Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3129

Architektur und Implementierung ereignis- und situationsgetriebener Workflows

Sascha Julien Retter

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Matthias Wieland
begonnen am:	10. Januar 2011
beendet am:	12. Juli 2011
CR-Klassifikation:	H.3.5, H.4.1, J.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Vor der eigentlichen, inhaltlichen Auseinandersetzung mit dem Thema dieser Diplomarbeit werden einführende Überlegungen zur Motivation vorgestellt, und es werden einige Konventionen und Rahmenbedingungen für diese Diplomarbeit festgelegt.

1.1. Motivation

Die Workflow-Technologie hat ihren Ursprung im Document Routing und Case Processing Mitte der 1980er Jahre. Seitdem gab es zahlreiche Entwicklungen in diesem Bereich, und Workflow Management Systeme (WfMSe) finden immer häufiger ihren Einsatz in Unternehmen. Seit kurzem rückt daneben jedoch immer stärker ein anderes Paradigma in den Fokus: Complex Event Processing (CEP).

Der Begriff CEP wurde zum ersten Mal von David Luckham in seinem Buch „The Power of Events“ [Luc02], erschienen im Jahr 2002, verwendet. Sensoren und Softwaresysteme liefern in immer größerem Umfang feingranulare Informationen, die möglichst schnell weiterverarbeitet werden müssen. Diese Beschleunigung wird laut der Vorhersagen von Gartner für das Jahr 2011 weiter zunehmen. Unter dem Punkt „Addressing Key Advancements in Application Architecture“ konstatiert Gartner „die Beschleunigung, die vor einigen Jahren mit serviceorientierten Architekturen begann, findet ihre Fortsetzung durch den immer häufiger werdenden Einsatz von ereignisgesteuerten Systemen“[1].

Es liegt also nahe, die in den Unternehmen eingesetzten Workflow-Systeme um die Fähigkeiten des Complex Event Processing bzw. um Fähigkeiten von Systemen zur Situationserkennung zu erweitern. Im Rahmen dieser Diplomarbeit soll die Architektur eines solchen Systems entworfen und eine Implementierung entwickelt werden.

Es sind somit z.B. Szenarien denkbar, in denen Ereignisse von Produktionssystemen (z.B. Smartfactories) und aus Enterprise Resource Planning (ERP)-Systemen von speziellen Systemen zur Ereignisverarbeitung verarbeitet werden, und dann nur die akkumulierten Ergebnisse von Workflow-Systemen weiterverarbeitet werden.

[1] http://www.gartner.com/DisplayDocument?doc_cd=208777

1.2. Konventionen

Verwendung englischer Begriffe

Ich verwende in meiner Diplomarbeit immer dann englische Begriffe, wenn es sich um einen feststehenden Fachbegriff handelt oder wenn es keine mir bekannte, äquivalente und eindeutige deutsche Entsprechung gibt. Für die Bezeichnungen, die sich auf die Implementierung beziehen, werden ebenso englische Begriffe verwendet.

Abkürzungen

Alle Abkürzungen werden bei der ersten Verwendung ausgeschrieben. Die Abkürzung steht bei der ersten Verwendung in Klammern. Im Folgenden wird in der Regel nur noch die Abkürzung verwendet. Unter A.1 im *Anhang* ist ein Verzeichnis aller Abkürzungen zu finden.

Zitierung

Sofern Inhalte aus dem Web verwendet werden, wird die URL der Quelle als Fußnote angegeben. Alle anderen Quellen werden wie folgt gekennzeichnet: [WMKL09]. Im Anhang unter A.2 befindet sich ein Literaturverzeichnis, dem detaillierte Informationen zu all diesen Quellen entnommen werden können.

Hervorgehobener Text

Bezeichnungen im Text, die sich auf Bezeichnungen der Implementierung, auf Tabellen, Aufzählungen oder Abbildungen beziehen, werden *hervorgehoben*. Bei Verweisen auf Kapitel im Text wird der Titel eines Kapitels ebenfalls *hervorgehoben*.

Darstellung von Ereignissen

Um Ereignisse auf einer abstrakten, konzeptionellen Ebene darzustellen, verwende ich eine Darstellung wie in Abbildung 1.1. Es wird immer die Instanz eines Ereignis-Typs, im Folgenden nur noch Ereignis genannt, dargestellt. Der Typ der Instanz wird durch die Farbe und den Buchstaben repräsentiert. Die Attribute 1 - n eines Ereignisses des Typs A werden als A1 - A$n, n \in \mathbb{N}$ dargestellt. Die Bezeichnung von Attributen durch den Namen des Typs und eine natürliche Zahl ist erforderlich, um die Attribute verschiedener Ereignis-Typen auseinanderhalten zu können. Wäre der Bezeichner eines Attributs nur eine natürliche Zahl, so würde eine Verwechslungsgefahr zwischen Attributen unterschiedlicher Typen bestehen. Mit der vorgestellten Notation lassen sich verschiedene Instanzen eines Ereignis-Typs nicht

unterscheiden. Dies stellt aber auf der Abstraktionsebene, auf der diese Notation verwendet wird, kein Problem dar.

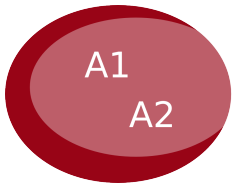


Abbildung 1.1.: Darstellung eines Ereignisses mit zwei Attributen

Darstellung von Geschäftsprozessen

Für alle Darstellungen von Geschäftsprozessen verwende ich die Business Process Modelling Notation oder seit Version 2.0 Business Process Model and Notation (BPMN). Es wird davon ausgegangen, dass der Leser BPMN 2.0 versteht. In Kapitel 2.2 wird lediglich eine kurze Einführung gegeben, die die für die folgende Arbeit relevanten Aspekte hervorhebt.

1.3. Rechtliche Hinweise

Alle Lizenzen verwendeter Software sind zu beachten. Folgende Bibliotheken oder Programme wurden bei der Implementierung verwendet.

Bibliothek	**Lizenz**
Esper	General Public License v2
EclipseLink	Eclipse Public License
H2	Eclipse Public License
Open Message Queue	Common Development and Distribution License
Activiti	Apache License

Tabelle 1.1.: Lizenzen der verwendeten Programmbibliotheken

1.4. Aufbau des Dokuments

Der Aufbau dieses Dokuments orientiert sich grob am Ablauf eines Softwareprojekts. Zunächst werden in Kapitel 2 einige für das Verständnis wichtige *Grundlagen* behandelt. Danach wird in Kapitel 3 versucht, einen *Überblick* über die verwandten Arbeiten zu geben, die zusammen mit den *Grundlagen* und der *Motivation* die Basis der weiteren Arbeit bilden.

Danach werden in Kapitel 4 *Anforderungen* an das System definiert. Im Anschluss werden einige zentrale *Technologien* in Kapitel 5 vorgestellt. In Kapitel 6 wird auf Basis der *Anforderungen* eine *Architektur* für das System entworfen. In Kapitel 7 werden das Vorgehen bei der *Implementierung* und die Erweiterungsmöglichkeiten beschrieben. Danach wird in Kapitel 8 - *Evaluation* - das Ergebnis der Implementierung untersucht. Ferner wird diskutiert, wie sich die Implementierung nutzen lässt, ob alle Anforderungen aus Kapitel 4 realisiert wurden, wie das System optimal eingesetzt werden kann und welche Verbesserungsmöglichkeiten es gibt. In Kapitel 9 - *Zusammenfassung und Ausblick* - werden die Ergebnisse der Arbeit zusammengefasst und offene Fragen benannt.

2. Grundlagen

Dieses Kapitel enthält einige Grundlagen, die für das Verständnis der nachfolgenden Kapitel benötigt werden. Viele Grundlagen können hier aber nur angerissen werden. Insofern erhebt die Darstellung der Grundlagen in dieser Diplomarbeit in keiner Weise einen Anspruch auf Vollständigkeit.

2.1. Workflow-Management

Wie bereits in der Einleitung erwähnt, spielen WfMSe in vielen Unternehmen eine wichtige Rolle. WfMSe werden in vielen Bereichen eingesetzt, um Geschäftsprozesse auf einem Rechner auszuführen.

Im Folgenden werden zentrale Begriffe aus dem Bereich Workflow-Management definiert, die in dieser Diplomarbeit verwendet werden. Die verwendeten Definitionen stammen aus [LR00] und aus dem Glossar der Workflow Management Coalition (WfMC)[1].

In [LR00] wird ein Geschäftsprozess als eine Folge von Aktivitäten beschrieben, die von verschiedenen Personen ausgeführt werden. Ein solcher Geschäftsprozess wird typischerweise auf gleiche Weise immer von Neuem wiederholt. Ein Prozessmodell wird verwendet, um einen Geschäftsprozess zu beschreiben. Ein solches Prozessmodell definiert alle möglichen Pfade eines Geschäftsprozesses und alle Regeln, die festlegen, welche Pfade genommen und welche Aktionen ausgeführt werden müssen. Ein Prozessmodell dient als Vorlage für die Instantiierung aller konkreten Prozesse. Solche Prozesse müssen nicht auf einem Rechner ausgeführt werden. Geschäftsprozesse können sowohl aus Teilen bestehen, die auf einem Rechner ausgeführt werden, als auch aus Teilen, die nicht von einem Rechner unterstützt ausgeführt werden. Die auf dem Rechner ausführbaren Teile werden als Workflow-Modell bezeichnet [LR00]. Analog zu Prozessmodell und Prozess wird die Instanz eines Workflow-Modells Workflow genannt. Ein WfMS ist ein System, mit dem Workflows durch die Anwendung von Software definiert, erstellt und verwaltet werden können. Dabei werden die Workflows auf einer oder mehreren Workflow-Engines ausgeführt, die in der Lage sind, Prozessdefinitionen zu interpretieren, mit Workflow-Teilnehmern zu interagieren, und ggf. IT-Werkzeuge und Anwendungen aufzurufen [WfM99].

[1] http://www.wfmc.org/Glossaries-FAQs/View-category.html

Modellierung

Zur Definition von Geschäftsprozessen mit Softwareunterstützung stehen verschiedene Notationen bzw. Sprachen zur Verfügung. Folgende Sprachen werden in [LK06] als weit verbreitet oder für die Zukunft relevant bezeichnet:

- UML 2.0 - Activity Diagram (AD)
- Business Process Definition Metamodel (BPDM)
- Business Process Modelling Notation (BPMN)
- Event Driven Process Chains (EPC)
- Integrated DEFinition Method 3 (IDEF3)
- Petri-Netze
- Role Activity Diagram (RAD)

Aus meiner Sicht ist heutzutage für die Modellierung vor allem die BPMN von Bedeutung. BPMN ist ein offener Standard, wird von zahlreichen Herstellern, darunter z.B. IBM und SAP, unterstützt[2], und mit der Version 2.0 besitzt BPMN eine definierte Ausführungssemantik.

Mit der Version 2.0 findet eine Umdeutung des Akronyms von Business Process Modelling Notation hin zu Business Process Model and Notation statt [OMG10]. Diese Umdeutung spiegelt die Tatsache wider, dass BPMN nicht mehr nur eine Modellierungssprache ist, sondern auch eine definierte Ausführungssemantik und zudem ein XML-Serialisierungsformat besitzt, das beschreibt, wie die technischen Details des Prozesses gespeichert werden [FRH10, S. 199].

Für BPMN 2.0 existieren bereits einige wenige Ausführungsumgebungen. Zum Zeitpunkt des Beginns der Diplomarbeit waren mir JBOSS JBPM[3] und Activiti[4] bekannt, die in Kapitel 5 betrachtet werden.

2.2. Business Process Model and Notation

Es folgen nun einige Erläuterungen zur Ausführung von BPMN 2.0 und zur Modellierung des technischen Prozessmodells (siehe Abbildung 2.1). Sämtliche Ausführungen basieren auf dem Buch „Praxishandbuch BPMN“. Die Ausführung von BPMN 2.0 beruht auf dem Token-Konzept. Dies bedeutet, dass der Ablauf eines Workflows sich immer gerade an der Stelle befindet, an der sich ein Token befindet. Bei parallelen Abläufen existieren mehrere Token, d.h. zu Beginn eines parallelen Ablaufs wird ein einzelnes Token entsprechend der

[2] http://www.omg.org/bpmn/BPMN_Supporters.htm
[3] http://www.jboss.org/jbpm
[4] http://www.activiti.org

Anzahl paralleler Zweige geklont. Am Ende eines parallelen Ablaufs müssen die Zweige entsprechend synchronisiert und überzählige Token konsumiert werden.

Die Autoren von [FRH10] legen großen Wert auf die Unterscheidung verschiedener Ebenen bei der Prozessmodellierung. Sie unterscheiden dabei folgende Ebenen:

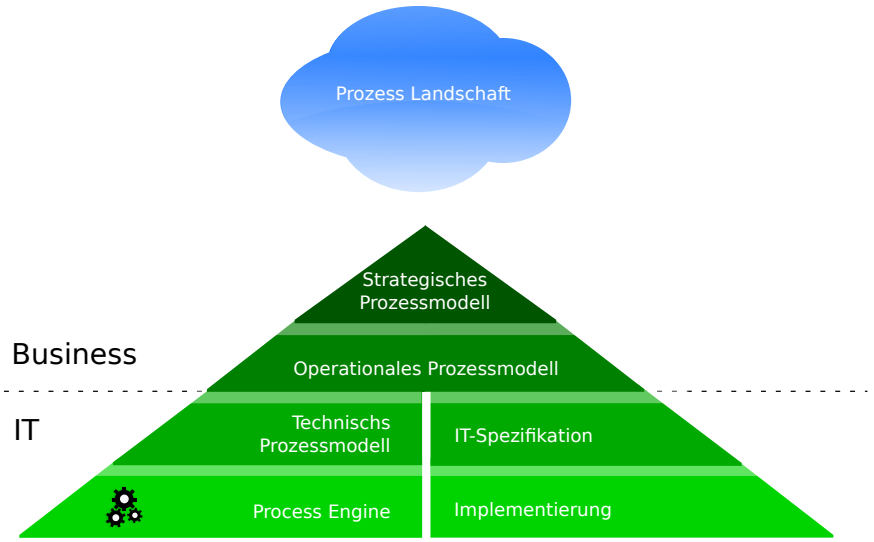


Abbildung 2.1.: BPMN Modellierungsebenen (Quelle: vgl. [FRH10])

Die Autoren beschreiben weiterhin ein Vorgehen, bei dem erst zwischen den Ebenen zwei und drei entschieden wird, welche Technologie und Prozess Engine eingesetzt wird. Wird eine BPMN 2.0 Engine eingesetzt, erfolgt eine Verfeinerung des Modells der Ebene zwei um die technisch notwendigen Details [FRH10, S. 188 f.]. Die Alternativen, eine andere Engine einzusetzen oder das Prozess Modell mit Hilfe einer klassischen Programmiersprache umzusetzen, verfolge ich an dieser Stelle nicht weiter, da bei der Umsetzung der Diplomarbeit eine BPMN 2.0 Engine zum Einsatz kommen soll.

Ereignisse

Da Ereignisse in dieser Diplomarbeit eine zentrale Rolle spielen, werden im Folgenden Ereignisse in BPMN 2.0 näher betrachtet.

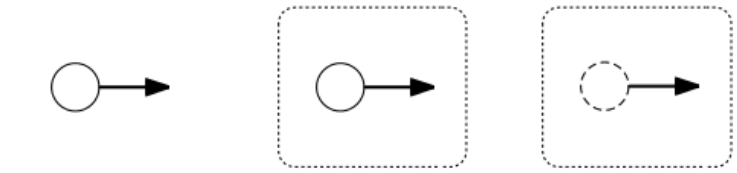
Ereignisse können laut [FRH10, S. 48] dazu führen, dass:

- der Prozess gestartet wird,
- der Prozess oder ein Prozesspfad fortgesetzt wird,

- die aktuell in Bearbeitung befindliche Aufgabe oder der Teilprozess abgebrochen wird,
- während der Bearbeitung einer Aufgabe oder eines Teilprozesses ein weiterer Prozesspfad durchlaufen wird.

Im Folgenden werden alle eingetretenen Ereignisse (im Original „catching events“) abgebildet, die BPMN zur Verfügung stellt. Im Vergleich zur BPMN 1.2 kamen in der Notation bei den Ereignissen die nicht-unterbrechenden Ereignisse hinzu.

In Abbildung 2.2 werden Start-Ereignisse dargestellt. In Teilbild (a) ist ein Ereignis dargestellt, durch das ein Prozess gestartet wird. In Teilbild (b) ist ein Start-Ereignis abgebildet, das einen Ereignis-Teilprozess startet und den Oberprozess abbricht. Das Teilbild (c) zeigt ein Ereignis, das einen Ereignis-Teilprozess startet, ohne den Oberprozess zu beenden.



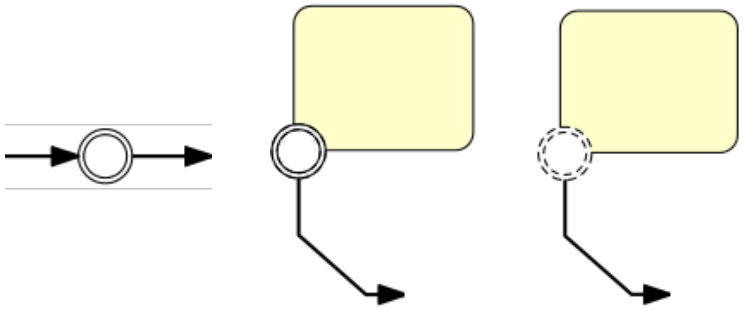
(a) Start-Ereignis

(b) Start-Ereignis Teilprozess - unterbrechend

(c) Start-Ereignis Teilprozess - nicht unterbrechend

Abbildung 2.2.: BPMN Start-Ereignis

In Abbildung 2.3 sind Zwischen-Ereignisse abgebildet. In Teilbild (a) ist ein Ereignis dargestellt, das den Prozess unterbricht, bis das Ereignis eintritt. Das Teilbild (b) zeigt eine Aktivität, die unterbrochen wird, wenn das Ereignis eintritt. In Teilbild (c) ist eine Aktivität abgebildet, die bei Eintritt des Ereignisses nicht abgebrochen wird.



(a) Kontrollfluss unterbrechendes Ereignis

(b) Aktivität unterbrechendes Ereignis

(c) Aktivität nicht unterbrechendes Ereignis

Abbildung 2.3.: BPMN Start-Ereignisse

2.3. Complex Event Processing

Der Begriff CEP wurde erstmals in dem im Jahr 2002 erschienen Buch „The Power of Events“ [Luco2] von David Luckham verwendet. Im Alltagsgebrauch ist ein Ereignis etwas, was passiert [Luco2, S. 88].

Ein Ereignis hat nach [Luco2, S.88] drei Aspekte:

- Form
- Bedeutung
- Beziehung

Mit *Form* ist die Repräsentation eines Ereignisses gemeint. David Luckham schreibt: „Die Form eines Ereignisses ist ein Objekt“. Wobei mit Objekt eine beliebige Objekt-Repräsentation z.B. in Form eines Strings, eines Tupels oder eines Objekts im Sinne einer objektorientierten Programmiersprache gemeint ist. Dieses Objekt kann verschiedene Eigenschaften enthalten (z.B. wo das Ereignis aufgetreten ist).

Zu einem Ereignis gehört ebenso seine *Bedeutung*. Ein Ereignis steht immer für eine Aktivität. Modelltheoretisch ausgedrückt ist ein Ereignis ein deskriptives Modell einer Aktivität.

Der dritte Aspekt, der zu einem Ereignis gehört, ist die *Beziehung*. Eine Aktivität steht bezüglich des zeitlichen und kausalen Zusammenhangs und der Aggregation in einer Beziehung zu anderen Aktivitäten. Der gleiche Zusammenhang gilt, da ein Ereignis ein Modell einer Aktivität ist, also ebenso für Ereignisse.

Ein komplexes Ereignis ist eine Abstraktion anderer Ereignisse. Bei diesen anderen Ereignissen kann es sich entweder um andere komplexe Ereignisse oder aber um einfache Ereignisse handeln. Einfach sind Ereignisse, die sich nicht weiter in Teilereignisse unterteilen lassen. Da es in vielen Fällen für die weitere Verarbeitung keine Rolle spielt, ob es sich bei einem Ereignis um ein komplexes Ereignis oder ein einfaches Ereignis handelt, verwende ich in diesem Fall den Begriff Ereignis sowohl für einfache Ereignisse als auch für komplexe Ereignisse[Luco2].

