
22         <udf>
23             <key>trader</key>
24             <value>kroepfl_p</value>
25             <type>string</type>
26         </udf>
27         <udf>
28             <key>risklevel</key>
29             <value>medium</value>
30             <type>string</type>
31         </udf>
32         <udf>
33             <key>exposure</key>
34             <value>2324230.12</value>
35             <type>long</type>
36         </udf>
```

```
37      </udfs>
38 </Event>
```

B.2.1.4 Backoffice Deal Validation

```
 1 <Event xmlns="http://www.mercatis.com/lighthouse3">
 2     <context>
 3         <deployedComponentCode>Deal Validator</deployedComponentCode
               >
 4         <deploymentLocation>BackOfficeSystem</deploymentLocation>
 5     </context>
 6     <code>BODealValidationOK</code>
 7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
           dateOfOccurrence>
 8     <level>INFO</level>
 9     <machineOfOrigin>BackOfficeSystem</machineOfOrigin>
10     <message>Backoffice deal validation for deal RO-123 was
           successful</message>
11     <udfs>
12         <udf>
13             <key>dealID</key>
14             <value>RO-123</value>
15             <type>string</type>
16         </udf>
17     </udfs>
18 </Event>
```

B.2.1.5 Counterparty Confirmation OK

```
 1 <Event xmlns="http://www.mercatis.com/lighthouse3">
 2     <context>
 3         <deployedComponentCode>Electronic Confirmation Service</
               deployedComponentCode>
 4         <deploymentLocation>BackOfficeSystem</deploymentLocation>
 5     </context>
 6     <code>CounterpartyConfirmationOK</code>
 7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
           dateOfOccurrence>
 8     <level>INFO</level>
 9     <machineOfOrigin>TradingSystem-node1</machineOfOrigin>
10     <message>Counterparty e.on trading ltd confirmed dealID RO-123</
           message>
11     <udfs>
12         <udf>
13             <key>dealID</key>
14             <value>RO-123</value>
15             <type>string</type>
16         </udf>
```

```
17          <udf>
18              <key>counterparty</key>
19              <value>e.on trading ltd</value>
20              <type>string</type>
21          </udf>
22          <udf>
23              <key>duns</key>
24              <value>1234567</value>
25              <type>int</type>
26          </udf>
27          <udf>
28              <key>commodity</key>
29              <value>CO2 Cert</value>
30              <type>string</type>
31          </udf>
32          <udf>
33              <key>price</key>
34              <value>100.134,20</value>
35              <type>double</type>
36          </udf>
37          <udf>
38              <key>volume</key>
39              <value>250</value>
40              <type>int</type>
41          </udf>
42          <udf>
43              <key>delivery</key>
44              <value>2010-03-01</value>
45              <type>date</type>
46          </udf>
47
48      </udfs>
49  </Event>
```

B.2.1.6 New Deal Error

```
1  <Event xmlns="http://www.mercatis.com/lighthouse3">
2      <context>
3          <deployedComponentCode>Market Interface</
               deployedComponentCode>
4          <deploymentLocation>Trading System</deploymentLocation>
5      </context>
6      <code>NewDealError</code>
7      <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
           dateOfOccurrence>
8      <level>INFO</level>
9      <machineOfOrigin>TradingSystem-node1</machineOfOrigin>
10     <message>Could not parse deal body from deal RO-111.</message>
```

```
11      <udfs>
12          <udf>
13              <key>dealID</key>
14              <value>RO-111</value>
15              <type>string</type>
16          </udf>
17          <udf>
18              <key>market</key>
19              <value>RORC</value>
20              <type>string</type>
21          </udf>
22      </udfs>
23 </Event>
```

B.2.1.7 Deal Validation Error

```
1 <Event xmlns="http://www.mercatis.com/lighthouse3">
2     <context>
3         <deployedComponentCode>Deal Validator</deployedComponentCode
              >
4         <deploymentLocation>BackOfficeSystem</deploymentLocation>
5     </context>
6     <code>UniqueKeyContraintViolation</code>
7     <dateOfOccurrence>${__time(yyyy-MM-dd'T'HH:mm:ss.S)}+02:00</
          dateOfOccurrence>
8     <level>INFO</level>
9     <machineOfOrigin>BackOfficeSystem</machineOfOrigin>
10    <message>ORA-0001: UniqueKeyContraintViolation: Deals.DealID.</
          message>
11 </Event>
```

B.2.2 CEP-Eventdefinitionen

B.2.2.1 Szenario 1

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent where code='
          PortfolioRiskAnalyserOK'</query>
5     <event>
6         <code>CEP:PortfolioRiskAnalyserOK</code>
7         <level>INFO</level>
8         <machineOfOrigin>ftibco99</machineOfOrigin>
9         <message>Protfolio Risk Analyser was successul</message>
10        <context>
11            <deployedComponentCode>CEP-Engine-1</
                  deployedComponentCode>
```

```
12              <deploymentLocation>CEP Engine</deploymentLocation>
13            </context>
14      </event>
15    </cepevent>
16 </cepevents>
17 <?xml version="1.0" encoding="UTF-8" ?>
18 <cepevents>
19   <cepevent>
20     <query>select * from XMLEvent where code='BODealValidationOK'</
           query>
21     <event>
22         <code>CEP:BODealValidationOK</code>
23         <level>INFO</level>
24         <machineOfOrigin>ftibco99</machineOfOrigin>
25         <message>Deal Validation in Backoffice was successful.</
               message>
26         <context>
27             <deployedComponentCode>CEP-Engine-1</
                   deployedComponentCode>
28             <deploymentLocation>CEP Engine</deploymentLocation>
29           </context>
30     </event>
31   </cepevent>
32 </cepevents>
33 <?xml version="1.0" encoding="UTF-8" ?>
34 <cepevents>
35   <cepevent>
36     <query>select portfolio.udfs.udf[1].value as portfolioID ,
           validator.* as p
37     from
38     XMLEvent(code="PortfolioRiskAnalyserOK") .win:time(60
39     sec) as portfolio ,
40     XMLEvent(code="BODealValidationOK").win:time(60 sec)
41     as validator
42     where
43     validator.udfs.udf[1].value = portfolio.udfs.udf[1].
44     valu
45     select * from XMLEvent  where code='CounterpartyConfirmationOK'
           </query>
46     <event>
47         <code>CEP:ValidationOK</code>
48         <level>INFO</level>
49         <machineOfOrigin>ftibco99</machineOfOrigin>
50         <message>Validation of Deal Successful.</message>
51         <context>
52             <deployedComponentCode>CEP-Engine-1</
                   deployedComponentCode>
```

```
53              <deploymentLocation>CEP Engine</deploymentLocation>
54            </context>
55      </event>
56    </cepevent>
57 </cepevents>
58
59 <?xml version="1.0" encoding="UTF-8" ?>
60 <cepevents>
61   <cepevent>
62     <query>select * from XMLEvent where code='
           PortfolioRiskAnalyserOK' and
63     udfs.udf[1].value regexp 'Green Portfolio*'</query>
64     <event>
65         <code>CEP:GreenportfolioRiskOK</code>
66         <level>INFO</level>
67         <machineOfOrigin>ftibco99</machineOfOrigin>
68         <message>Successfully checked risk level of green portfolio
               .</message>
69         <context>
70             <deployedComponentCode>CEP-Engine-1</
                   deployedComponentCode>
71             <deploymentLocation>CEP Engine</deploymentLocation>
72           </context>
73     </event>
74   </cepevent>
75 </cepevents>
```

B.2.2.2 Szenario 2

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>select * from XMLEvent 'here context.
          deployedComponentCode='Market Interface'
5     and level='ERROR' and code != 'CEP: MarketInterfaceError'</query
          >
6     <event>
7         <code>CEP:MarketInterfaceError</code>
8         <level>ERROR</level>
9         <machineOfOrigin>ftibco99</machineOfOrigin>
10        <message>Error receiving deal.</message>
11        <context>
12            <deployedComponentCode>CEP-Engine-1</
                  deployedComponentCode>
13            <deploymentLocation>CEP Engine</deploymentLocation>
14          </context>
15    </event>
16  </cepevent>
```