Der Gedanke hinter CEP ist die Strukturierung von Ereignissen in Abstraktionsebenen. Ein Beispiel ist in 2.4 dargestellt.

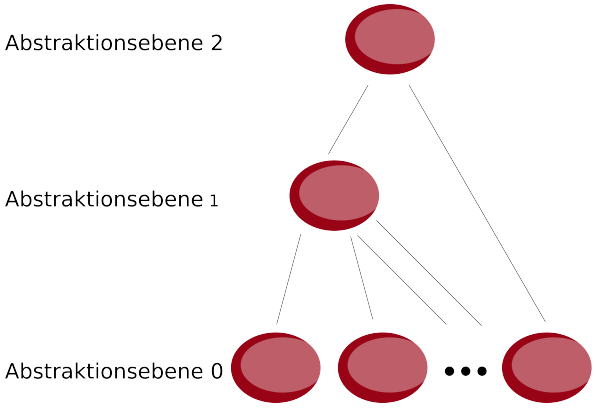


Abbildung 2.4.: Abstraktionsebenen bei der Verarbeitung von Ereignissen

Ein komplexes Ereignis ist also entweder eine Abstraktion mehrerer weniger-abstrakter Ereignisse der gleichen Abstraktionsebene oder eine Abstraktion von Ereignissen unterschiedlicher Abstraktionsebenen. So ist das Ereignis der *Abstraktionsebene 1* in der Abbildung 2.4 eine Abstraktion mehrerer Ereignisse der *Abstraktionsebene 0*, und das Ereignis der *Abstraktionsebene 2* ist eine Abstraktion von Ereignissen der Ebenen 0 und 1.

Das Ziel von CEP ist es, möglichst in Echtzeit eine große Anzahl von Ereignissen zu verarbeiten. Abbildung 2.5 zeigt schematisch die Verarbeitung von Ereignissen mit Hilfe einer CEP-Engine.

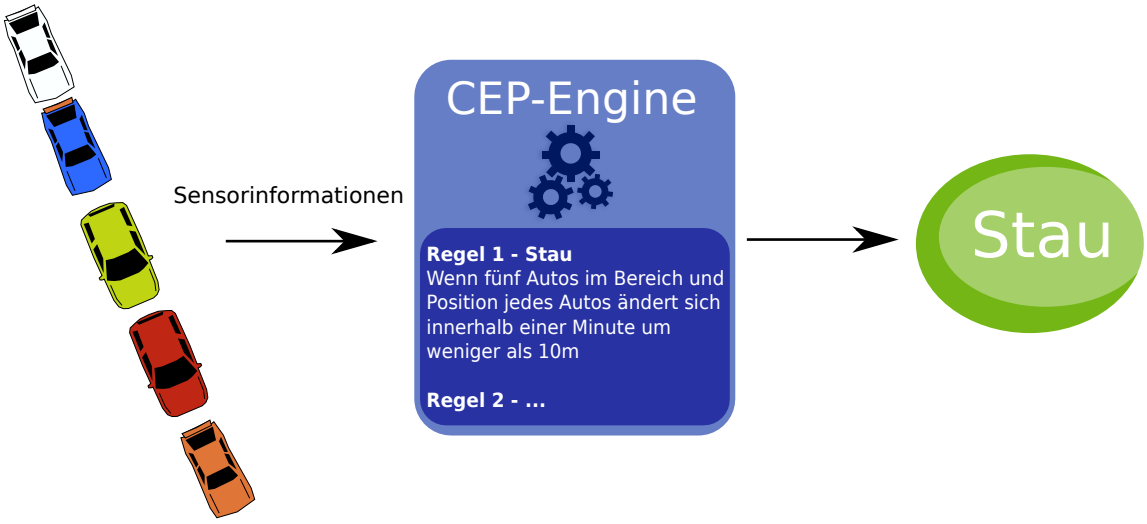


Abbildung 2.5.: Verarbeitung von Ereignissen

Die Verarbeitung der Ereignisse wird durch sogenannte Event Pattern Languages (EPLs) gesteuert. EPLs dienen der Beschreibung von Ereignismustern. Wird ein solches Muster von einer CEP-Engine erkannt, so wird ein Ereignis erzeugt. Momentan gibt es noch keinen Standard für EPLs, und jeder Hersteller verwendet somit eine eigene Sprache. Grundsätzlich lassen sich nach [BE07] vier Dimensionen unterscheiden, die eine Sprache zum Erkennen von Ereignismustern bzw. der Abfrage von Ereignissen unterstützen muss:

- data extraction - Datenextraktion
- event composition - Ereigniskomposition
- temporal (and causal) relationships - zeitliche (und kausale) Verknüpfungen
- event accumulation - Ereignisakkumulation

Die *Datenextraktion* dient der Filterung von Daten aus bestehenden Ereignissen. Die Abbildung 2.6 zeigt, dass einzelne Eigenschaften eines Ereignisses ausgewählt werden, die dann die Eigenschaften eines neuen komplexen Ereignisses bilden.

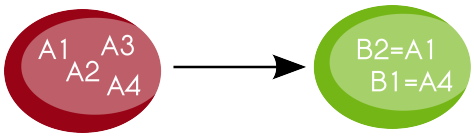


Abbildung 2.6.: Datenextraktion aus Ereignissen

Bei der *Ereigniskomposition* werden dagegen, wie in Abbildung 2.7 dargestellt, die Eigenschaften eines Ereignisses mit denen anderer Ereignisse zusammengesetzt.

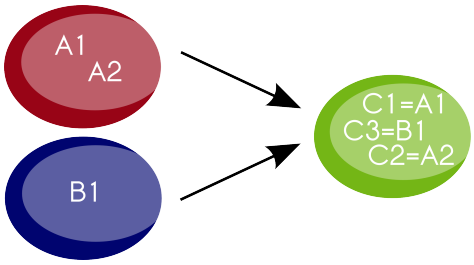


Abbildung 2.7.: Ereigniskomposition

Bei der *Ereignisakkumulation*, dargestellt in Abbildung 2.8, werden die Attribute verschiedener Ereignisse mit Hilfe eines Operators akkumuliert. Da ein Ereignisstrom prinzipiell unendlich ist, müssen für die Berechnung Zeiträume oder eine gewünschte Anzahl von Ereignissen angegeben werden. Eine solche Akkumulation kann z.B. der Berechnung des durchschnittlichen Gewinns der letzten Stunde oder der Berechnung des durchschnittlichen Gewinns der letzten zehn Buchungs-Ereignisse dienen.

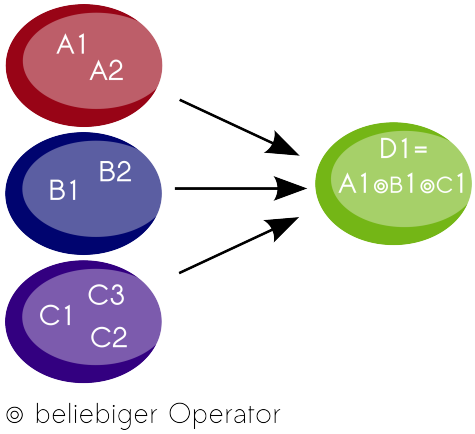


Abbildung 2.8.: Ereignisakkumulation

Schließlich gibt es noch zeitliche oder kausale Abhängigkeiten von Ereignissen. Es muss also möglich sein, auszudrücken, dass Ereignisse zeitlich oder kausal eine vorgegebene Ordnung erfüllen. In Abbildung 2.9 wird ein neues Ereignis des Typs C erzeugt, wenn ein Ereignis des Typs A und ein Ereignis des Typs B in dieser zeitlichen Abfolge auftreten.

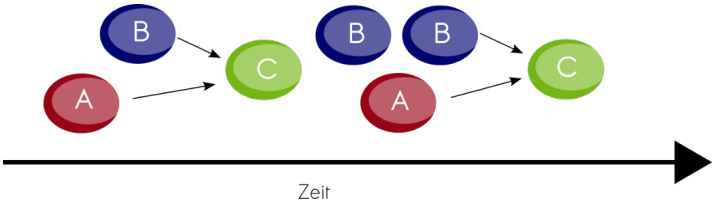


Abbildung 2.9.: Zeitliche Abhängigkeit von Ereignissen

Es gibt, wie in Tabelle 2.1 dargestellt, eine Reihe von Systemen und Engines für CEP. Die Tabelle erhebt in keiner Weise einen Anspruch auf Vollständigkeit.

Hersteller	Engine	EPL	Lizenz
Oracle	Oracle CEP	Continuous Query Language (CQL)	proprietär
Codehaus	(N)Esper	Esper EPL	GPL
Tibco	Tibco Business Events	Rete-basierte Regelsprache	proprietär
Sybase	Sybase CEP	Continuous Computation Language (CCL)	proprietär

Tabelle 2.1.: CEP-Engines

Die Fachstudie „Vergleich von Complex Event Processing-Ansätzen für Business Activity Monitoring“ [BDK10], durchgeführt am Institut für Architektur von Anwendungssystemen (IAAS) der Universität Stuttgart, hat einige CEP-Werkzeuge untersucht und ist dabei zu dem Ergebnis gekommen, dass vor allem im Hinblick auf Leistungsfähigkeit und Ausgereiftheit der EPL mit Esper[5] eine sehr leistungsfähige Ausführungsumgebung zur Verfügung steht [BDK10]. Esper ist ein Opensource-Projekt und eignet sich auch daher sehr gut für eine Diplomarbeit. Aus diesem Grund werde ich bei der Realisierung des erarbeiteten Konzepts auf Esper zurückgreifen.

2.4. Sonstige Definitionen und Begriffe

Die folgenden Begriffe und Abkürzungen werde ich in meiner Diplomarbeit immer wieder verwenden, ohne näher darauf einzugehen, was sich dahinter verbirgt. Sollten Zweifel an der Bedeutung eines Begriffs bestehen, den ich in meiner Diplomarbeit verwende, so sind die Glossare der WfMC[6], was Workflows betrifft, und das Glossar der Event Processing Technical Society (EPTS)[7], was CEP betrifft, geeignete Quellen, an denen ich mich nach Möglichkeit orientiert habe. An dieser Stelle werde ich mich auf eine ganz kurze Definition einiger verwendeter Begriffe beschränken, die ich bis hierhin noch nicht erklärt habe.

Event Driven Architecture (EDA) = ereignisgesteuerte Architektur

Event Driven Architecture (EDA) ist ein Architekturstil. In einer EDA sind zentrale Komponenten durch Ereignisse gesteuert bzw. kommunizieren über Ereignisse [LS08].

Service Oriented Architecture (SOA) = serviceorientierte Architektur

Service Oriented Architecture (SOA) ist ein Architekturstil, dessen zentrale Eigenschaft die lose Kopplung von Services ist. Services stellen eine Funktionalität an einer Netzwerkadresse über verschiedene Transportprotokolle und Formate und mit unterschiedlichen Quality of Service (QoS)-Eigenschaften zur Verfügung. Ein Service steht wie Elektrizität, Wasser, Gas etc. immer zur Verfügung [CLS+05], muss also weder erstellt noch zerstört werden.

[5] http://esper.codehaus.org
[6] http://www.wfmc.org
[7] http://www.ep-ts.com

Context = Kontext

„Kontext ist jede Information, die geeignet ist, die Situation einer Entität zu beschreiben. Eine Entität ist eine Person, ein Ort oder ein Objekt, die relevant für die Interaktion zwischen einem Benutzer und einer Anwendung, einschließlich des Benutzers und der Anwendung selbst, ist [Dey01].“

Context-aware = kontextbewusst

„Ein System ist Context-aware, wenn es Kontext verwendet, um relevante Informationen und/oder Services bereitzustellen, deren Relevanz für Benutzer abhängig von der Aufgabe des Benutzers ist [Dey01].“

Situation

Ich verwende den Begriff Situation in meiner Diplomarbeit, wie er in [Dey01] verwendet wird. Eine Situation besteht aus allen Umständen, die relevant für eine Entität sind. Aus meiner Sicht ist der Kontext ein deskriptives Modell einer Situation.

Business Rule = Geschäftsregeln

Den Begriff Geschäftsregel verwende ich in dieser Arbeit gemäß der Verwendung in [MIK08]. „Im allgemeinen ist eine Geschäftsregel eine Aussage mit dem Ziel, das Verhalten und die Informationen einer Organisation zu leiten oder zu beeinflussen [SN03].“ In [MIK08] werden strukturelle Regeln wie Integritätsregeln (z.B. jede Buchung muss einen Posten enthalten) oder Ableitungsregeln (z.B. ein Kunde, mit einem monatlichen Umsatz von 2000 Euro, ist ein Premiumkunde) und operationale Regeln wie Transformationsregeln (z.B. das Alter eines Kunden kann nur einmal pro Jahr geändert werden) oder Reaktionsregeln (z.B. bei einem Kreditantrag über 100.000 Euro muss dieser von einem Mitarbeiter der Ebene drei genehmigt werden) genannt.

Event Processing Rule = Ereignisverarbeitungsregeln

Event Processing Rules können auf viele verschiedene Arten (endlicher Automat, Java-Code, SQL-Code, Event-Condition-Action Rules) beschrieben werden [LS08]. Mit Regeln wird deklarativ beschrieben, wie die Verarbeitung von Ereignissen durch eine entsprechende Engine abläuft.

3. Verwandte Arbeiten

Nachdem die Grundlagen im vorangegangenen Kapitel präsentiert wurden, werden nun einige Arbeiten, die für die Diplomarbeit maßgeblich sind, vorgestellt. Grundsätzlich lassen sich die relevanten, wissenschaftlichen Arbeiten in vier Gruppen aufteilen. Es gibt Arbeiten, die die Modellierung von Geschäftsprozessen und Geschäftsregeln bzw. Ereignissen behandeln. Die nächste Gruppe beschäftigt sich generell mit der Verarbeitung von Ereignissen im Zusammenhang mit Workflows, und die dritte relevante Gruppe befasst sich mit kontext- und situationsbezogenen Workflows. Die vierte Gruppe beschäftigt sich mit der Ableitung von Ereignissen bzw. dem logischen Schließen auf Ereignisse. Natürlich existieren auch zwischen den einzelnen Arbeiten Überlappungen, so dass sich diese nicht immer ganz eindeutig einer Gruppe zuordnen lassen. In solchen Fällen habe ich versucht, die Arbeit ihres Themenschwerpunkts entsprechend einzuordnen.

3.1. Modellierung von Geschäftsprozessen, Geschäftsregeln und Ereignissen

Eine interessante Arbeit zur Modellierung von Ereignissen ist [DGB07]. Die Autoren schlagen darin eine Sprache zur Modellierung von Ereignissen vor, die sie Business Event Modelling Language (BEMN) nennen. Der Name deutet schon die Anlehnung an BPMN an. Tatsächlich sprechen die Autoren sogar von einer Erweiterung der BPMN [DGB07].

Die in Abbildung 3.1 dargestellten grafischen Notationselemente werden vorgestellt, es wird eine formale Semantik für sie definiert und ein Metamodell präsentiert:

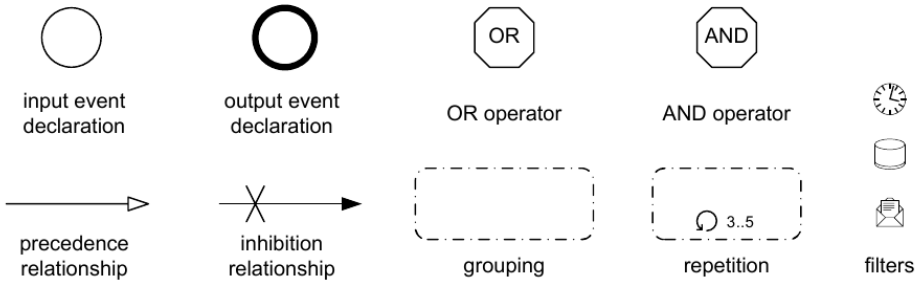


Abbildung 3.1.: Notationselemente von BEMN (Quelle: [DGB07])

Aber nicht nur die Darstellung sondern auch die Semantik, auf die ich an dieser Stelle nicht weiter eingehen will, ist stark an die von BPMN angelehnt. Die Kompatibilität zwischen BEMN und BPMN erfordert dies letztendlich auch.

Am Ende werfen die Autoren die Frage auf, welche Aspekte nun mit welchem Paradigma modelliert werden sollten. Diese Frage wird zumindest teilweise auch in weiter unten betrachteten Arbeit [DKGZ10] beantwortet.

Auch [MIK08] widmet sich der Frage, welche Geschäfts-Aspekte eher mit der klassischen Geschäftsprozess-Modellierung abgebildet werden und für welche Aspekte eher ein regelbasierter Ansatz gewählt werden sollte. Es werden fünf Kriterien identifiziert, mit denen sich bestimmen lassen soll, ob ein Aspekt eher mit einer Regel oder eher mit einem Geschäftsprozess modelliert werden sollte. Abbildung 3.2 zeigt eine Übersicht der Kriterien.

	Geschäftsregeln			Geschäftsprozesse	
Frequenz Änderungen	stündlich	täglich	wöchentlich	monatlich	jährlich
Implementierungs-verantwortung	Business User	Business Analyst	Business / System Analyst	System Analyst	Programmierer
Grad der Auswirkung	sehr gering	gering	mittel	hoch	sehr hoch
Ursprung der Änderung	intern	Teilbereich	Fachbereich	Geschhäfts-partner	Externe
betroffener Bereich	konzernweit	mehrere Prozesse	Prozess	Aktivität	innerhalb d. Aktivität

Abbildung 3.2.: Modellierung: Entscheidungs-Framework (Quelle: [MIK08])

In ihrem Fazit stellen die Autoren fest, dass sie in ihrer Arbeit die gemeinsame visuelle Modellierung von Geschäftsprozessen und Geschäftsregeln nicht berücksichtigt haben. Ein Ansatz hierzu bietet die bereits oben erwähnte Erweiterung von BPMN, die in der Arbeit [DGB07] vorgeschlagen wird.

3.2. Workflows und Ereignisverarbeitung

Eine für diese Diplomarbeit besonders relevante Arbeit ist [WMKL09]. Ein zentraler Punkt dieser Arbeit ist der Gedanke der Vereinheitlichung der Ansätze von EDA und SOA, die in der Industrie jeweils für sich bereits weit verbreitet sind [WMKL09]. Die als Service Oriented Event Driven Architecture (SOEDA) bezeichnete Architektur basiert auf der Integration beider Architekturen durch Event-driven Process Chains (EPCs). Auf Modellierungsebene werden also EPCs verwendet. Das Modell wird zur Ausführung mit Hilfe eines modellgetriebenen Ansatzes transformiert. Als Zielsprachen werden BPEL für die Workflows und Esper-Regeln für das CEP vorgeschlagen. Durch SOEDA sollen also die Vorteile beider Architekturen vereint werden. Als Vorteile von EDA-basierten Systemen werden die Fähigkeiten aufgezählt, flexibel auf ad-hoc Änderungen zu reagieren, Situationen zu erkennen und eine große Anzahl von Ereignissen oder Datenströmen verarbeiten zu können. Für SOAs werden als Vorteile die Einhaltung von Standards, die Interoperabilität und die Integration von legacy Systemen beschrieben.

In [AESW08] wird ein Referenzmodell für Event Driven Business Process Management (EDBPM) vorgestellt, wobei die Autoren unter EDBPM eine Verknüpfung von Business Process Management (BPM) und CEP verstehen. Allerdings beschränken sich die Autoren in ihrer Betrachtung von EDBPM fast ausschließlich auf Business Activity Monitoring (BAM). Am Ende kommen die Autoren aber zu dem Schluss, dass in Zukunft Geschäftsprozesse auch automatisch auf Ereignisse reagieren können sollten [AESW08].

Auch die Arbeit [DKGZ10] setzt sich mit der Integration verschiedener Paradigmen auseinander. Dazu gehören Workflow-Management (WfM), Business Rules Management (BRM) und CEP. Einen Schwerpunkt der Arbeit bilden die Integration der Paradigmen auf Metamodellebene und die Frage, welche Aspekte wie und mit welchem der drei Paradigmen modelliert werden sollten.

In [BDG07] wird ebenfalls die Notwendigkeit zur Modellierung von Ereignissen im Umfeld von Geschäftsprozessen dargestellt. Allerdings liegt der Schwerpunkt dann auf der Beschreibung von Mustern für Composite Events und ihrer Realisierbarkeit mit Business Process Execution Language (BPEL) und BPMN. Einen Überblick über die Composite Event Patterns gibt Tabelle 3.1. Die Autoren sind der Auffassung, dass eine nahtlose Modellierung von Prozessen und Ereignissen benötigt wird, um eine konsistente Sicht für Prozessexperten zu erreichen. Sowohl BPEL als auch BPMN besitzen nicht die Fähigkeit, verschiedene Abstraktionsebenen (siehe Kapitel 2.3) auszudrücken. BPMN bietet abgesehen davon aber mehr Ereignistypen. Trotzdem zeigt sich, dass sich weder mit BPEL noch mit BPMN alle vorgeschlagenen Muster realisieren lassen [BDG07].

Composite Event Pattern	Beschreibung
1. Event Conjunction	Zwei oder mehr Ereignisse müssen in der richtigen Reihenfolge eintreten.
2. Event Cardinality	Eine festgelegte Anzahl von Ereignissen desselben Typs treten ein.
3. Event Disjunction	Alternative Ereignisse treten in einer bestimmten Reihenfolge auf.
4. Inhibiting Event	Ein Ereignis tritt ein, während ein anderes Ereignis nicht eintritt.
5. Event Time Relation	Zwei Ereignisse treffen während oder außerhalb eine bestimmten Zeitfensters ein.
6. Subscription Time Relation	Ein Ereignis tritt in einem Zeitfenster ein, das relativ zum Zeitpunkt der Subscription ist.
7. Consumption Time Relation	Ein Ereignis tritt mindestens eine bestimmte Zeit vor dem Konsum ein.
8. Absolute Time Relation	Ein Ereignis tritt vor oder nach einem festgelegten Zeitpunkt ein.
9. Event Data Dependency	Die Daten zweier eintretender Ereignisse stehen in einer bestimmten Relation zueinander.
10. Process Instance Data Dependency	Ein eintretendes Ereignis steht in einer bestimmten Relation zu Daten der zugehörigen Prozess-Instanz.
11. Environment Data Dependency	Ein eingetretenes Ereignis steht in einer bestimmten Relation zu Daten, die für alle Prozess-Instanzen verfügbar sind.
12. Consume Once	Ein Ereignis kann höchstens von einer Prozess-Instanz konsumiert werden.
13. Consume Multiple Times	Ein Ereignis kann mehrer Male (ggf. auch von der gleichen Prozess-Instanz) konsumiert werden.

Tabelle 3.1.: Composite Event-Patterns (Quelle: vgl. [BDG07])

3.3. Kontext- und situationsbezogene Workflows

Zu kontextbezogenen Workflows existiert eine Vielzahl von Arbeiten. Ich erwähne hier nur einige, die für diese Diplomarbeit von Bedeutung sind.

In dem Konferenzbeitrag [WKNL07] wird das Ziel verfolgt, technische Prozesse mit Workflow Systemen, die ursprünglich für Geschäftsprozesse entwickelt wurden, zu modellieren und auszuführen. Als grundlegender Unterschied zwischen traditionellen Geschäftsprozessen und technischen Prozessen werden dabei die Ereignisse der Realwelt aufgeführt, die von besonderer Bedeutung für technische Prozesse sind. In diesem Zusammenhang wird vom sogenannten Business-Production-Gap gesprochen. Diese Lücke soll mit context-aware Workflows geschlossen werden. Die Autoren stellen fest, dass eine schnelle Reaktion auf Kontextänderungen und die Anpassung an eine veränderte Umgebung wichtige Anforderungen darstellen, die von etablierten Workflow Sprachen nicht unterstützt wird. In der zitierten Arbeit wird eine Erweiterung (Context4BPEL) von BPEL vorgeschlagen, um context-aware Workflows zu realisieren. Für die Realisierung wird in der Arbeit über die BPEL-Erweiterung die Nexus Context Management Plattform[1] angebunden. Über eine zusätzliche Komponente, den Context Event Scheduler, können Workflows per Context-Event instantiiert werden, und einzelne Workflow Instanzen können sich für bestimmte Context-Events registrieren, die dann zur Steuerung des Kontrollflusses verwendet werden können. Neben dem Konsum von Context-Events (asynchron), die von der Nexus Platform erzeugt werden, kann ein Workflow auch synchron Context-Queries an die Nexus Plattform stellen [WKNL07].

„Modeling Dynamic Context Awareness for Situated Workflows“ [WHR09] greift zwei Ansätze aus den Arbeiten [WKNL07], [HCC05], [HCKC06] und [SCCY07] auf und kritisiert sie im Hinblick auf die Notwendigkeit einer umfassenden Modellierung und die fehlende Flexibilität bei der Modellierung bezogen auf die zur Laufzeit relevanten Kontextinformationen. In der Arbeit ist die Rede von Adaptable Pervasive Flow (APF), einer weitreichenden Erweiterung des klassischen Workflow-Paradigmas. Die Autoren betrachten dabei drei Aspekte. Beim ersten Aspekt geht es um die Erweiterung klassischer Workflows um Situationsabhängigkeit. Hinter dem zweiten Aspekt verbirgt sich ein Konzept, das die Autoren dynamische Kontextprovisionierung nennen. Es ermöglicht die dynamische Bestimmung relevanter Kontextinformationen zur Laufzeit. Der dritte und letzte Aspekt betrifft einen Mechanismus zur Constraint- und Ereignisbehandlung, der es möglich macht, auf Änderungen des relevanten Kontextes geeignet zu reagieren. Das Grundprinzip funktioniert wie im Folgenden beschrieben. Für die Repräsentation der realen Welt wird ein Entitäten-Modell vorgeschlagen. Teil dieses Konzepts sind sogenannte *Entity Events*, die generiert werden, wenn zu einem bestimmten Zeitpunkt etwas in der realen Welt passiert, das die Änderung einer Entität verursacht. Des Weiteren wird das Konzept von Scopes eingeführt, um Aktivitäten eines Workflows zu gruppieren. Während der Modellierung werden die Scopes mit Attachments versehen, die einen Entitäts-Typ beschreiben. Zur Laufzeit wird der Workflow dann mit einer passenden Entitäten-Instanz assoziiert und hat somit Zugriff auf die Kontextinformationen dieser Entität. Über sogenannte *Context-Frames* wird der Zugriff auf Entitäten ermöglicht, die bei der Modellierung noch nicht bekannt sind. Durch ihren Ansatz, relevante Kontextinformationen über Attachments bzw. Context-Frames verfügbar zu machen, wollen die Autoren die Menge der Kontextinformationen auf den relevanten Kontext beschränken. Über die Entity-Events und entsprechende Event-Handler soll es möglich sein, auf Änderungen in der Realwelt zu reagieren [WHR09].

[1] `http://www.nexus.uni-stuttgart.de`

3.4. Ableitung von Ereignissen

Ein in der IT immer wieder auftretendes Problem ist die Behandlung von Sachverhalten der Realität, die mit Unschärfe bzw. Unsicherheit behaftet sind. Es gibt bei der Modellierung von Ereignissen und dem Schließen auf Ereignisse verschiedene Ansätze. Schließen bedeutet in diesem Zusammenhang den Vorgang der Ableitung bzw. Abstraktion von Ereignissen. Wie bereits in Kapitel 2.3 beschrieben, kann es sich hierbei um die Abstraktion von Ereignissen unterschiedlicher oder gleicher Abstraktionsstufen handeln.

Der am weitesten verbreitete Ansatz zur Modellierung ist die Modellierung von Ereignissen und das Schließen durch Regeln, die wie ein Geschäftsprozess im Voraus modelliert werden.

Dieser Ansatz wird z.B. von Max Pucher, Chief Architect bei ISIS Software[2], in seinem Blog[3] kritisiert. Mit einer Modellierung von Ereignissen im Voraus würde man in die Falle einer zu sehr vereinfachten Ursache-Wirkungs-Kette tappen. Ein zentraler Punkt seiner Kritik bezieht sich auf die Unschärfe von Ereignissen, der man seiner Ansicht nach mit einer Modellierung im Voraus nicht gerecht wird.

Die Unschärfe und Unsicherheit von Ereignissen werden in einigen wissenschaftlichen Artikeln betrachtet, und es werden Lösungsansätze entwickelt.

In „Complex Event Processing over Uncertain Data“ [WGET08] werden Bayesche' Netze in Kombination mit einem Monte Carlo Sampling Algorithmus verwendet, um Ereignisse zu verarbeiten, die mit einer Unsicherheit behaftet sind.

Beide Ansätze haben gemeinsam, dass auch aus Ereignissen, die mit einer Unsicherheit behaftet sind, Ereignisse abgeleitet bzw. aus ihnen auf Ereignisse geschlossen werden kann.

In „Event Modelling and Reasoning with uncertain Information for Distributed Sensor Networks“ [MLM10] wird für die Event Modellierung und logische Schlussfolgerung aus den Ereignissen ein Rahmenwerk entworfen, das auf der Dempster-Shafer-Theorie beruht. Das Rahmenwerk erlaubt die Modellierung von mit einer Unsicherheit behafteten Ereignissen und das Schließen auf Ereignisse. Um eine Wissensbasis einzubeziehen, werden sogenannte Domain-Events eingeführt, die das Wissen repräsentieren. So kann das Wissen mit in die Regeln zur Schlussfolgerung einbezogen werden.

Im Gegensatz zu Pucher sehe ich kein generelles Problem darin, Regeln und Ereignisse im Voraus zu modellieren. Meiner Meinung nach hängt es stark von der Anwendung ab, ob eine Modellierung im Voraus möglich ist oder nicht. In Fällen, in denen eine Modellierung von Regeln im Voraus nicht möglich ist, kann ein Ansatz wie in [WGET08] oder [MLM10] verwendet werden. Allerdings stehen meines Wissens noch keine in der Praxis verwendbaren Systeme zur Verfügung. Beide Konzepte lassen sich nach meiner Einschätzung nicht ohne Weiteres mit bestehenden CEP-Engines realisieren. Es sollte aber kein Problem darstellen,

[2] http://www.isis-papyrus.com
[3] http://isismjpucher.wordpress.com/2010/11/15/can-bpmn-and-rules-handle-complex-events-no

ein System, das das Schließen auf der Basis von Ereignissen erlaubt, die mit einer Unsicherheit behaftet sind, in das System zu integrieren, welches im Rahmen dieser Diplomarbeit entwickelt wird.

4. Anforderungen

Die Spezifikation ist eines der wichtigsten Dokumente, nach [LL06] sogar das wichtigste Dokument, das während eines Softwareprojekts entsteht. Eine Spezifikation enthält die wesentlichen funktionalen und nicht-funktionalen Anforderungen aus Kunden- bzw. Benutzersicht und ist somit die Referenz für die gesamte, weitere Entwicklung von Software. Nun unterscheidet sich eine Diplomarbeit schon alleine aufgrund der Tatsache, dass bei einer Diplomarbeit im Allgemeinen nicht die Software im Sinne einer ausführbaren Software sondern das Konzept im Vordergrund steht, deutlich von einem Softwareprojekt in der Industrie. Trotzdem werde ich in diesem Kapitel einige Anforderungen an die Architektur und an die Umsetzung dieser Architektur sammeln. Nur so ist aus meiner Sicht die zielgerichtete Entwicklung einer guten Softwarearchitektur und später deren Umsetzung möglich.

Wie in einer Spezifikation üblich, werde ich versuchen, bei der Sammlung von Anforderungen eine klare Trennung von funktionalen und nicht-funktionalen Anforderungen vorzunehmen. Da bei einer Diplomarbeit im Allgemeinen kein Kunde im eigentlichen Sinne zur Verfügung steht, werde ich die Anforderungen auf Basis der Ausschreibung der Diplomarbeit, der Konzepte und Ideen aus den in Kapitel 3 vorgestellten Arbeiten erheben. Da kein Kunde zur Verfügung steht, werden einige Anforderungen absichtlich offen gelassen und in Kapitel 6 diskutiert. Die in diesem Kapitel genannten Anforderungen sind Anforderungen aus einer imaginären Kunden- bzw. Benutzersicht und enthalten absichtlich wenig technische Details. Die technischen Details werden in den folgenden Kapiteln 5 und 6 betrachtet.

Häufig enthält eine Spezifikation ein Kapitel „Einsatzbereich und Ziele“. Dieses Kapitel habe ich mit der Motivation in Kapitel 1.1 mehr oder weniger vorweggenommen.

4.1. Vorüberlegungen

Es soll eine Architektur entworfen werden, die bestehende Systeme integriert. Dabei handelt es sich um eine Workflow-Engine und um Systeme zur Verarbeitung von Ereignissen bzw. Erkennung von Situationen auf der Basis von Ereignissen. So soll es möglich sein, auch große Mengen von Ereignissen in einem speziell für diese Zwecke entwickelten System zu verarbeiten und nur geringe Mengen relevanter Ereignisse in einer Workflow-Engine weiter zu verarbeiten. Es soll möglich sein, beliebige Workflow-Engines und Systeme zur Ereignisverarbeitung bzw. Situationserkennung zu verwenden. Damit wird die Idee, verschiedene Paradigmen zu kombinieren, von [WMKL09], [AESW08] und [DKGZ10] aufgegriffen. In diesem Fall wird also, wie in Abbildung 4.1 dargestellt, die Idee des Workflow-Managements mit der des CEP kombiniert. Gegenstand dieser Arbeit ist vor allem die Interaktion zwischen

WfMS und einem System zur Ereignis- bzw. Situationsverarbeitung. Es wird also eine Architektur geschaffen, die die Integration eines WfMS mit einem oder mehreren Systemen zur Verarbeitung von Ereignissen bzw. Situationen ermöglicht. Der Aspekt der Modellierung wird vernachlässigt.

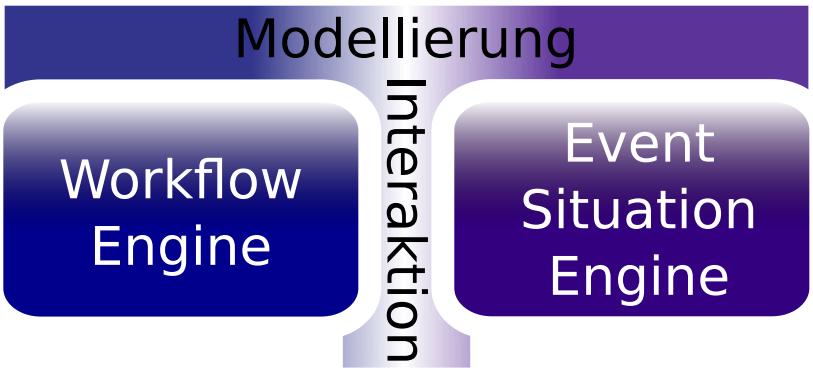


Abbildung 4.1.: Was soll die Architektur bzw. Implementierung leisten?

4.2. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen werden zu einem Großteil durch den Charakter von Ereignissen und durch WfMSe und Systeme zur Ereignisverarbeitung bzw. Situationserkennung (z.B. CEP-Engines) vorgegeben. Einige andere nicht-funktionale Anforderungen wie eine gute Erweiterbarkeit sollten heutzutage Standard sein.

Einsatz bestehender Systeme und Sprachen

Durch die Integration bestehender Systeme soll der Implementierungsaufwand gering gehalten werden. Außerdem lassen sich die Stärken bestehender Workflow-Systeme und die Stärken von Systemen zur Ereignisverarbeitung bzw. Situationserkennung nutzen, ohne diese implementieren zu müssen. Bereits bestehende Erfahrungen mit solchen Systemen und verbreiteten Sprachen können weiter genutzt werden. Anders als in [WHR09] können Erfahrungen im Bereich von CEP genutzt werden, und es muss kein neues Prinzip verstanden werden. Es müssen lediglich Erfahrungen gesammelt werden, wie sich die beiden Prinzipien in einem kombinierten Ansatz effizient nutzen lassen. Dies sollte zu einer hohen Akzeptanz bei Benutzern führen.

Erweiterbarkeit

Das entstehende System soll erweiterbar sein. Dies bedeutet, es soll auf einfache Weise möglich sein, neue Systeme zu integrieren. Dies gilt für die Integration neuer bzw. weiterer Systeme zur Ereignisverarbeitung bzw. Situationserkennung genauso wie für neue Ereignisquellen, die einfach in das Gesamtsystem integrierbar sein sollen. Ebenso soll ein Austausch der Workflow-Engine mit geringem Entwicklungsaufwand möglich sein.

Performanz

Der Durchsatz und die Latenz sollen durch die Integration nicht negativ beeinflusst werden. Insbesondere sollen der Durchsatz und die Latenz für die Verarbeitung von Ereignissen im Wesentlichen vom verwendeten System, also z.B. der verwendeten CEP-Engine, abhängen, da sonst ein wichtiger Vorteil der Workflow-Engine-externen Verarbeitung zunichte gemacht wird. Dieser Vorteil besteht z.B. in der Optimierung von CEP-Engines im Hinblick auf die Verarbeitung einer großen Menge von Ereignissen (hoher Durchsatz) bei möglichst geringer Verzögerung (geringe Latenz). Die Verarbeitungsgeschwindigkeit der Ergebnis-Ereignisse soll im Wesentlichen davon abhängen wie schnell die Workflow-Engine Ereignisse verarbeiten kann. Der Durchsatz und die Latenz sollen durch die Integration möglichst nicht negativ beeinflusst werden. Bestimmend für Durchsatz und Latenz sollen im Wesentlichen die Workflow-Engine und die Systeme zur Verarbeitung von Ereignissen bzw. Situationen sein.

Transparenz

Die Nutzung von CEP-Systemen oder vergleichbaren Systemen aus Workflows heraus soll möglichst einfach und transparent erfolgen. Dies gilt insbesondere für die Modellierung auf der Business-Ebene. Dem wichtigen Aspekt von WfM und SOAs, das Business-IT-Gap zu reduzieren bzw. ganz zu überwinden, soll also nicht entgegengewirkt werden. Wesentlich für die Nutzung eines Systems ist die Akzeptanz durch die Benutzer. Durch eine möglichst transparente Integration von Systemen zur Ereignis- und Situationsverarbeitung und die einheitliche Modellierung auf Grundlage einer bekannten Notation bzw. Sprache soll die Akzeptanz durch die Benutzer gefördert werden.

4.3. Funktionale Anforderungen

Im Wesentlichen wird die Funktionalität von einem WfMS und von einem oder mehreren Systemen zur Ereignisverarbeitung bzw. Situationserkennung erbracht. Aus diesem Grund werden an dieser Stelle nur wenige funktionale Anforderungen spezifiziert. Neue Funktionalität entsteht im Wesentlichen durch Emergenz. Dies bedeutet, dass durch die Integration

eines WfMSs und eines oder mehrerer anderer Systeme zur Ereignisverarbeitung bzw. Situationserkennung Funktionalität entsteht, die über die Funktionalität der Einzelsysteme hinausgeht.

Die meisten funktionalen Anforderungen sollten im Idealfall für den Benutzer möglichst transparent erfüllt werden.