```
17 </cepevents>
```

B.2.2.3 Szenario 3

```
 1 <?xml version="1.0" encoding="UTF-8" ?>
 2 <cepevents>
 3   <cepevent>
 4     <query>select * from XMLEvent where code='
         PortfolioRiskAnalyserOK'</query>
 5     <event>
 6         <code>CEP:PortfolioRiskAnalyserOK</code>
 7         <level>INFO</level>
 8         <machineOfOrigin>ftibco99</machineOfOrigin>
 9         <message>Protfolio Risk Analyser was successul</message>
10         <context>
11             <deployedComponentCode>CEP-Engine-1</
                   deployedComponentCode>
12             <deploymentLocation>CEP Engine</deploymentLocation>
13           </context>
14     </event>
15   </cepevent>
16 </cepevents>
17 <?xml version="1.0" encoding="UTF-8" ?>
18 <cepevents>
19   <cepevent>
20     <query>select * from XMLEvent where code='BODealValidationOK'</
         query>
21     <event>
22         <code>CEP:BODealValidationOK</code>
23         <level>INFO</level>
24         <machineOfOrigin>ftibco99</machineOfOrigin>
25         <message>Deal Validation in Backoffice was successful.</
             message>
26         <context>
27             <deployedComponentCode>CEP-Engine-1</
                   deployedComponentCode>
28             <deploymentLocation>CEP Engine</deploymentLocation>
29           </context>
30     </event>
31   </cepevent>
32 </cepevents>
33 <?xml version="1.0" encoding="UTF-8" ?>
34 <cepevents>
35   <cepevent>
36     <query>select portfolio.udfs.udf[1].value as portfolioID ,
         validator.* as p
37     from
38     XMLEvent(code="PortfolioRiskAnalyserOK") .win:time(60
```

```
39      sec) as portfolio ,
40      XMLEvent(code="BODealValidationOK").win:time(60 sec)
41      as validator
42      where
43      validator.udfs.udf[1].value = portfolio.udfs.udf[1].
44      valu
45      select * from XMLEvent  where code='CounterpartyConfirmationOK'
            </query>
46      <event>
47          <code>CEP:ValidationOK</code>
48          <level>INFO</level>
49          <machineOfOrigin>ftibco99</machineOfOrigin>
50          <message>Validation of Deal Successful.</message>
51          <context>
52              <deployedComponentCode>CEP-Engine-1</
                    deployedComponentCode>
53              <deploymentLocation>CEP Engine</deploymentLocation>
54            </context>
55      </event>
56    </cepevent>
57 </cepevents>
58
59 <?xml version="1.0" encoding="UTF-8" ?>
60 <cepevents>
61    <cepevent>
62      <query>select * from XMLEvent where code='
            PortfolioRiskAnalyserOK' and
63      udfs.udf[1].value regexp 'Green Portfolio*'</query>
64      <event>
65          <code>CEP:GreenportfolioRiskOK</code>
66          <level>INFO</level>
67          <machineOfOrigin>ftibco99</machineOfOrigin>
68          <message>Successfully checked risk level of green portfolio
                .</message>
69          <context>
70              <deployedComponentCode>CEP-Engine-1</
                    deployedComponentCode>
71              <deploymentLocation>CEP Engine</deploymentLocation>
72            </context>
73      </event>
74    </cepevent>
75 </cepevents>
```

B.2.2.4 Szenario 4

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3    <cepevent>
```

```
4     <query>select * from pattern [every start=XMLEvent(code="
          NewDealOK") ->
5           (timer:interval(3 sec) and not done=XMLEvent(code="
                PortfolioRiskAnalyserOK", udfs.udf[0].value = start.
                udfs.udf[0].value))]</query>
6     <event>
7         <code>CEP:PortfolioRiskAnalysisLATE</code>
8         <level>WARNING</level>
9         <machineOfOrigin>ftibco99</machineOfOrigin>
10        <message>Error in deal validation.</message>
11        <context>
12            <deployedComponentCode>CEP-Engine-1</
                  deployedComponentCode>
13            <deploymentLocation>CEP Engine</deploymentLocation>
14          </context>
15    </event>
16  </cepevent>
17 </cepevents>
```