Den funktionalen Anforderungen liegt der folgende grobe Ablauf zu Grunde:

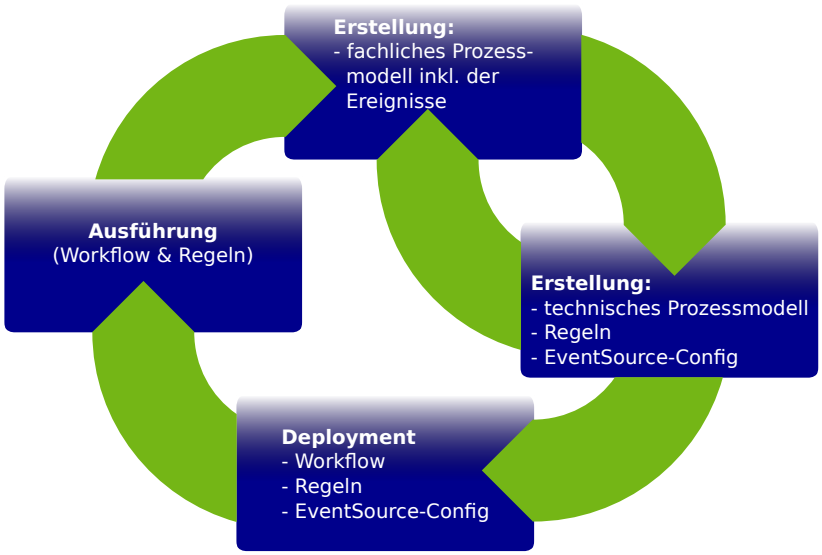


Abbildung 4.2.: Lebenszyklus eines Modells (Workflow-Modell, Regeln, Ereignisquellen-Konfigurationen)

Zunächst entstehen in einem iterativen Vorgehen ein Prozessmodell und die zugehörigen Regeln, die die vom Prozess benötigten Ereignisse erzeugen. Im nächsten Schritt werden Prozessmodell, Regeln und die Konfiguration von Ereignisquellen bereitgestellt (Deployment). In der Phase der Ausführung werden Instanzen des Prozessmodells von der Workflow-Engine ausgeführt. Das System zur Ereignisverarbeitung bzw. Situationserkennung leitet auf Basis der Regeln Ereignisse bzw. Situationen aus einfachen Ereignissen ab und stellt diese der Prozessinstanz zur Verfügung. Schließlich sollte eine Überprüfung des Modells erfolgen und ggf. von vorne begonnen werden.

Bei der Modellierung sollen nach Möglichkeit verschiedene Abstraktionsebenen (z.B. Businessebene, technische Ebene - vgl. 2.1) zur Verfügung stehen.

Es muss die Möglichkeit geben, Regeln zu verwalten und beliebige Ereignisquellen zu nutzen.

Es sollen möglichst viele der in [BDG07] vorgeschlagenen Composite Event Patterns realisierbar sein. Durch die Integration beliebiger Ereignisquellen soll eine Integrationsplattform

auf Ereignisebene entstehen. Die in [BDG07] beschriebenen Composite Event Patterns betrachten nicht die Reaktion auf eingetretene Ereignisse. Reaktionen sollen grundsätzlich im Workflow-Modell realisiert werden. Zusätzlich soll es aber möglich sein, Workflow-Instanzen per Ereignis zu erzeugen.

5. Technologien

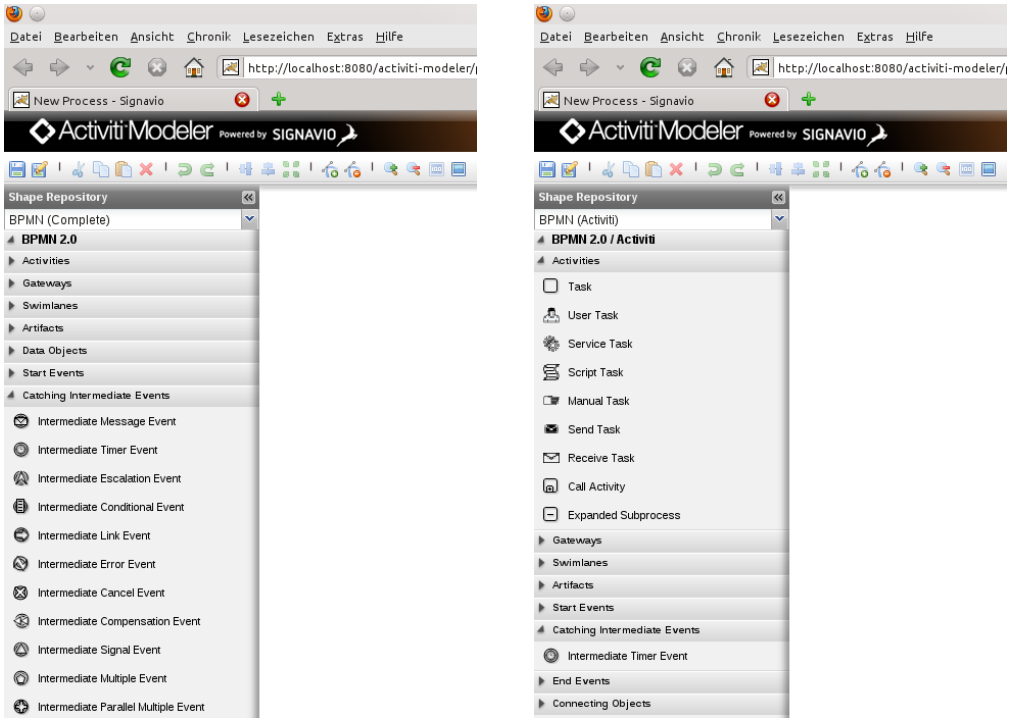
5.1. BPMN-Engines

Aufgrund der weitreichenden Ereignis-Konzepte, die von BPMN 2.0 unterstützt werden, habe ich mich für die Nutzung einer BPMN-Engine entschieden. Als BPMN-Engines kommen, wie bereits erwähnt, JBPM 5 des JBOSS-Projekts[1] und Activiti ein OpenSource Projekt, das vom Unternehmen Alfresco[2] initiiert wurde und von einigen anderen Firmen unterstützt wird, in Frage. Beide Engines sind in Java implementiert und basieren auf einem ähnlichen Konzept. Sie basieren auf einer sogenannten Process Virtual Machine (PVM). Die PVM ist ein Konzept zur Ausführung von Graphen, die aus Knoten und Kanten bestehen. Auf Basis der PVM können praktisch beliebige graphbasierte Sprachen implementiert werden. Im konkreten Fall wurde in beiden Fällen BPMN 2.0 implementiert. Es lassen sich aber auch BPEL, XPDL und andere Sprachen mit dem Konzept einer PVM realisieren[3].

Activiti

Bei der Evaluation von Activiti im Hinblick auf die Ereignisfähigkeit zeigt sich, dass viele BPMN 2.0 Elemente noch nicht implementiert sind. Insbesondere bei den Ereignissen sind bisher lediglich die angehefteten Ereignisse Timer und Fehler implementiert. Sowohl das Activiti-kompatible Subset von BPMN 2.0 in der Modellierungsumgebung von Activiti (siehe Abbildung 5.1) als auch der Parser der Engine unterstützen keine anderen Ereignisse. Insbesondere werden keine Nachrichten-Ereignisse unterstützt.

[1] http://www.jboss.org
[2] http://www.alfresco.com
[3] http://docs.jboss.com/jbpm/pvm/article



(a) Activiti - Vollständige BPMN 2.0 Palette

(b) Activiti - Subset von BPMN 2.0, das von Activiti unterstützt wird

Abbildung 5.1.: Activiti Modeler - BPMN-Palette

JBPM5

JBPM5 scheint bei der Implementierung von BPMN 2.0 Elementen schon weiter fortgeschritten. So unterstützt JBPM5 Nachrichten-Ereignisse als Start-Ereignisse und Zwischen-Ereignisse[4].

Auswahl einer Engine

Insgesamt scheint mir die Activiti-Engine besser strukturiert und durchdachter. Die Dokumentation ist bei beiden Projekten eher mangelhaft. Während das Benutzerhandbuch bei JBPM5 etwas weiter fortgeschritten ist, ist der Quellcode der Activiti-Engine besser kommentiert. Da möglicherweise Änderungen am Quellcode notwendig sind und das Activiti-Projekt einen wesentlich aktiveren Eindruck macht, habe ich mich für den Einsatz von Activiti bei

[4] `https://hudson.jboss.org/hudson/job/jBPM5/lastSuccessfulBuild/artifact/target/jbpm-5.1-SNAPSHOT-docs-build/jbpm-docs/html/ch01.html#d0e74`

meiner Implementierung entschieden. Aus diesem Grund folgen hier nun noch einige Details zum Aufbau (siehe Abbildung 5.2) von Actviti.

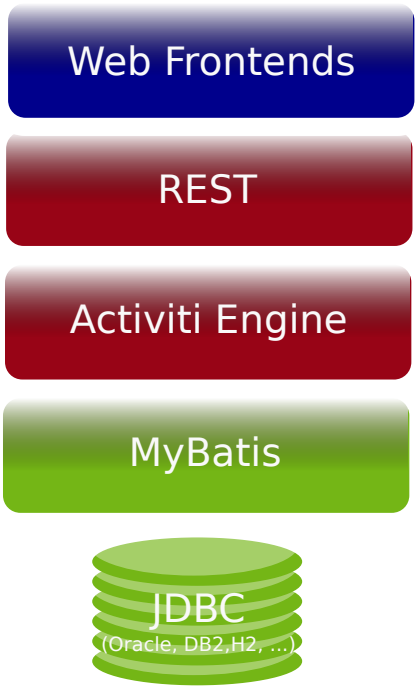


Abbildung 5.2.: Aufbau von Activiti (Quelle: vgl. [Men11])

Es stehen für Activiti verschiedene Benutzungsschnittstellen in Form von *WebFrontends* zur Verfügung. Diese kommunizieren mit der *Activiti Engine* per *REST*. Als Abstraktionsschicht für persistente Daten dient *MyBatis*. *MyBatis* unterstützt eine große Zahl gängiger Datenbankmanagement-Systeme (DBMSe) als Datenbank-Backend.

Die zentralen Benutzungsschnittstellen von Activiti sind:

Activiti-Modeler dient der Modellierung von Geschäftsprozessen.

Activiti-Probe ermöglicht das Deployment von Prozessen und die Anzeige von Informationen z.B. zu den laufenden Prozessinstanzen und der Activiti-Engine.

Activiti-Explorer dient der Anzeige von Tasks und dem Starten von Prozessinstanzen.

Activiti-Designer ein Eclipse-Plugin zur Modellierung von Geschäftsprozessen

Kurz vor dem Druck der Diplomarbeit, wurden die Funktionen der Anwendungen Activiti-Probe und Activiti-Explorer sinnvollerweise im Activiti-Explorer gebündelt.

5.2. CEP-Engine

Esper wurde als CEP-Engine bereits kurz in Kapitel 2.3 erwähnt. Auf Basis der Fachstudie [BDK10] wurde dort bereits entschieden, Esper für die Implementierung zu verwenden. Bei Esper handelt es sich um eine freie CEP-Engine, die für Java und für .NET zur Verfügung steht. Der Hersteller Espertech[5] schreibt: „Esper erlaubt die schnelle Entwicklung von Anwendungen, die große Mengen von Nachrichten oder Ereignissen verarbeiten. Esper filtert und analysiert Ereignisse auf verschiedene Weise und reagiert in Echtzeit auf Bedingungen, die von Interesse sind. Esper verwendet eine optimierte Sprache für die Verarbeitung einer großen Anzahl zeitabhängiger Ereignisse“.

Einen Überblick über die Architektur gibt Abbildung 5.3.

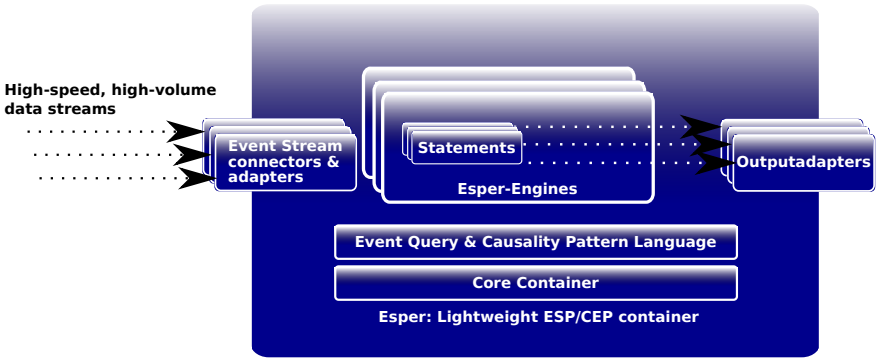


Abbildung 5.3.: Architekturüberblick der CEP-Engine Esper (Quelle: vgl. `http://www.espertech.com/products/esper.php`)

Für die Verarbeitung von Ereignissen sind die *Statements* von zentraler Bedeutung. Ein *Statement* besteht aus einer Regel (*Event-Query* oder *Causality Pattern*) und einem oder mehreren Listenern (im Sinne des Observer-Patterns).

Sobald ein *Statement* beim *CoreContainer* registriert wird, werden also Ereignisse des Ereignis-Stroms von einer *Esper-Engine* analysiert. Wird ein Muster erkannt oder liefert eine Query Ergebnisse, so werden diese ebenfalls in Form von Ereignissen an Listener, die an *Statements* gebunden sind, ausgeliefert und können weiterverarbeitet werden. Listener sind einfache POJOs, die das Interface *com.espertech.esper.client.UpdateListener* implementieren. Esper lässt sich somit gut in (bestehende) Java-Anwendungen integrieren[6].

Die Ergebnis-Ereignisse, die von Esper produziert werden, können also über eine eigene Implementierung in einer beliebigen Java-Anwendung weiterverarbeitet werden. Außerdem

[5] `http://www.espertech.com`
[6] `http://www.espertech.com/products/esper.php`

stellt Esper *Adapter* zur Verfügung, um Ereignisse direkt per Java Messaging Service (JMS) zu publizieren.

6. Architektur

In diesem Kapitel wird eine Architektur beschrieben, die es ermöglicht, die Anforderungen aus Kapitel 4 umzusetzen. Natürlich kann die Architektur hier nicht mit einem Detaillierungsgrad wie in einem Entwurfsdokument eines realen Softwareprojekts beschrieben werden. So verzichte ich unter anderem komplett auf einen Entwurf der Benutzungsschnittstellen und beschränke mich an vielen Stellen bewusst darauf, das Prinzip deutlich zu machen.

Mein Ansatz besteht darin, bestehende WfMSe mit bestehenden Systemen zur Ereignis- bzw. Situationsverarbeitung zu integrieren. Bei der Architektur des Systems orientiere ich mich dabei konzeptionell am Referenz-Modell der WfMC. Abbildung 6.1 zeigt auf der linken Seite das WfMC Referenz-Modell für WfMSe und auf der rechten Seite analog dazu ein System zur Ereignis- und Situationsverarbeitung. Diese Diplomarbeit beschäftigt sich im Wesentlichen mit dem *Rule Enactment*, also der Ausführung.

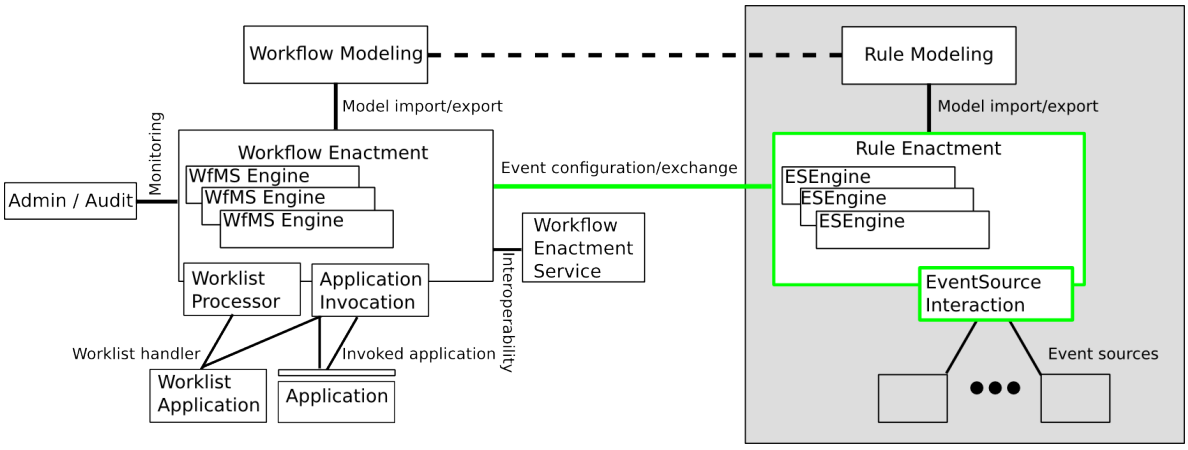


Abbildung 6.1.: Überblick im Kontext des WfMC-Referenz-Modells (basierend auf dem Workflow Reference Model Diagram `http://www.wfmc.org/reference-model.html` der WfMC)

Abbildung 6.2 gibt einen Gesamtüberblick über die Architektur. Der Architekturüberblick zeigt folgende zentrale Komponenten, die implementiert werden müssen:

- ESCStore
- ESEngine
- InstanceManager

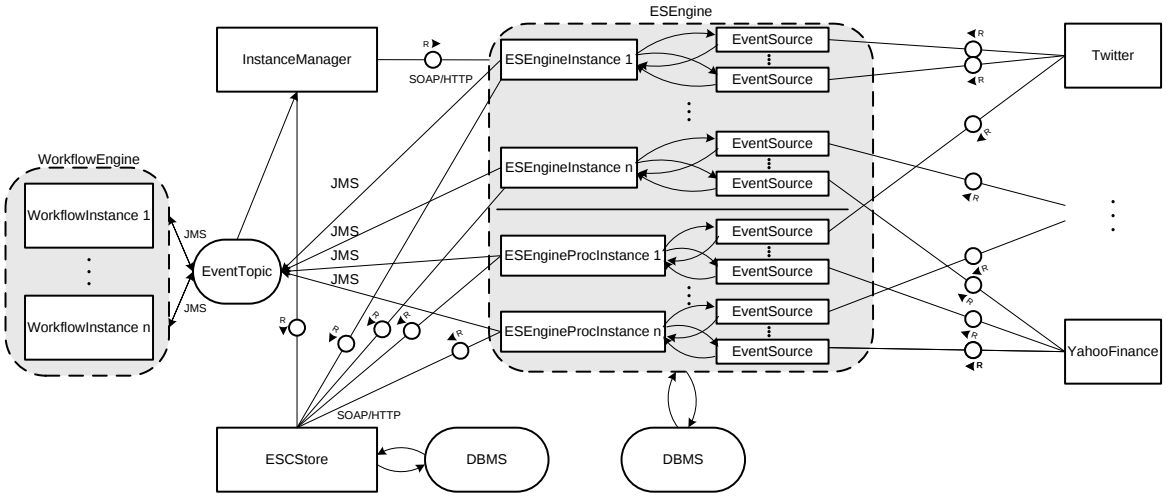


Abbildung 6.2.: Überblick Gesamtarchitektur

Die anderen Komponenten sind eine *Workflow-Engine*, verschiedene Datenquellen (Twitter, YahooFinance, ...), eine Message oriented Middleware (MoM) und ein DBMS. Die Integration dieser fertigen Komponenten wird im Abschnitt 6.3 beschrieben.

6.1. Komponenten

Im Folgenden werden zunächst die zu implementierenden Komponenten beschrieben, ehe dann genauer auf die Datenformate und die Kommunikation eingegangen wird.

6.1.1. ESCStore

Der Name ESCStore steht für Event Situation Configuration Store und dient der Verwaltung von Regeln und Ereignisquellenkonfigurationen. Der ESCStore soll Regeln unterschiedlichster Ereignis- bzw. Situationsverarbeitungssysteme und verschiedenster Ereignisquellen (siehe 6.1.2) speichern. Er bietet eine Web Service-Schnittstelle (siehe A.2) an. Natürlich wäre es möglich, auf den ESCStore zu verzichten und alle Regeln zur Laufzeit direkt zu übermitteln.

Mehrere Punkte sprechen jedoch dafür, den ESCStore wie im Folgenden beschrieben zu realisieren:

Wiederverwendbarkeit Verschiedene Prozesse können per Deklaration die gleichen Regeln verwenden.

Transparenz Der Prozessmodellierer muss kein Regelexperte sein und kann die Regeln deklarativ verwenden. Es kann also klar zwischen den Rollen Regelmodellierer und Prozessmodellierer unterschieden werden.

Entkopplung Die Optimierung von Regeln erfordert keine Änderung eines oder mehrerer Prozesse, da die Regeln per Name im Prozess deklariert werden.

Die Abbildung 6.3 zeigt die *ESCStore*-Klasse und das Datenmodell für die Regeln und Ereignisquellenkonfigurationen, die im ESCStore abgelegt werden.

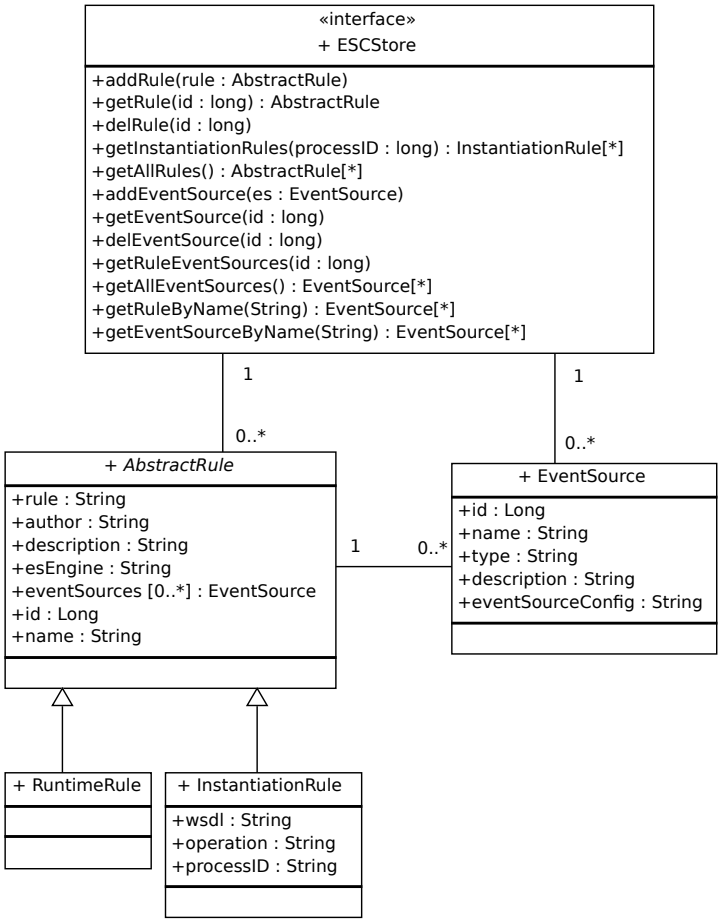


Abbildung 6.3.: Klassendiagramm der ESCStore Komponente

Bezüglich der Regeln gilt es abzuwägen, ob Regeln im ESCStore workflow-spezifisch sind, also jedes Prozess-Modell seine eigenen Regeln hat, oder ob die Regeln global definiert werden und dann per Deklaration festgelegt wird, welche Regeln welcher Prozess verwendet. Gegen den ersten Ansatz spricht, dass Regeln bei der Erstellung neuer Prozesse immer wieder kopiert werden müssen, auch wenn die gleichen Ereignisse verarbeitet werden. Der Ansatz die Regeln so zu speichern, dass sie für alle Prozesse verwendbar sind, hat aber ebenfalls Nachteile. Wird eine Regel in ihrer Semantik für einen Prozess geändert, kann dies zu Problemen bei anderen Prozessen führen. Die Deklaration der zu verwendenden Regeln in der Configuration Task wird aufwändiger. Dennoch habe ich mich dazu entschieden, die Regeln global zu definieren und die Verwendung explizit in der Configuration Task zu deklarieren. Dieser Ansatz bietet die Flexibilität, zu verwendende Ereignisse/Regeln erst zur Laufzeit, d.h. nach der Instantiierung, festzulegen.

Es gibt zwei Arten von Regeln. Regeln, die zur Laufzeit von einer Prozessinstanz benötigt werden, werden im Folgenden als Laufzeitregeln bezeichnet. Im Klassendiagramm heißt die entsprechende Klasse *RuntimeRule*. Regeln, die dazu dienen, eine Prozessinstanz zu erzeugen, werden als Instantiierungsregeln (*IntantiationRule*) bezeichnet. Folgende Informationen werden für jede Regel gespeichert:

Name Der Name der Regel

ESEngine Name der ESEngine-Implementierung (z.B. Esper), die diese Regel verarbeiten soll

Rule Die engine-spezifische Regel

Author Der Name des Autors

Description Natürlichsprachliche Beschreibung der Regel

EventSources* Dieses Attribut ist optional und enthält ggf. die Konfiguration von Ereignisquellen, die von der Regel benötigt werden.

Instantiierungsregeln benötigen zusätzlich folgende Informationen:

ProcessID Die ID des Prozesses, der gestartet werden soll

WSDL Die WSDL des Prozesses

Operation Die Operation, die zum Starten/Instantiieren aufgerufen werden muss

Neben den Regeln kann der ESCStore auch Konfigurationen für EventSources verwalten. Standardmäßig wird die Konfiguration aus dem ESCStore geladen, um die Benutzung des Systems für unerfahrene Benutzer möglichst einfach zu gestalten. Um trotzdem flexibel zu sein, soll es möglich sein, EventSources in der Configuration Task zu konfigurieren (siehe 6.1.2). Wird die Konfiguration in der Configuration Task vorgenommen, wird die vorgenommene und nicht die im ESCStore gespeicherte Konfiguration verwendet.

6.1.2. ESEngine

Der Name ESEngine steht für Event Situation Engine. Die ESEngine stellt also die Komponente zur Ereignisverarbeitung bzw. Situationserkennung dar. Die ESEngine bietet eine Web Service-Schnittstelle, die direkt auf das Interface ESEngine (siehe Abbildung 6.4) abgebildet wird.

Neben dem Interface *ESEngine* sind vor allem die Interfaces *ESEngineInstance* und *EventListener* von Bedeutung. Um externe Ereignisse zu abonnieren und verarbeiten zu können, muss außerdem das Interface *EventSource* (siehe 6.1.2) für alle gewünschten Ereignisquellen implementiert werden.

Um ein existierendes System zur Ereignisverarbeitung bzw. Situationserkennung zu nutzen, muss also eine Mapping-Schicht implementiert werden, die das konkrete System auf die Interfaces *ESEngine* und *ESEngineInstance* abbildet.

Das *ESEngine*-Interface stellt Methoden zur Verwaltung von ESEngine-Instanzen zur Verfügung, die direkt über den Web Service angesprochen werden sollen.

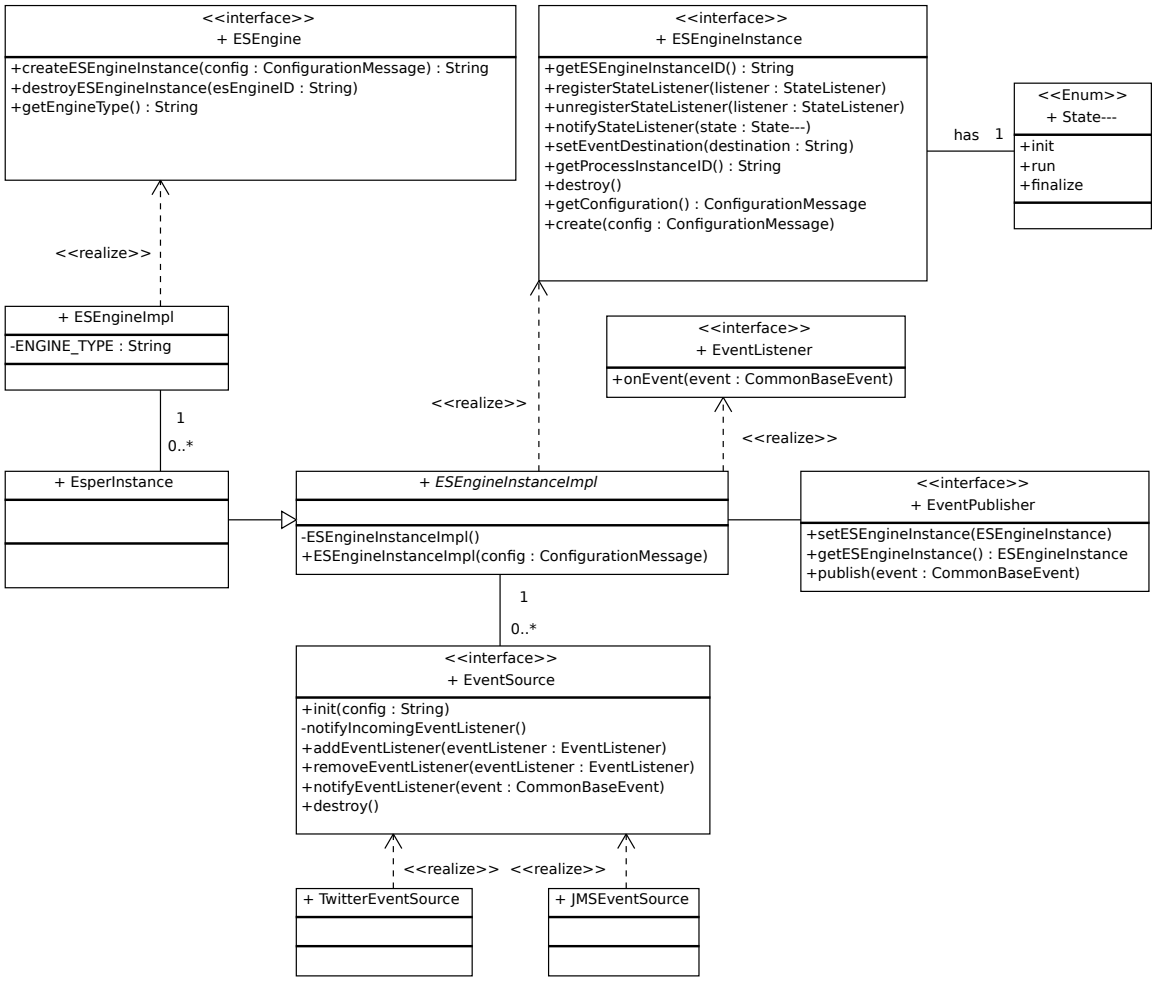


Abbildung 6.4.: Klassendiagramm der ESEngine-Komponente

Um Workflow-Instanzen per Ereignis erzeugen zu können, muss für jedes Workflow-Modell, für das eine Instantiierungsregel existiert, eine ESEngine-Instanz (in der Abbildung 6.2 *ESEngineProcInstance* genannt) gestartet werden, und es müssen die entsprechenden Regeln geladen werden. Für Instantiierungsregeln erzeugt der ESCStore die ConfigurationMessage und publiziert diese über das EventTopic, aus dem der InstanceManager die ConfigurationMessage liest und eine entsprechende ESEngine-Instanz erzeugt. Eintreffende ConfigurationMessages werden von einer ESEngine persistent gespeichert, um im Fehlerfall bei der Recovery des Systems bestehende ESEngine-Instanzen wieder herstellen zu können. Das persistente Speichern von Daten, die aktuell von den ESEngine-Instanzen verarbeitet werden sollen, ist nicht sinnvoll, da es sich um eine große Anzahl von Daten handelt, die möglichst schnell verarbeitet werden soll.

EventSource

Das *EventSource*-Interface enthält Methoden-Signaturen, um von verschiedensten Ereignisquellen (z.B. Twitter, YahooFinance, Sensoren, ...) zu abstrahieren. Die EventSource dient zum einem dem Abonnement von Ereignissen und zum anderen der Transformation von Ereignissen in das intern verwendete Common Base Event (CBE)-Ereignisformat (siehe 6.2.2). Um vor der Verarbeitung, durch die jeweilige ESEngine-Implementierung (z.B. Esper), bereits eine Vorverarbeitung der Ereignisse durchzuführen, kann ebenfalls das Konzept der EventSource verwendet werden. Es ist so z.B. möglich, Ereignisse zu klassifizieren und erst dann an die jeweilige ESEngine weiterzuleiten. Die Anbindung der EventSource an das restliche System folgt dem Observer-Pattern.

Eine Implementierung des Interface *EventSource* erlaubt die Einbindung praktisch beliebiger Ereignisquellen. Es sind also Ereignisquellen wie JMS-Topics / Queues genauso denkbar wie AJAX-HTTP-Streams oder ganz andere Quellen.

Um eine neue Ereignisquelle hinzuzufügen, muss also lediglich das Interface *EventSource* implementiert werden.

6.1.3. InstanceManager

Der InstanceManager übernimmt die Erzeugung von ESEngine-Instanzen. Grundsätzlich besteht die Möglichkeit, die Funktionalität auch in der Komponente ESEngine zu realisieren. Die separate Implementierung hat aber viele Vorteile. Die Funktionalität muss nur einmal realisiert werden, auch wenn es mehrere verschiedene ESEngine-Implementierungen (Esper, Oracle CEP, ...) geben kann. Ein weiterer Vorteil ist, dass nicht jede ESEngine-Implementierung alle ConfigurationMessages empfangen und prüfen muss, ob das Attribut *engineType* dem eigenen Typ entspricht und somit die Engine für die Ausführung verwendet werden soll. Der *InstanceManager* ruft eine ESEngine des entsprechenden Typs auf. Es ist also möglich, an dieser Stelle eine Lastverteilung vorzunehmen, falls verschiedene ESEngines des gleichen Typs (z.B. Esper) zur Verfügung stehen. Dies ist insbesondere dann interessant, wenn eine große Anzahl von Prozessinstanzen und Prozessen existieren oder es gar mehrere Workflow-Engines gibt. Um die verschiedenen ESEngines zu verwalten, würde eine Art Verzeichnis benötigt. Die Implementierung der Lastverteilung wird im Weiteren nicht mehr Gegenstand der Diplomarbeit sein. Es sollte lediglich begründet werden, warum der InstanceManager als eigenständige Komponente ausgelagert wird und nicht als Teil der ESEngine implementiert werden sollte.

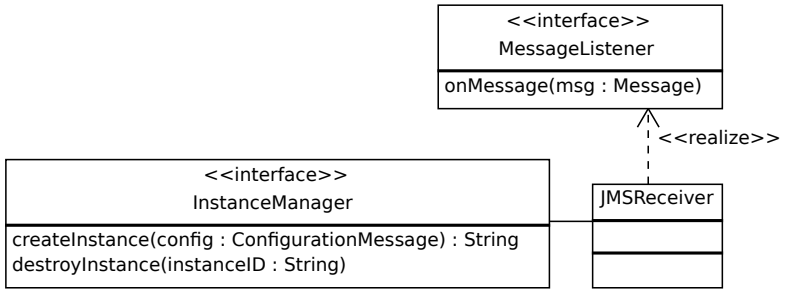


Abbildung 6.5.: Klassendiagramm der InstanceManager-Komponente

6.2. Datenmodellierung

Als Datenformate werden, soweit möglich, verbreitete, standardisierte Formate eingesetzt. Stehen keine passenden Formate zur Verfügung, wird auf XML als Technologie zurückgegriffen, und es werden eigene XML-Schemata definiert.

6.2.1. ConfigurationMessage

Die ConfigurationMessage ist eine Nachricht zur Initialisierung einer ESEngine. Die ConfigurationMessage besteht aus zwei Bereichen. Der erste Bereich, im Folgenden EventDeclaration genannt, deklariert, welche Ereignisse von der Workflow-Instanz verwendet werden. Der zweite Bereich konfiguriert die EventSource, welche die Ereignisse für die Verarbeitung in der ESEngine liefern sollen. Der Bereich wird im Folgenden SourceConfiguration genannt. Die SourceConfiguration ist so gestaltet, dass neue Ereignisquellen möglichst einfach verwendet werden können. Um eine neue Ereignisquelle nutzen zu können, muss das Interface *de.uni_stuttgart.informatik.eventum.esengine.management.EventSource* (siehe 6.4) implementiert werden.

Das einer ConfigurationMessage zugrunde liegende XML-Schema befindet sich im Anhang unter A.2.

6.2.2. Ereignisformat

Als Ereignisformat wird das Format der CBE-Spezifikation von IBM[1] verwendet. „Die Common Base Event Spezifikation definiert einen neuen Mechanismus, um Ereignisse in Geschäftsanwendungen zu verwalten ...“[2]. Das CBE-Format wurde wohl ursprünglich

[1] `http://www.ibm.com`
[2] `http://www.ibm.com/developerworks/library/specification/ws-cbe/`

ausschließlich für Maschinenereignisse vor allem im Bereich Netzwerke und nicht für CEP entwickelt. Es lässt sich aber nutzen, auch wenn einige Attribute nicht unbedingt sinnvoll erscheinen.