B.2.2.5 Szenario 5

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cepevents>
3   <cepevent>
4     <query>insert into MemoryStream
5     select deal.udfs.udf[1].value as dealId,
6     traderConfirmation.* as t
7     from
8       XMLEvent(code="NewDealOK").win:time(60 sec) as deal,
9       XMLEvent(code="TraderConfirmationOK").win:time(60 sec) as
            traderConfirmation
10    where
11      deal.udfs.udf[1].value = traderConfirmation.udfs.udf[1].value
12    </query>
13    <event>
14        <code>CEP:PortfolioRiskAnalyserOK</code>
15        <level>INFO</level>
16        <machineOfOrigin>ftibco99</machineOfOrigin>
17        <message>Protfolio Risk Analyser was successul</message>
18        <context>
19            <deployedComponentCode>CEP-Engine-1</
                  deployedComponentCode>
20            <deploymentLocation>CEP Engine</deploymentLocation>
21          </context>
22    </event>
23  </cepevent>
24 </cepevents>
25
```

```
26 <cepevents>
27   <cepevent>
28     <query>select evt.* from pattern [every mem=MemoryStream(t.code=
           "TraderConfirmationOK") ->
29          (timer:interval(3 sec)
30        and not
31          evt=XMLEvent(code="DealValidationOK", evt.udfs.udf[1].value=
              mem.t.udfs.udf[1].value))]
32     </query>
33     <event>
34         <code>CEP:DealValidationLATE</code>
35         <level>WARNING</level>
36         <machineOfOrigin>ftibco99</machineOfOrigin>
37         <message>Deal Validation has not finished within specified
              timeframe</message>
38         <context>
39             <deployedComponentCode>CEP-Engine-1</
                  deployedComponentCode>
40             <deploymentLocation>CEP Engine</deploymentLocation>
41           </context>
42     </event>
43   </cepevent>
44 </cepevents>
```

Literaturverzeichnis

[1] http://dictionary.cambridge.org/. [Online; accessed 03-July-2009].

[2] http://www.askoxford.com. [Online; accessed 03-July-2009].

[3] http://www.merriam-webster.com/dictionary/monitoring. [Online; accessed 03-July-2009].

[4] http://www.duden.de/. [Online; accessed 03-July-2009].

[5] http://www.oasis-open.org/. [Online; accessed 05-July-2009].

[6] http://complexevents.com/stanford/rapide/. [Online; accessed 08-July-2009].

[7] http://www.darpa.mil/. [Online; accessed 08-July-2009].

[8] http://complexevents.com/event-processing/. [Online; accessed 11-July-2009].

[9] http://httpd.apache.org/. [Online; accessed 13-July-2009].

[10] http://kapowtech.com/index.php/products/kapow-web-data-server/standard-edition. [Online; accessed 13-July-2009].

[11] http://www.apacs.org.uk/. [Online; accessed 14-July-2009].

[12] http://www.ggf.org. [Online; accessed 16-July-2009].

[13] http://eu-datagrid.web.cern.ch/. [Online; accessed 16-July-2009].

[14] http://globus.org/toolkit/. [Online; accessed 17-July-2009].

[15] http://activemq.apache.org/performance.html. [Online; accessed 12-August-2009].

[16] http://www.gridforum.org/. [Online; accessed 2009-08-07].

[17] http://www.tibco.com/software/messaging/enterprise-message-service/default.jsp. [Online; accessed 2009-08-07].

[18] http://java.sun.com/developer/technicalArticles/Ecommerce/jms/index.html. [Online; accessed 2009-08-07].

[19] http://www.fiorano.com/docs/jms_performance_comparison.pdf. [Online; accessed 7-August-2009].

[20] http://www.capitalware.biz/dl/docs/krissoft_jms_performance.pdf. [Online; accessed 7-August-2009].

[21] http://activemq.apache.org. [Online; accessed 22-October-2009].

[22] http://web.progress.com/sonic/sonic-esb.html. [Online; accessed 22-October-2009].

[23] http://www-01.ibm.com/software/integration/wmq/. [Online; accessed 22-October-2009].