Prinzipiell können alle Möglichkeiten, die das XML-Schema bietet, von einer Ereignisquelle verwendet werden. Um jedoch zu zeigen, wie eine sinnvolle Verwendung aussehen könnte, zeige ich hier ein Beispiel für ein Ereignis im CBE-Format aus dem Szenario in Kapitel 8.1. Das Ereignis wurde von der TwitterEventSource erzeugt.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commonBaseEventType creationTime="2011-06-22T13:03:41.836+02:00" extensionName="Tweet"
    xmlns:ns2="http://www.ibm.com/AC/commonbaseevent1_0_1">
    <ns2:extendedDataElements type="String" name="category">
        <ns2:values>neutral</ns2:values>
    </ns2:extendedDataElements>
    <ns2:extendedDataElements type="String" name="text">
        <ns2:values>Welche Rolle #socialmedia bei Bayer MaterialScience spielt:
            http://t.co/yCPgp6p #socbiz #Wikis #Blogs #casestudy #lesenswert</ns2:values>
    </ns2:extendedDataElements>
    <ns2:sourceComponentId componentType="WebApplication" locationType="Hostname"
        location="twitter.com" componentIdType="ServiceName" subComponent="twitter.com"
        component="twitter.com"/>
    <ns2:situation categoryName="OtherSituation">
        <ns2:situationType xsi:type="ns2:OtherSituation" reasoningScope="EXTERNAL"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
    </ns2:situation>
</commonBaseEventType>
```

Die von der CBE-Spezifikation definierten Attribute sind zu einem großen Teil optional. Lediglich die Attribute bzw. Elemente *creationTime*, *sourceComponentId* und *situation* müssen, gemäß der Spezifikation, vorhanden sein.

Attribut	**Beschreibung**
creationTime	Datum und Uhrzeit zu denen das Eintritt eingetreten ist.
sourceComponentId	Id der Komponente die von dem Ereignis oder Situation betroffen ist.
situation	Gibt den Typ der Situation angibt die zu dem Ereignis geführt hat.

Tabelle 6.1.: Common Base Event: Benötigte Attribute

Um Ereignisquellen-spezifische Daten auszudrücken, definiert das CBE-Format sogenannte *extendedDataElements*. Ein *extendedDataElement* hat die Attribute *type* und *name* sowie ein Element *value*.

Die CBE-Spezifikation erlaubt die folgenden Datentypen sowie die entsprechenden Array-Typen:

- byte
- short
- int
- long
- float
- double
- string
- dateTime
- hexBinary
- boolean

Bei der Implementierung einer EventSource muss also darauf geachtet werden, dass alle benötigten Attribute gesetzt werden und nur die erlaubten Datentypen verwendet werden.

Weitere Informationen können der CBE-Spezifikation unter `http://www.ibm.com/developerworks/autonomic/books/fpy0mst.htm#HDRAPPA` entnommen werden.

6.3. Kommunikation und Datenintegration

Für die Kommunikation werden zum einen Web Services[3] (SOAP über HTTP) und zum anderen JMS[4] verwendet.

Eine Entkopplung von Workflow-Engine und den Systemen zur Ereignis- bzw. Situationsverarbeitung ist zwingend notwendig, um die Workflow-Engine vor einer Überflutung von Ereignissen zu schützen. Asynchronität entspricht dem Charakter von Ereignissen. Für die Kopplung der Systeme wird aus diesem Grund eine MoM eingesetzt. Die Entkopplung der Systeme per MoM trägt zur Robustheit und Erweiterbarkeit bei. Falls einzelne Teilsysteme ausfallen oder nicht erreichbar sind, führt das in der Regel nicht dazu, dass das komplette System ausfällt. Durch die Verwendung einer MoM erhält man außerdem eine Indirektion beim Nachrichtenaustausch. Dies führt zu einer guten Erweiterbarkeit, da die Komponenten sich nicht kennen müssen.

Als MoM wird eine JMS-Implementierung eingesetzt. Diese Implementierung stellt das *EventTopic* zur Verfügung über das die Kommunikation zwischen WfMS, *ESEngine* und *InstanceManager* stattfindet (siehe Abbildung 6.2).

[3] `http://www.w3.org/TR/ws-arch/`
[4] `http://www.oracle.com/technetwork/java/jms/index.html`

Das Sequenzdiagramm in Abbildung 6.6 zeigt exemplarisch den Ablauf der Kommunikation.

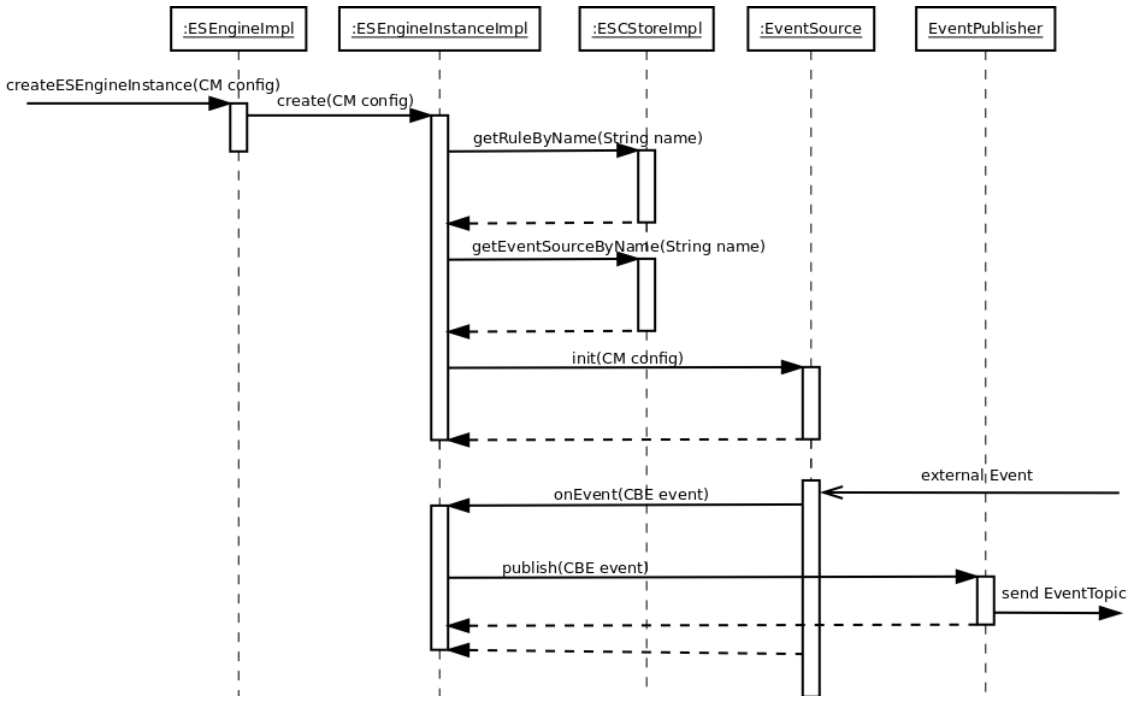


Abbildung 6.6.: Sequenzdiagramm: Erstellung einer ESEngine-Instanz; Abruf von Daten aus dem *ESCStore*; Initialisierung einer *EventSource*; Empfang eines Ereignisses

Das EventTopic ist, wie bereits erwähnt, das Austauschmedium zwischen einer Workflow-Engine und dem übrigen System. Trifft eine ConfigurationMessage ein, ruft der InstanceManager den Web Service einer passenden ESEngine auf. Passend bedeutet in diesem Fall, dass die ESEngine vom richtigen Typ ist, der Typ der ESEngine also mit dem Attribut *engineType* in der ConfigurationMessage (siehe A.2) übereinstimmt.

In einer realen Umgebung kann es sinnvoll sein, nicht nur einen sondern mehrere InstanceManager zu haben. So wird das Problem eines Single Point of Failure oder Bottlenecks umgangen. Es müsste dann eine zusätzliche Queue verwendet werden, um sicherzustellen, dass genau ein InstanceManager die ConfigurationMessage liest.

Innerhalb der ESEngine-Komponente wird beim Aufruf von *createESEngineInstance* eine Instanz der Implementierung von *ESEngineInstance* erzeugt. Danach wird der zugehörige EventListener konfiguriert, indem die SourceConfiguration übergeben wird. Im Detail bedeutet dies, dass eine EventSource-Implementierung erzeugt und konfiguriert wird. Dann registriert sich der EventListener als Listener bei der EventSource-Implementierung. Beim Eintreffen eines Ereignisses bei der EventSource-Implementierung wird das entsprechende Ereignis in das CBE-Format transformiert und dann an den EventListener weitergereicht,

von der ESEngine weiterverarbeitet, und schließlich werden die Ergebnisereignisse an das EventTopic weitergeleitet und können von der Workflow-Engine konsumiert werden.

Für die Kommunikation der Ereignisproduzenten (z.B. Twitter, YahooFinance, Sensoren, ...) kommen unterschiedlichste Protokolle in Frage. Die Protokolle sind durch den Ereignisproduzenten vorgegeben.

7. Implementierung

Dieses Kapitel beschreibt die konkrete Umsetzung des in Kapitel 6 skizzierten Entwurfs. Auf die Implementierung von grafischen Benutzungsoberflächen wird verzichtet. Lediglich das Eclipse-Plugin von Activiti zur Modellierung von Prozessen wurde um zwei Aktivitäten erweitert.

Die Implementierung integriert die beiden Systeme Activiti[1] als Workflow-Engine und Esper[2] als CEP-Engine. Bei der Implementierung wurde viel Wert auf die Trennung zwischen einem anwendungsspezifischem Code (hier also Activiti und Esper) und einem Code, der die Basis für die Integration darstellt, gelegt. Es handelt sich also lediglich um eine Art Referenzimplementierung, die sich auch leicht auf andere Workflow-Engines und insbesondere andere Systeme zur Ereignisverarbeitung übertragen lässt.

7.1. Vorgehen bei der Implementierung

Im Folgenden wird das grobe Vorgehen bei der Implementierung beschrieben.

7.1.1. Hauptkomponenten

Bei der Implementierung wurde großer Wert auf die Trennung zwischen Datenmodell und Anwendungslogik gelegt. Für die Komponenten ESCStore und ESEngine wurden zunächst die Datenmodelle entwickelt. Dazu wurden XML-Schemata entwickelt und per Java Architecture for XML Binding (JAXB) Java-Klassen erzeugt. Die generierten Klassen wurden um Java Persistence API (JPA)-Annotationen ergänzt. Auf diese Weise wird beschrieben, wie Daten in der Datenbank persistent abgelegt werden. Es wird also letztendlich das Datenbankschema mithilfe von Annotationen beschrieben. Aus meiner Sicht sind sowohl Lese- als auch Schreiboperationen für die Performance des Systems von untergeordneter Bedeutung. Schreiboperationen finden im Wesentlichen nur beim Hinzufügen von neuen Regeln oder Ereignisquellen-Konfigurationen statt. Leseoperationen finden bei der Instantiierung von ESEngines statt. Aus diesem Grund ist in diesem Fall nichts gegen den Einsatz eines Object-Relational-Mapper (ORM) einzuwenden. Der Standardweg in Java ist die Nutzung der JPA. Als Implementierung wurde die Referenzimplementierung EclipseLink[3] gewählt.

[1] http://www.activiti.org
[2] http://esper.codehaus.org
[3] http://www.eclipse.org/eclipselink/

Als DBMS wird H2[4] eingesetzt. Das DBMS kann aufgrund der Verwendung der JPA aber leicht ausgetauscht werden.

Die eigentliche Verarbeitung von Ereignissen bzw. Situationen erfolgt durch entsprechende Engines. Im konkreten Fall wurde Esper[5] verwendet. Esper ist, wie bereits in Kapitel 5.2 erwähnt, in der Lage, eine große Anzahl von Ereignissen mit hoher Geschwindigkeit zu verarbeiten. Ein persistentes Speichern von Ereignissen würde die Verarbeitungsgeschwindigkeit von Esper stark reduzieren. Dies ist mit den Zielen, eine möglichst große Anzahl von Ereignissen mit möglichst geringer Verzögerung zu verarbeiten, nicht vereinbar. Esper sieht trotzdem Konzepte zur Speicherung historischer Daten vor. Diese werden hier aber nicht weiter betrachtet, da es nur schwer möglich ist, ein allgemeingültiges Konzept auf dieser Basis zu entwickeln.

Nach der Entwicklung der Datenmodelle wurden per Web Service Description Language (WSDL) die Web Services beschrieben und per Java API for XML - Web Services (JAX-WS) die entsprechenden Skeletons generiert. Standardmäßig generiert JAX-WS das Datenmodell für Web Services automatisch. Mithilfe von sogenannten Binding-Files kann allerdings ein zuvor mit JAXB[6] generiertes Datenmodell verwendet werden. Dies hat mehrere Vorteile:

- Die Datenmodelle können von verschiedenen Web Services wiederverwendet werden, und das Datenmodell muss nicht für jeden Web Service erneut generiert werden.
- Änderungen an der Schnittstelle erfordern nicht unbedingt eine Änderung des Datenmodells und umgekehrt.

7.1.2. Activiti

Activiti (siehe Kapitel 5) ist eine vollständig in Java implementierte BPMN 2.0-Workflow-Engine. Activiti unterstützt nur eine Teilmenge der Elemente von BPMN 2.0 und wird wahrscheinlich wie auch andere BPMN-Engines nie alle Elemente implementieren. Leider sind bis zur Activiti-Version 5.4 keine relevanten Elemente zur Ereignisverarbeitung implementiert. Die Unterstützung zur Erweiterung ist aber gegeben. Für sogenannte ServiceTasks kann ein eigenes Verhalten implementiert werden. Außerdem lässt sich die Palette des Eclipse-Designers sehr leicht erweitern. Es besteht somit die Möglichkeit, eigene Aktivitäten zu realisieren.

Im Rahmen der Diplomarbeit habe ich zwei Aktivitäts-Typen implementiert, die der Interaktion mit dem für Ereignisverarbeitung zuständigen System dienen. Die Configuration Task ermöglicht die Auswahl der vom Prozess benötigten Ereignisse und optional die Konfiguration von Ereignisquellen. Die Receive Event Task wartet bis ein bei der Modellierung festgelegtes Ereignis eintritt und speichert dieses Ereignis in einer zuvor festgelegten Variable.

[4] http://www.h2database.com
[5] http://esper.codehaus.org
[6] http://jaxb.java.net/

Die Aktivität für den Empfang von Ereignissen ist nichts anderes als ein JMS-Subscriber mit MessageSelector, um nur die für diesen Prozess relevanten Ereignisse zu empfangen.

Wie in Kapitel 5 bereits erwähnt, bietet Activiti zwei verschiedene Möglichkeiten zur Modellierung von Prozessen an - zum einen das angepasste Modellierungswerkzeug von Signavio und zum anderen ein Eclipse-Plugin. Diese beiden Anwendungen spiegeln auch die Sichten auf einen Workflow wider. Während das Eclipse-Plugin von Prozessingenieuren für die technische Umsetzung genutzt wird, wird ein Prozessanalyst eher das Werkzeug von Signavio benutzen. Da es bei der Umsetzung der Verarbeitung von Ereignissen eher um technische Details geht, die sich von einer abstrakten Sichtweise eines Prozessanalysten unterscheiden wird, wurde im Rahmen der Diplomarbeit lediglich eine Erweiterung für das Eclipse-Plugin entwickelt. Dies bedeutet, dass die oben beschriebenen Aktivitäten, wie in Abbildung 7.1 dargestellt, in der Palette des Eclipse-Designers zur Verfügung stehen.

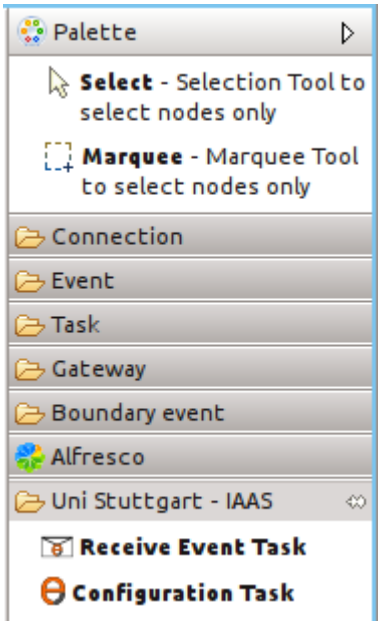


Abbildung 7.1.: Erweiterung der Palette des Eclipse-Designers von Activiti um zwei Aktivitäten

7.2. Erweiterungen

Im Folgenden wird skizziert, welche Erweiterungsmöglichkeiten das System anbietet und wie neue Systeme zur Ereignis- bzw. Situationsverarbeitung oder Workflow-Engines integriert werden können.

EventSources

Wie bereits unter 6.1.2 beschrieben, dienen EventSources dazu, externe Ereignisse zu empfangen. Um eine einfache Erweiterbarkeit zu gewährleisten, werden Ereignisquellen per Reflection zur Laufzeit geladen. So können leicht neue Ereignisquellen hinzugefügt werden. Ein entsprechendes Java Archive (JAR), das eine oder mehrere Implementierung(en) des Interface *de.uni_stuttgart.informatik.eventum.esengine.management.EventSource* enthält, muss sich im Classpath befinden. Verwendet werden können solche Ereignisquellen, indem das Attribut *type* einer EventSource in der ConfigurationMessage (siehe A.2) oder beim Hinzufügen im ESCStore auf den vollständig qualifizierten Klassennamen der Implementierung gesetzt wird.

Neue Ereignisquellen sind, wie in 6.1.2 beschrieben, eine Abstraktion, um grundsätzlich beliebige externe Ereignisquellen (z.B. Twitter, Yahoo Finance, Sensoren) einzubinden. Um eine neue Ereignisquelle bereitzustellen, muss das Interface *de.uni_stuttgart.informatik.eventum.esengine.management.EventSource* implementiert oder die abstrakte Klasse *de.uni_stuttgart.informatik.eventum.esengine.management.AbstractEventSource* erweitert werden. *AbstractEventSource* stellt Implementierungen für die Verwaltung der EventListener bereits zur Verfügung und beschleunigt so die Implementierung neuer Ereignisquellen.

Die Methoden *init* und *destroy* müssen implementiert werden. Die Implementierung der Methode *init* muss den Empfang der Ereignisse von externen Ereignisquellen, z.B. Twitter-Nachrichten, deren Transformation in ein Common Base Event und schließlich den Aufruf von *notifyEventListener(CommonBaseEvents cbe)* beinhalten, um das Ereignis an das System zur Ereignis- bzw. Situationverarbeitung weiterzureichen. In der Methode *destroy* können Aufräumarbeiten, wie z.B. das Schließen von Verbindungen, vorgenommen werden. Natürlich kann eine EventSource-Implementierung grundsätzlich jede mit der Programmiersprache Java realisierbare Funktionalität implementieren. Für bestimmte Fälle könnte es z.B. sinnvoll sein, Ereignisse vorzuverarbeiten (z.B. die Klassifikation mithilfe eines Bayes-Klassifikators).

Integration eines Systems zur Verarbeitung von Ereignissen/Situationen

Wie bereits oben erwähnt, wurde eine möglichst strikte Trennung zwischen anwendungsspezifischem Quellcode, und Quellcode zur Verwaltung von Regeln, Ereignisquellen und Systemen zur Ereignis- bzw. Situationsverarbeitung vorgenommen. Um ein neues System zur Ereignis- bzw. Situationsverarbeitung zu integrieren, muss eine entsprechende Implementierung für *ESEngine.wsdl* realisiert werden. Außerdem muss die abstrakte Klasse *de.uni_stuttgart.informatik.eventum.esengine.management.ESEngineInstanceImpl* erweitert werden. Die Implementierung dieser Klasse stellt eine Instanz einer Engine für Ereignis- bzw. Situationsverarbeitungen dar. Die Implementierung des Web Service erstellt für jede Prozessinstanz eine solche Engine-Instanz.

Das Laden von Regeln oder Konfigurationen für Ereignisquellen ist für den Entwickler transparent. Er kann über die Methode *getRules()* der abstrakten Klasse *ESEngineManagementImpl* direkt auf die Regeln zugreifen. Ereignisquellen werden automatisch geladen und konfiguriert.

Um andere Systeme zur Ereignis- bzw. Situationsverarbeitung als Esper zu unterstützen, muss die Implementierung des InstanceManager entsprechend angepasst werden. Falls eine ConfigurationMessage (siehe A.2) mit dem entsprechenden *engineType* eintrifft, muss die Operation *createInstance(ConfigurationMessage config)* einer ESEngine dieses Typs aufgerufen werden.

Integration einer anderen Workflow-Engine

Die Workflow-Engine kommuniziert ausschließlich per JMS mit den anderen Komponenten. Dies wurde bereits in Kapitel 6.3 beschrieben. Aus Sicht der Workflow-Engine gestaltet sich die Schnittstelle folgendermaßen. Die Workflow-Engine muss zum einen eine ConfigurationMessage (siehe A.2) erzeugen und über eine *javax.jms.ObjectMessage* im Topic bereitstellen. Dabei muss die JMS-Nachricht die Eigenschaft *InitializationMessage = true* enthalten. Ereignisse für die Workflow-Engine werden im gleichen Topic zur Verfügung gestellt. Die JMS-Nachrichten enthalten zwei Eigenschaften:

CorrespondingProcessID Ermöglicht es, das Ereignis einer Prozessinstanz zuzuordnen.

EventType Enthält den Typ des Ereignisses. Der JMSEventPublisher (siehe ref-fig:instancemanager) setzt die Eigenschaft *eventType* auf den Wert des Attributs *ExtensionName* eines CommonBaseEvents. Der ExtensionName sollte von jeder *EventSource* gesetzt werden.

Weitere Funktionalität

Nicht implementiert wurde das Konzept der InstantiationRules. Die InstantiationRules sollen dazu dienen, eine Prozessinstanz per Ereignis zu erzeugen. Der ESCStore implementiert alle dafür notwendigen Konzepte. Lediglich der InstanceManager muss erweitert werden. Eine einfache Möglichkeit der Implementierung besteht darin, für jeden Prozess mit InstantiationRules eine eigene ESEngine-Instanz zu erzeugen, die alle InstantiationRules für diesen Prozess verarbeitet. Außerdem muss eine Komponente implementiert werden, die bei Eintreten eines Ereignisses eine Prozessinstanz bei der Workflow-Engine erzeugt.

8. Evaluation

Im Nachhinein zeigt sich, dass sämtliche neue BPMN-Engines wohl noch einige Zeit benötigen werden, bis mehr Konzepte, die die Ereignisverarbeitung betreffen, unterstützt, werden. Vielleicht zeigt sich hierin auch, wie komplex die Implementierung der Elemente von BPMN 2.0 zur Ereignisverarbeitung ist. Sollte dieser Schluss richtig sein, wäre mein Ansatz bestätigt, die Ereignisverarbeitung im eigentlichen Sinne auszulagern und Systeme dafür zu verwenden, die explizit dafür entwickelt wurden (z.B. ein CEP-System wie Esper).

Bei der für die Implementierung im Rahmen dieser Diplomarbeit eingesetzten Workflow-Engine Activiti hat sich jedenfalls in Hinsicht auf die Unterstützung von Ereignissen, während ich diese Diplomarbeit verfasst habe, keine nennenswerte Weiterentwicklung gezeigt. Trotzdem lassen sich viele Fälle mit der entstandenen Implementierung lösen, ohne dass eine Implementierung der BPMN 2.0 Konzepte in Activiti vorhanden ist. Dazu wurde im Rahmen der Diplomarbeit die Receive Event Task entwickelt (siehe 7.1.2), die auf das Eintreten eines Ereignisses wartet.

Im Folgenden werden einige Grundmuster gesammelt, die bei der Verarbeitung von Ereignissen mit BPMN auftreten (siehe auch 2.3), und es wird eine Abbildung auf die entwickelte Receive Event Task vorgenommen. Die Auflistung der Grundmuster in Tabelle 8.1 erhebt keinen Anspruch auf Vollständigkeit, sollte aber nahezu alle Fälle abdecken.

Das Fehlen einiger Elemente der BPMN-Spezifikation in Activiti lässt sich nicht immer kompensieren. Deshalb versuche ich, eine Klassifikation der Muster in drei Klassen vorzunehmen.

Klasse 1 lässt sich vollständig auf das im Rahmen der Diplomarbeit implementierte System abbilden.

Klasse 2 erfordert geringe Erweiterungen der Implementierung, die im Rahmen der Diplomarbeit entstanden ist. Diese Erweiterungen sind in der Architektur bereits vorgesehen.

Klasse 3 Die Semantik lässt sich nicht äquivalent abbilden und erfordert die Implementierung weiterer Elemente der BPMN-Spezifikation. Trotzdem gibt es in vielen Fällen eine adäquate Modellierung.

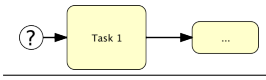
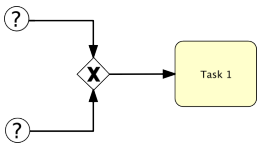
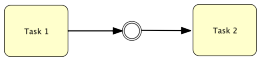
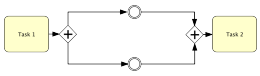
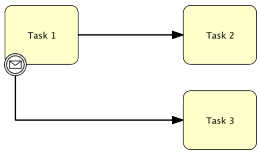
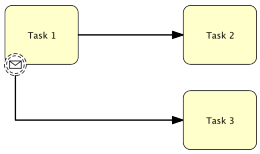
Pattern	Abbildung	Klasse	Kurzbeschreibung
		2	Prozessinstanziierung durch Ereignis
		2	Prozessinstanziierung durch Ereignis A oder Ereignis B, A $\neq$ B
		1	Warten, bis Ereignis eintritt
		1	Paralleles Warten auf Ereignisse
	abhängig vom konkreten Fall	3	Das Eintreten eines Ereignisses führt zum Abbruch von Task 1. Task 3 wird ausgeführt.
	abhängig vom konkreten Fall	3	Nach dem Eintreten eines Ereignisses wird Task 3 gestartet. Task 1 läuft weiter.

Tabelle 8.1.: BPMN Event-Patterns

Wir sehen, dass mit dem entstandenen System die Muster 3 und 4 realisierbar sind. Die Muster 1 und 2 erfordern nur einige geringe Ergänzungen des Systems, die bereits vorgesehen sind (siehe Kapitel 6). Lediglich für 5 und 6 gibt es keine semantisch äquivalente Abbildung. Allerdings ist nicht nur aufgrund von 5 und 6 eine Implementierung der entsprechenden Elemente von BPMN wünschenswert. Vor allem im Hinblick auf Roundtrip-Engineering ist eine einheitliche Modellierung wichtig, um ein einheitliches technisches und fachliches Modell zu haben.

Die Fähigkeiten zur Ereignisverarbeitung im Sinne der Composite Event Patterns aus [BDG07] (siehe 3.1) hängen vor allem von der ESEngine-Implementierung (siehe 6.1.2) ab. Für die Implementierung im Rahmen der Diplomarbeit wurde Esper verwendet. Eine kurze Übersicht, welche Composite Event Patterns von Esper unterstützt werden, gibt Tabelle 8.2.

	Composite Event Pattern	BPEL	BPMN	Diplomarbeit
1.	Event Conjunction	-	-	+
2.	Event Cardinality	-	-	+
3.	Event Disjunction	+	+	+
4.	Inhibiting Event	-	-	+
5.	Event Time Relation	-	-	+
6.	Subscription Time Relation	+/-	+/-	+/-
7.	Consumption Time Relation	-	-	-
8.	Absolute Time Relation	+/-	+/-	+/-
9.	Event Data Dependency	-	-	+
10.	Process Instance Data Dependency	+	-	-
11.	Environment Data Dependency	-	-	+
12.	Consume Once	+	+	+
13.	Consume Multiple Times	-	-	+

+: unterstützt, -: nicht unterstützt, +/-: teilweise unterstützt

Tabelle 8.2.: Composite Event Patterns (basierend auf [BDG07])

Die erste Analyse zeigt, dass der Großteil der in [BDG07] beschriebenen Composite Event Patterns unterstützt werden. Eine tiefergehende Analyse der Möglichkeiten von Esper war aufgrund des Umfangs der Esper-EPL nicht möglich. Der Tabelle liegt also nur eine erste grobe Analyse zugrunde. Ich sehe lediglich Probleme beim Zugriff auf externe Daten zum Zeitpunkt der Regelverarbeitung durch die ESEngine und bei Ereignissen, die abhängig vom Zeitpunkt der Subscription von Ereignissen behandelt werden sollen.

In den Anforderungen wurden einige nicht-funktionale Anforderungen aufgeführt. Eine genaue Performance-Untersuchung würde den Rahmen dieser Diplomarbeit sprengen. Aufgrund der verwendeten Technologien und Komponenten liegt der Schluss jedoch nahe, dass die Performance der formulierten Anforderung entspricht. Die Erweiterbarkeit ist durch den Einsatz von Standards und freien Technologien wie XML, Web Services und JMS und sauber definierten Schnittstellen und Formaten gegeben. Einige Tipps zur Erweiterung wurden in Kapitel 7.2 beschrieben. Lediglich die Anforderung, eine möglichst transparente und einfache Benutzung zu garantieren, gestaltet sich nicht so einfach. Dies ist vor allem darauf zurückzuführen, dass die Sprachen für CEP-Systeme nicht standardisiert und meist sehr kompliziert sind. Transparenz ist nur insofern gewährleistet, dass die Regelverarbeitung und alles, was damit zu tun hat, zumindest für den Prozessmodellierer relativ transparent ist. Für einen Entwickler, der neue ESEngines integrieren will, ist dagegen die Interaktion mit dem ESCStore, sowie das Laden und die Konfiguration von EventSources völlig transparent.

8.1. Szenario

Im Folgenden wird ein einfaches Szenario skizziert, um einen Überblick darüber zu geben, was das entstandene System leisten kann. Ich greife aus mehreren Gründen auf ein sehr einfaches Aktien-Kauf Szenario zurück,

1. um ein möglichst gut verständliches Beispiel zu haben,
2. da die Modellierung von Esper-Regeln (zu Esper siehe Kapitel 5.2) sehr komplex ist,
3. weil Ereignisdaten für das Szenario (Börsenkurse u. Twitternachrichten) leicht verfügbar sind,
4. weil bei der hohen Dynamik am Aktienmarkt eine hohe Verarbeitungsgeschwindigkeit von Vorteil ist,
5. und da sich in dieses Szenario leicht weitere Ereignisquellen einbinden lassen, um eine Bewertung von Aktien vorzunehmen (z.B. RSS-Feeds).

8.1.1. Der Prozess

Der Prozess des Szenarios in Abbildung 8.1 wartet nach der Konfiguration des Systems zur Ereignisverarbeitung mithilfe des *Configuration Task* (siehe im Kapitel 7.1.2) darauf, dass ein Ereignis eintrifft, das den Kauf einer Aktie aufgrund des Kursverlaufs empfiehlt. Danach wartet der Prozess noch auf gute Nachrichten, in diesem Fall auf einen Tweet, zum Unternehmen, dessen Aktien gekauft werden sollen. Im Folgenden werden der Prozess und dessen Ablauf genauer beschrieben.

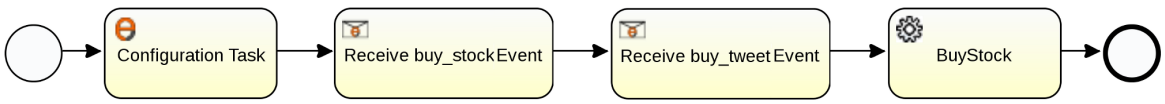


Abbildung 8.1.: Prozess des Szenarios

Nach dem Start des Ereignisses muss zunächst eine *Configuration Task* die Ereignisverarbeitung konfigurieren.

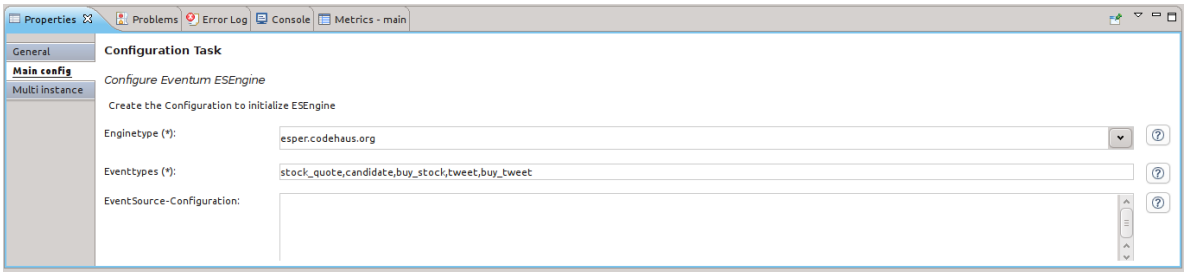


Abbildung 8.2.: Benutzungsoberfläche des Eclipse-Designers, die es erlaubt, Attribute für die *ConfigurationMessage* festzulegen

Die beiden darauf folgenden Receive Event Tasks (*Receive buy_stock Event* und *Reveice buy_tweet* Event) sind Aktivitäten, die auf das Eintreffen eines Ereignisses warten und dieses Ereignis dem Prozess zur Verfügung stellen. Die *Receive buy_stock Event Task* wartet auf das Eintreffen des Ereignisses *buy_stock*. Die Konfiguration ist in Abbildung 8.3 zu sehen. Die nachfolgende Task erwartet dann noch das Eintreten des Ereignisses *buy_tweet*, und danach führt die Aktivität *BuyStock* den Aktienkauf aus. Vernachlässigt wird an dieser Stelle die Tatsache, dass der Prozess nicht beliebig lange auf ein *buy_tweet*-Ereignis warten sollte, sondern *buy_event* und *buy_tweet* in einem unmittelbaren zeitlichen Zusammenhang stehen sollten. Dieses Problem ließe sich aber auf einfache Weise mit einem Timer lösen.

Abbildung 8.3.: Benutzungsoberfläche zur Konfiguration der Receive Event Task - Es wird festgelegt, auf welches Ereignis gewartet werden soll, und wie der Bezeichner der Variable heißen soll, die nach dem Empfang die Ereignisdaten enthält.

8.1.2. Die Ereignisverarbeitung

Für die Ereignisverarbeitung werden zwei Ereignisquellen benötigt, nämlich die für Twitter-Nachrichten (genannt Tweets) und die für Aktienkurse.

Für die Ereignisverarbeitung sind folgende Regeln erforderlich. In dieser Diplomarbeit wird Esper zur Verarbeitung der Ereignisse eingesetzt. Daher wird hier die Sprache von Esper (Esper-EPL) verwendet.

Regeln und Erklärung

Listing 8.1 Regel: stock_event

```
INSERT INTO stock_event SELECT value.values[0] AS value, symbol.values[0] AS symbol FROM
       Event[SELECT * FROM extendedDataElements WHERE name = "l1"] AS value,
       Event[SELECT * FROM extendedDataElements WHERE name = "s"] AS symbol
```

Diese Regel erzeugt *stock_event*-Ereignisse mit den Attributen *value* (Kurs-Wert) und *symbol* (Symbol des Unternehmens), falls ein eingehendes Ereignis die Attribute *l1* und *s* enthält. Bei YahooFinance (einem Dienst, der Aktienkurse zur Verfügung stellt), steht *l1* für den letzten Aktienkurs und *s* für die Abkürzung bzw. das Symbol des Unternehmens.

Listing 8.2 Regel: candidate

```
INSERT INTO candidate SELECT a.symbol AS symbol,
                          a.value AS value,
                          -1+(cast(b.value,float) / cast(a.value,float)) AS change FROM
       pattern[EVERY a=stock_event -> b=stock_event(symbol = a.symbol)]
```

Diese Regel erzeugt *candidate*-Ereignisse mit den Attributen *symbol* und *value*. Für alle Paare von Ereignissen A,B des Typs *stock_event*, für die gilt: auf A folgt B und das Symbol ist für A und B identisch.

Listing 8.3 Regel: buy_stock

```
INSERT INTO buy_stock SELECT symbol,change,value FROM candidate WHERE change > 0.00009
```

Erzeugt ein Ereignis *buy_stock*, wenn der *change*-Wert eines *candiate*-Ereignisses größer als 0.009% ist.

Listing 8.4 Regel: tweet

```
INSERT INTO tweet SELECT category.values[0] AS category, text.values[0] AS text FROM
       Event[SELECT * FROM extendedDataElements WHERE name = "category"] AS category,
       Event[SELECT * FROM extendedDataElements WHERE name = "text"] AS text
```

Erzeugt ein tweet-Ereignis, wenn der Ereignisstrom ein Ereignis mit den *extendedDataElements* *category* und *text* enthält.

Listing 8.5 Regel: buy_tweet

```
INSERT INTO buy_tweet SELECT category, text FROM tweet WHERE category = "buy"
```

Erzeuge ein *buy_tweet* Ereignis, wenn ein tweet-Ereignis das *extendedDataElement* das Attribut *category* mit dem Wert *buy* enthält.

Die Ereignisquellen

Wie in den Kapiteln 6.1.2 und 7.2 beschrieben, dienen *EventSources* dazu, Ereignisse externer Systeme zu abonnieren, zu transformieren, ggf. vorzuverarbeiten und schließlich an das System weiterzuleiten.

TwitterEventSource Diese EventSource erlaubt das Abonnement von Tweets, die bestimmte Begriffe enthalten. Die Tweets werden mithilfe eines Bayes-Klassifikators in die Klassen *sell*, *neutral* und *buy* eingeteilt.

Als Konfiguration erwartet diese EventSource lediglich die durch Kommata getrennten Begriffe.

Ziel ist, Tweets über Aktiengesellschaften zu klassifizieren und somit zu erkennen, ob es sich um gute, neutrale oder schlechte Nachrichten handelt.

YahooStockQuoteEventSource Diese EventSource stellt Börsenkurse des Dienstes YahooFinance[1] zur Verfügung. Die YahooStockQuoteEventSource erwartet zur Konfiguration das Symbol der Aktiengesellschaft und danach per Komma getrennt beliebige Attribute, die von YahooFinance unterstützt werden (einige Details zur YahooFinance API sind z.B. unter `http://brusdeylins.info/projects/yahoo-finance-api/` zu finden).

Der Ablauf

Wie also läuft das beschriebene Szenario nun ab. Nachdem eine Prozessinstanz gestartet wurde, wird die *Configuration Task* ausgeführt. Sie erzeugt eine *ConfigurationMessage*, die im EventTopic platziert wird.

Die ConfigurationMessage wird nun vom InstanceManager aus dem EventTopic gelesen, und es wird eine ESEngine-Instanz entsprechend der ConfigurationMessage vom InstanceManager erzeugt. Die ESEngine-Instanz ruft die entsprechenden Regeln und zugehörigen EventSource-Konfigurationen vom ESCStore ab und erzeugt die EventSource-Instanzen und konfiguriert die ESEngine-Instanz, in diesem Fall Esper.

Die über die EventSources eintreffenden externen Ereignisse werden von Esper entsprechend der Regeln verarbeitet. Die Ergebnisse in Form von Ereignissen werden schließlich über das EventTopic publiziert und so für Activiti zur Verfügung gestellt.

Nach der *Configuration Task* wird die erste *Receive Event Task* ausgeführt. Das *EventTopic* wird abonniert. Nachrichten, die für die Prozessinstanz relevant sind und den entsprechenden Event-Typ haben, werden mit einem *JMS-MessageSelector* auswählt. Die Task speichert das Ereignis beim Eintreffen in einer Prozessvariable unter dem Namen, der bei der Modellierung

[1] `http://finance.yahoo.com`

des Prozesses angegeben wurde. Die zweite *Receive Event Task* wartet dann noch, bis ein Tweet-Ereignis eintrifft, das in die Klasse *buy* eingeordnet wurde. Danach wird die *Buy-Task* ausgeführt.

8.1.3. Andere Varianten bzw. Modellierungsmöglichkeiten

Mit einer zusätzlichen Regel lässt sich der Prozess des vorgestellten Szenarios wie in Abbilding 8.4 weiter vereinfachen.

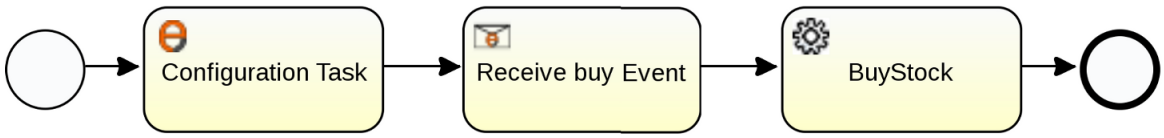


Abbildung 8.4.: Vereinfachter Szenario-Prozess

Die zusätzliche Regel erzeugt ein Ereignis, wenn innerhalb eines bestimmten Zeitraums ein *buy_tweet*-Event auf ein *buy_stock*-Event folgt. Die zusätzliche Regel sieht folgendermaßen aus:

Listing 8.6 Regel: buy