[24] http://www.oracle.com/database/index.html. [Online; accessed 22-October-2009].

[25] http://www.microsoft.com/SQL/default.mspx. [Online; accessed 22-October-2009].

[26] http://www-01.ibm.com/software/data/db2/. [Online; accessed 22-October-2009].

[27] http://www.mysql.com/. [Online; accessed 22-October-2009].

[28] http://www.postgresql.org/. [Online; accessed 22-October-2009].

[29] http://web.progress.com/apama/. [Online; accessed 22-October-2009].

[30] http://www.aleri.com/. [Online; accessed 22-October-2009].

[31] http://www.uc4.com/en/products/uc4-reporting-management-tools/complex-event-processing/. [Online; accessed 22-October-2009].

[32] http://www.tibco.com/software/complex-event-processing/. [Online; accessed 22-October-2009].

[33] http://esper.codehaus.org. [Online; accessed 22-October-2009].

[34] http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/Level.html. [Online; accessed 7-August-2009].

[35] http://esper.codehaus.org/esper/performance/performance.html. [Online; accessed 12-August-2009].

[36] Reference model for service oriented architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/, 2006.

[37] *Wahrig Deutsches Wörterbuch*. Wissen Media Verlag, 2008.

[38] Ron Anderson-Lehman. Continental airlines flies high with real-time business intelligence. *Security Zone, Plattform für Informationssicherheit*, 2004.

[39] B Azvine, Z Cui, D D. Nauck, and B Majeed. Real time business intelligence for the adaptive enterprise. In *CEC-EEE '06: Proceedings of the The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, page 29, Washington, DC, USA, 2006. IEEE Computer Society.

[40] Alan Rodger Balachandar Ganesh. Ca unicenter® network and systems management r11.1 technology audit. http://www.butlergroup.com/research/KCInterPages/%7B40F56540-4484-4F02-AFA4-3223A591333F%7D.asp, Januar 2008.

[41] Hewlett-Packard Development Company. Hp operations center software datasheet. https://h10078.www1.hp.com/cda/hpdc/display/main/, 2008. [Online, accessed 2009-10-22].

[42] Hewlett-Packard Development Company. Hp operations manager software datasheet. https://h10078.www1.hp.com/cda/hpdc/, 2008. [Online, accessed 2009-10-22].

[43] Andy Cooke, Alasdair J G Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, and Antony Wilson. R-gma: An information integration system for grid monitoring. In *Proceedings of the 11th International Conference on Cooperative Information Systems*, pages 462–481, 2003.

[44] IBM Corporation. Ibm tivoli datasheet. http://www-01.ibm.com/software/tivoli/products/monitor/, 2009. [Online, accessed 2009-10-22].

[45] IBM Corporation. Ibm tivoli user manual. http://www-01.ibm.com/software/tivoli/products/monitor/, 2009. [Online, accessed 2009-10-22].

[46] Thomas H. Davenport. *Process innovation : reengineering work through information technology*. Harvard Business School Press, 1993.

[47] Thomas H. Davenport. The fad that forgot people. *The Fast Company*, 1(1), 1995.

[48] Roy Schulte David Luckham. Event processing glossary - version 1.1. http://complexevents.com/wp-content/uploads/2008/08/epts-glossary-v11.pdf, 2008. [Online; accessed 08-July-2009].

[49] CA Computer Associates GmbH. Unicenter® network and systems management r11 datenblatt, 2008.

[50] Michael Hammer and James Champy. *Reengineering the corporation : a manifesto for business revolution*. Harper Business, 1. ed edition, 1993.

[51] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology, March-April 2008, ETH Zurich*, 7(3):125–151, 2008.

[52] Philip Howard. Progress apama: An evaluation by bloor research. http://www.progress.com/apama/web/global/bloor_esp_review/index.ssp, October 2006. [Online, accessed 2009-10-22].

[53] TIBCO Software Inc. Business factor factsheet. http://www.tibco.com/software/business-activity-monitoring/businessfactor/, 2006. [Online; accessed 2009-10-22].

[54] Peter Schrey Jan-Peter Richter, Harald Haller. Serviceorientierte architektur. *Informatik Spektrum*, Oktober, 2005.

[55] Tobias Vogel Kristin Wende Christine Legner Jan W. Schwemm, Roger Heutschi. Serviceorientierte architekturen: Einordnung im business engineering. Technical report, Universität St. Gallen, Institut für Wirtschaftsinformatik, 2006.

[56] H.; Bayer F.; Karagiannis D Junginger, S.; Kühn. Workflow-based business monitoring. *Fischer, L. (Ed.): Workflow Handbook 2004*, pages 65–80, October 2004.

[57] D. Karagiannis, S. Junginger, and R. Strobl. Introduction to business process management systems concepts. *Business Process Modelling, Lecture Notes in Computer Science*, pages 81–106, 1996.

[58] Dimitris Karagiannis. Bpms: Business process management systems. *SIGOIS Bulletin, Special Issue: business process reengineering*, 16(1):10–13, 1995.

[59] P. G. W. Keen and M. S. Scott Morton. *Decision Support Systems: An Organizational Perspective*. Addison-Wesley, Reading, 1978.

[60] Srilata Zaheer Adrian S Choo Kevin Linderman, Roger G. Schroeder. Six sigma: a goal-theoretic perspective. *Journal of Operations Management*, 21:193–203, 2003.

[61] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA : Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, November 2004.

[62] Rolf Kralitsch. Sox: eine herausforderung für it-abteilungen und it-dienstleister. *Security Zone Newsletter*, March 2005.

[63] P. Kung, C. Hagen, M. Rodel, and S. Seifert. Business process monitoring & measurement in a large bank: challenges and selected approaches. In *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, pages 955–961, 2005.

[64] Harald Kühn and Dimitris Karagiannis. Modellierung und simulation von geschäftsprozessen. *WISU - Das Wirtschaftsstudium*, 30:1161–1170, 2001.

[65] Louis Lovas. Apama blogs: A transparent new year. http://apama.typepad.com/my_weblog/2009/01/a-transparent-new-year.html, 2009.

[66] Tom Lubinski. Business activity monitoring: Process control for the enterprise. http://complexevents.com/wp-content/uploads/2008/06/sl_bam_021208.pdf, 2008.

[67] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[68] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[69] Sun Microsystems. http://java.sun.com/developer/technicalArticles/WebServices/soa/. [Online; accessed 2009-07-24].

[70] Harvey B. Newman, Iosif Legrand, Philippe Galvez, Ramiro Voicu, and Catalin Cirstoiu. Monalisa : A distributed monitoring service architecture. *CoRR*, cs.DC/0306096, 2003.

[71] Nick Russell, Wil M. P. van der Aalst, and Arthur. Exception handling patterns in process-aware information systems. Technical report, BPMcenter.org, 2006.

[72] Hermann J. Schmelzer and Wolfgang Sesselmann. *Geschäftsprozessmanagement in der Praxis. Kunden zufrieden stellen, Produktivität steigern, Wert erhöhen.* Hanser Fachbuch, 2007.

[73] Jörg Purucker Hinnerk Brügmann Stefan Reinheimer, Florian Lang. 10 antworten zu soa. *HMD Praxis der Wirtschaftsinformatik*, 253:7–8, 2007.

[74] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47, December 2005.

[75] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. Global grid formum: A grid monitoring architecture. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-1.pdf, 2002. [Online; accessed 2009-10-22].

[76] W. M. P. van der Aalst, Ter, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.