```
INSERT INTO buy SELECT a.value, a.symbol FROM Pattern[EVERY a=buy_stock->b=buy_tweet WHERE
    timer:within(10min)]
```

Eine andere Modellierungsmöglichkeit wäre, die Ereignisse, die von den Ereignisquellen zur Verfügung gestellt werden, mithilfe der Regeln *stock_event* und *tweet* direkt zu empfangen, um sie im Prozess weiter zu verarbeiten. Dazu müssten die Regeln *candidate*, *buy_stock* und *buy_tweet* mit Konzepten der BPMN nachgebildet werden. Ein erster Ansatz dafür könnte z.B. wie in Abbildung 8.5 aussehen. Wie im oben dargestellten Szenario wird hier nicht berücksichtigt, dass die Ereignisse in einem unmittelbaren zeitlichen Zusammenhang stehen müssen.

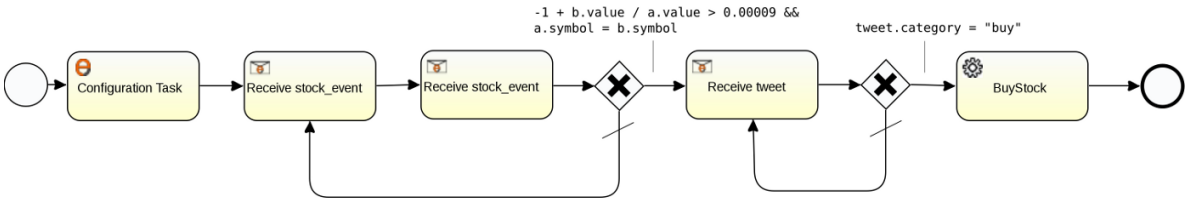


Abbildung 8.5.: Szenario-Prozess mit minimaler Anzahl Regeln

Es zeigt sich also, dass sich nicht nur in der Theorie wie in [MIK08] beschrieben(siehe auch Kapitel 3) die Frage stellt, welche Aspekte mit dem Paradigma Workflow und welche mit dem Paradigma CEP realisiert werden.

9. Zusammenfassung und Ausblick

Abschließend werden in diesem Kapitel die Erkenntnisse zusammengefasst. Es wird ein Fazit gezogen, inwieweit die Ziele der Aufgabenstellung erreicht werden konnten und welche Fragen offen geblieben sind bzw. sich neu ergeben haben.

Zusammenfassung

Nachdem in der *Einleitung* ein Einstieg vermittelt wurde, wurde mit den Kapiteln zu *Grundlagen* und *verwandten Arbeiten* die Basis geschaffen, um *Anforderungen* zu ermitteln und eine *Architektur* zu entwickeln. Danach wurden in Kapitel 7 das Vorgehen und einige Details der *Implementierung* vorgestellt, ehe dann in Kapitel 8 evaluiert wurde, inwieweit Achitektur und Implementierung die Aufgabenstellung und die in Kapitel 4 definierten Anforderungen erfüllen.

Der Fokus dieser Arbeit lag auf der Ausführungsphase. Es wurde eine Architektur geschaffen, die es ermöglicht, beliebige Ereignisse mit Hilfe des Konzepts der EventSource zu empfangen und dann mit einer CEP-Engine oder einem vergleichbaren, regelbasierten System zu verarbeiten und die Ergebnisse schließlich in einem WfMS zu nutzen. Damit geht mein Ansatz über den in [AESW08] hinaus, da es explizit dafür ausgelegt ist, auf bestimmte Ereignisse oder Situationen zu reagieren. Die Art der Reaktion auf Ereignisse, die z.B. von einem CEP-System verarbeitet und erzeugt werden, wird durch einen Prozess festgelegt, der von einem WfMS ausgeführt wird.

Ausblick

Diese Diplomarbeit konnte viele Bereiche nur anreißen, so dass es viele Fragen gibt, die noch näher betrachtet werden sollten. Vergleicht man die entstandene Architektur mit der Referenz-Architektur des WfMC für WfMS, so bemerkt man, dass z.B. kein Konzept für das Monitoring entwickelt wurde. Ebenso hat sich bei der Evaluation in Kapitel 8 gezeigt, dass ein Konzept zum Zugriff auf externe Daten, die nicht in Form von Ereignissen vorliegen, bei der Regelverarbeitung vollständig fehlt. Ein denkbarer Ansatz wird mit Domain-Events in [MLM10], betrachtet in Kapitel 3, genannt. Fast vollständig vernachlässigt wurde in dieser Arbeit die Betrachtung der Modellierung von Regeln und Prozessen. Lediglich in Kapitel 3 wurden einige Ansätze vorgestellt. Ich will versuchen, sie an dieser Stelle einzuordnen.

Die Modellierung von Regeln gestaltet sich in vielen Fällen schwierig, da z.B. bei Esper die EPL sehr mächtig aber auch sehr umfangreich ist. Ein wichtiger Schritt wäre eine einheitliche und standardisierte EPL. Bestehende Engines könnten ggf. weiterhin verwendet und die standardisierte EPL auf die jeweiligen nativen Sprachen abgebildet werden.

In [WMKL09] wird beschrieben, wie sich EPC auf Esper-Regeln abbilden lassen. Ein interessanter Ansatz im Zusammenspiel mit BPMN wäre es, die in [DGB07] vorgestellte Notation BEMN auf Esper abzubilden und somit eine einheitliche Möglichkeit für die Modellierung von Regeln und Prozessen zu entwickeln. Es stellt sich aber die Frage, inwieweit es überhaupt sinnvoll ist, ein einheitliches Modell für Regeln und Prozesse zu haben. Die Komplexität bei der Modellierung von Regeln kann eine Trennung sinnvoll machen. Generell stellt sich bei der Modellierung die Frage, welche Aspekte als Regeln und welche als Prozess modelliert werden. Eine gute Grundlage für die Entscheidung bildet das in Kapitel 3 vorgestellte Entscheidungsframework (siehe 3.2) aus [MIK08].

Im Kapitel über verwandte Arbeiten wurden kurz [WGET08] und [MLM10] erwähnt, die Ansätze aus dem Bereich der künstlichen Intelligenz wählen, um Regeln automatisch zur Laufzeit abzuleiten oder zu verbessern. Dieser Ansatz erfordert es, Regeln zur Laufzeit anpassen zu können.

Im Bereich des Konzepts der EventSources (siehe 6.1.2) gibt es viele offene Fragen. Eine interessante Frage ist z.B., wie EventSources im Hinblick auf Sicherheit, Qualität etc. modelliert werden können. Ein möglicher Ansatz könnte eine Lösung vergleichbar zu WS-Policy sein. Insgesamt fällt auf, dass es gewisse Parallelen zwischen Ereignisquellen und Web Services gibt. Eine Untersuchung, inwieweit es möglich und sinnvoll ist, Web Service Spezifikationen (WS-*) zu erweitern oder zumindest Konzepte aus der Web Service Welt auf EventSources zu übertragen, scheint sinnvoll.

Es ist zu erwarten, dass in Zukunft die Menge der in Echtzeit verfügbaren Daten weiter wächst und immer mehr Sensoren Daten in Form von Ereignissen liefern, die dann wiederum von den unterschiedlichsten Systemen verarbeitet werden, welche neue Ereignisse produzieren. Um einen Überblick über die große Anzahl von Ereignissen in Unternehmen zu bekommen, ist einerseits ein geeignetes Monitoring dieser Daten erforderlich, andererseits wird die Anzahl und Heterogenität von Ereignissen immer weiter zunehmen und durch manuelles Eingreifen von Menschen als Reaktion auf Monitoringergebnisse nicht mehr möglich sein. Daher wird es wichtig sein, aus Ereignissen Aktionen abzuleiten, die mehr und mehr automatisch erfolgen.

WfMSe sind in dem Bereich der Automatisierung von Abläufen weit verbreitet und zusammen mit dem Konzept des CEP in der Lage, Ereignisse zu verarbeiten und abhängig von diesen Ereignissen, Aktionen auszuführen. Diese Diplomarbeit zeigt, dass es technisch machbar ist, beides zu kombinieren. Bevor die Kombination aber zuverlässig und in der Breite eingesetzt werden kann, ist vor allem im Hinblick auf die folgenden Aspekte noch Forschung notwendig:

- Modellierung
- Benutzerfreundlichkeit

Sowohl CEP-Systeme für sich als auch ein System, wie es Gegenstand dieser Diplomarbeit ist, werden nur Erfolg haben, wenn die Systeme komfortabel und möglichst einfach zu bedienen sind.

A. Anhang

A.1. Abkürzungen

In der Informatik und anderen Wissenschaften werden häufig Bezeichnungen verwendet, die aus mehreren Worten bestehen. Aus diesem Grund werden diese Bezeichnungen häufig durch die Verwendung von Akronymen abgekürzt. Für jeden, der sich mit einem Thema oder Themengebiet befasst, wird der Gebrauch schnell selbstverständlich. Für Aussenstehende ist es dagegen häufig schwierig, die Abkürzungen zu verstehen. Aus diesem Grund sind nachfolgend alle verwendeten Abkürzungen aufgelistet.

APF	Adaptable Pervasive Flow
BAM	Business Activity Monitoring
BEMN	Business Event Modelling Language
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modelling Notation oder seit Version 2.0 Business Process Model and Notation
BRM	Business Rules Management
CBE	Common Base Event
CEP	Complex Event Processing
DBMS	Datenbankmanagement-System
EDA	Event Driven Architecture
EDBPM	Event Driven Business Process Management
EPC	Event-driven Process Chain
EPL	Event Pattern Language
EPTS	Event Processing Technical Society
ERP	Enterprise Resource Planning

IAAS	Institut für Architektur von Anwendungssystemen
JAR	Java Archive
JAX-WS	Java API for XML - Web Services
JAXB	Java Architecture for XML Binding
JMS	Java Messaging Service
JPA	Java Persistence API
MoM	Message oriented Middleware
ORM	Object-Relational-Mapper
PVM	Process Virtual Machine
SOA	Service Oriented Architecture
SOEDA	Service Oriented Event Driven Architecture
WfM	Workflow Management
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WSDL	Web Service Description Language

A.2. XML-Schemata und WSDLs

ConfigurationMessage

XML-Schema zur Beschreibung einer *ConfigurationMessage*, die für die Initialisierung einer *ESEngine*-Instanz benötigt wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/ConfigurationMessage"
       xmlns:tns="http://www.informatik.uni-stuttgart.de/eventum/ConfigurationMessage"
       xmlns:es="http://www.informatik.uni-stuttgart.de/eventum/EventSource"
       elementFormDefault="qualified">

       <xsd:import namespace="http://www.informatik.uni-stuttgart.de/eventum/EventSource"
              schemaLocation="./EventSource.xsd" />

       <xsd:element name="cm" type="tns:ConfigurationMessage"></xsd:element>

       <xsd:complexType name="ConfigurationMessage">
                     <xsd:sequence>
                            <xsd:element name="events" type="tns:Events" />
                            <xsd:element name="eventSources" type="tns:EventSources"/>
```

```
                </xsd:sequence>
                <xsd:attribute name="engineType" type="xsd:anyURI" use="required" />
                <xsd:attribute name="correlationID" type="xsd:anySimpleType"
                use="required" />
        </xsd:complexType>

        <xsd:complexType name="EventSources">
                <xsd:sequence>
                        <xsd:element name="eventSource" type="es:EventSource" minOccurs="0"
                            maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="Events">
                <xsd:sequence>
                        <xsd:element name="event" type="xsd:string" minOccurs="1"
                                maxOccurs="unbounded" />
                </xsd:sequence>
        </xsd:complexType>

</xsd:schema>
```

EventSource

XML-Schema zur Beschreibung der Konfiguration einer *EventSource*.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/EventSource"
        xmlns:tns="http://www.informatik.uni-stuttgart.de/eventum/EventSource"
        elementFormDefault="qualified">

        <complexType name="EventSource">
                <sequence>
                        <element name="description" type="string" />
                        <element name="eventSourceConfig" type="string" />
                </sequence>
                <attribute name="id" type="long" use="optional"/>
                <attribute name="name" type="string" use="required"/>
                <attribute name="type" type="string" use="required"/>
        </complexType>


</schema>
```

Rule

XML-Schema zur Beschreibung von Regeln, die vom *ESCStore* verwaltet und von *ESEngine*-Instanzen verarbeitet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/Rule"
       xmlns:r="http://www.informatik.uni-stuttgart.de/eventum/Rule"
           xmlns:es="http://www.informatik.uni-stuttgart.de/eventum/EventSource"
       elementFormDefault="qualified">

       <import schemaLocation="EventSource.xsd"
              namespace="http://www.informatik.uni-stuttgart.de/eventum/EventSource" />

       <complexType name="AbstractRule" abstract="true">
              <sequence>
                     <element name="rule" type="string" />
                     <element name="author" type="string" />
                     <element name="description" type="string" />
                     <element name="esEngine" type="anyURI" />
                     <element name="eventSources" minOccurs="0" maxOccurs="unbounded"
                         type="es:EventSource">
                     </element>
              </sequence>
              <attribute name="id" type="long" use="optional"/>
              <attribute name="name" type="string" use="required"/>
       </complexType>



       <complexType name="RuntimeRule">
              <complexContent>
                     <extension base="r:AbstractRule" />
              </complexContent>
       </complexType>

       <complexType name="InstantiationRule">
              <complexContent>
                     <extension base="r:AbstractRule">
                            <sequence>
                                   <element name="wsdl" type="string" minOccurs="0" />
                                   <element name="operation" type="string" minOccurs="0" />
                                   <element name="processID" type="string" minOccurs="0" />
                            </sequence>
                     </extension>
              </complexContent>
       </complexType>
</schema>
```

ESCStore

WSDL zur Beschreibung des *ESCStore* Web Service.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ESCStore"
   targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/">
<wsdl:types>
  <xsd:schema targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/"
      xmlns:Q1="http://www.informatik.uni-stuttgart.de/eventum/Rule"
      xmlns:Q2="http://www.informatik.uni-stuttgart.de/eventum/ESCResponse"
      xmlns:Q3="http://www.informatik.uni-stuttgart.de/eventum/EventSource">
         <xsd:import schemaLocation="../xml/EventSource.xsd"
             namespace="http://www.informatik.uni-stuttgart.de/eventum/EventSource"></xsd:import>
         <xsd:import schemaLocation="../xml/ESCResponse.xsd"
             namespace="http://www.informatik.uni-stuttgart.de/eventum/ESCResponse"></xsd:import>
         <xsd:import schemaLocation="../xml/Rule.xsd"
             namespace="http://www.informatik.uni-stuttgart.de/eventum/Rule"></xsd:import>
         <xsd:element name="addRule">
             <xsd:complexType>
                    <xsd:sequence>
                           <xsd:element name="Rule" type="Q1:AbstractRule"/>
                    </xsd:sequence>
             </xsd:complexType>
         </xsd:element>
         <xsd:element name="addRuleResponse" type="xsd:long">
         </xsd:element>
         <xsd:element name="getRule" type="xsd:long"></xsd:element>
         <xsd:element name="getRuleResponse" type="Q2:ruleResponse"></xsd:element>
         <xsd:element name="delRule" type="xsd:long"></xsd:element>
         <xsd:element name="getInstantiationRules" type="xsd:long">
         </xsd:element>
         <xsd:element name="getInstantiationRulesResponse"
             type="Q2:ruleResponse"></xsd:element>
         <xsd:element name="getAllRules"
             type="tns:emptyGetAllRulesParam">
         </xsd:element>
         <xsd:element name="getAllRulesResponse" type="Q2:ruleResponse"></xsd:element>

         <xsd:complexType name="emptyGetAllRulesParam"/>
         <xsd:element name="addEventSource" type="Q3:EventSource">
         </xsd:element>
         <xsd:element name="addEventSourceResponse" type="xsd:long"></xsd:element>
         <xsd:element name="getEventSource" type="xsd:long">
         </xsd:element>
         <xsd:element name="getEventSourceResponse"
             type="Q2:eventSourceResponse"></xsd:element>
         <xsd:element name="delEventSource" type="xsd:long">
         </xsd:element>
         <xsd:element name="getRuleEventSource" type="xsd:long">
         </xsd:element>
         <xsd:element name="getRuleEventSourceResponse"
             type="Q2:eventSourceResponse"></xsd:element>
         <xsd:element name="getAllEventSources"
             type="tns:emptyGetAllEventSourcesParam">
         </xsd:element>
         <xsd:element name="getAllEventSourcesResponse"
             type="Q2:eventSourceResponse"></xsd:element>

         <xsd:complexType name="emptyGetAllEventSourcesParam"></xsd:complexType>
```

```
          <xsd:element name="NoSuchRuleFault" type="xsd:string"></xsd:element>
          <xsd:element name="NoSuchEventSourceFault" type="xsd:string"></xsd:element>
          <xsd:complexType name="emptyResponse"/>
          <xsd:element name="getRuleByName" type="xsd:string">
          </xsd:element>
          <xsd:element name="getRuleByNameResponse" type="Q2:ruleResponse"></xsd:element>
          <xsd:element name="getEventSourceByName"
              type="xsd:string">
          </xsd:element>
          <xsd:element name="getEventSourceByNameResponse"
              type="Q2:eventSourceResponse"></xsd:element>
  </xsd:schema>
</wsdl:types>
<wsdl:message name="addRuleRequest">
      <wsdl:part name="parameters" element="tns:addRule"></wsdl:part>
</wsdl:message>
<wsdl:message name="addRuleResponse">
      <wsdl:part name="parameters" element="tns:addRuleResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="getRuleRequest">
      <wsdl:part name="parameters" element="tns:getRule"></wsdl:part>
</wsdl:message>
<wsdl:message name="getRuleResponse">
      <wsdl:part name="parameters" element="tns:getRuleResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="delRuleRequest">
      <wsdl:part name="parameters" element="tns:delRule"></wsdl:part>
</wsdl:message>
<wsdl:message name="delRuleResponse"/>
<wsdl:message name="getInstantiationRulesRequest">
      <wsdl:part name="parameters" element="tns:getInstantiationRules"></wsdl:part>
</wsdl:message>
<wsdl:message name="getInstantiationRulesResponse">
      <wsdl:part name="parameters" element="tns:getInstantiationRulesResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="getAllRulesRequest">
      <wsdl:part name="parameters" element="tns:getAllRules"></wsdl:part>
</wsdl:message>
<wsdl:message name="getAllRulesResponse">
      <wsdl:part name="parameters" element="tns:getAllRulesResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="addEventSourceRequest">
      <wsdl:part name="parameters" element="tns:addEventSource"></wsdl:part>
</wsdl:message>
<wsdl:message name="addEventSourceResponse">
      <wsdl:part name="parameters" element="tns:addEventSourceResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="getEventSourceRequest">
      <wsdl:part name="parameters" element="tns:getEventSource"></wsdl:part>
</wsdl:message>
<wsdl:message name="getEventSourceResponse">
      <wsdl:part name="parameters" element="tns:getEventSourceResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="delEventSourceRequest">
      <wsdl:part name="parameters" element="tns:delEventSource"></wsdl:part>
```

```
</wsdl:message>
<wsdl:message name="delEventSourceResponse"/>
<wsdl:message name="getRuleEventSourcesRequest">
      <wsdl:part name="parameters" element="tns:getRuleEventSource"></wsdl:part>
</wsdl:message>
<wsdl:message name="getRuleEventSourcesResponse">
      <wsdl:part name="parameters" element="tns:getRuleEventSourceResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="getAllEventSourcesRequest">
      <wsdl:part name="parameters" element="tns:getAllEventSources"></wsdl:part>
</wsdl:message>
<wsdl:message name="getAllEventSourcesResponse">
      <wsdl:part name="parameters" element="tns:getAllEventSourcesResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="delRuleFault">
      <wsdl:part name="fault" element="tns:NoSuchRuleFault"></wsdl:part>
</wsdl:message>
<wsdl:message name="delEventSourceFault">
      <wsdl:part name="fault" element="tns:NoSuchEventSourceFault"></wsdl:part>
</wsdl:message>
<wsdl:message name="getRuleByNameRequest">
      <wsdl:part name="parameters" element="tns:getRuleByName"></wsdl:part>
</wsdl:message>
<wsdl:message name="getRuleByNameResponse">
      <wsdl:part name="parameters" element="tns:getRuleByNameResponse"></wsdl:part>
</wsdl:message>
<wsdl:message name="getEventSourceByNameRequest">
      <wsdl:part name="parameters" element="tns:getEventSourceByName"></wsdl:part>
</wsdl:message>
<wsdl:message name="getEventSourceByNameResponse">
      <wsdl:part name="parameters" element="tns:getEventSourceByNameResponse"></wsdl:part>
</wsdl:message>
<wsdl:portType name="ESCStore">
      <wsdl:operation name="addRule">
            <wsdl:input message="tns:addRuleRequest"></wsdl:input>
            <wsdl:output message="tns:addRuleResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getRule">
            <wsdl:input message="tns:getRuleRequest"></wsdl:input>
            <wsdl:output message="tns:getRuleResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="delRule">
            <wsdl:input message="tns:delRuleRequest"></wsdl:input>
         <wsdl:output message="tns:delRuleResponse"/>
         <wsdl:fault name="fault" message="tns:delRuleFault"></wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="getInstantiationRules">
            <wsdl:input message="tns:getInstantiationRulesRequest"></wsdl:input>
            <wsdl:output message="tns:getInstantiationRulesResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getAllRules">
            <wsdl:input message="tns:getAllRulesRequest"></wsdl:input>
            <wsdl:output message="tns:getAllRulesResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="addEventSource">
```

```
            <wsdl:input message="tns:addEventSourceRequest"></wsdl:input>
            <wsdl:output message="tns:addEventSourceResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getEventSource">
            <wsdl:input message="tns:getEventSourceRequest"></wsdl:input>
            <wsdl:output message="tns:getEventSourceResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="delEventSource">
            <wsdl:input message="tns:delEventSourceRequest"></wsdl:input>
            <wsdl:output message="tns:delEventSourceResponse"></wsdl:output>
         <wsdl:fault name="fault" message="tns:delEventSourceFault"></wsdl:fault>
      </wsdl:operation>
      <wsdl:operation name="getRuleEventSources">
            <wsdl:input message="tns:getRuleEventSourcesRequest"></wsdl:input>
            <wsdl:output message="tns:getRuleEventSourcesResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getAllEventSources">
            <wsdl:input message="tns:getAllEventSourcesRequest"></wsdl:input>
            <wsdl:output message="tns:getAllEventSourcesResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getRuleByName">
            <wsdl:input message="tns:getRuleByNameRequest"></wsdl:input>
            <wsdl:output message="tns:getRuleByNameResponse"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getEventSourceByName">
            <wsdl:input message="tns:getEventSourceByNameRequest"></wsdl:input>
            <wsdl:output message="tns:getEventSourceByNameResponse"></wsdl:output>
      </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ESCStoreSOAP" type="tns:ESCStore">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="addRule">
            <soap:operation
                  soapAction="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/addRule"
                      />
            <wsdl:input>
                  <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                  <soap:body use="literal" />
            </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getRule">
            <soap:operation
                  soapAction="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/getRule"
                      />
            <wsdl:input>
                  <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                  <soap:body use="literal" />
            </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="delRule">
```

```
        <soap:operation
                soapAction="http://www.informatik.uni-stuttgart.de/eventum/ESCStore/delRule"
                    />
        <wsdl:input>
                <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
        <wsdl:fault name="fault">
                <soap:fault use="literal" name="fault" />
        </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getInstantiationRules">
        <soap:operation
                soapAction="http://.../eventum/ESCStore/getInstantiationRules" />
        <wsdl:input>
                <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getAllRules">
        <soap:operation
                soapAction="http://.../eventum/ESCStore/getAllRules" />
        <wsdl:input>
                <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="addEventSource">
        <soap:operation
                soapAction="http://.../eventum/ESCStore/addEventSource" />
        <wsdl:input>
                <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getEventSource">
        <soap:operation
                soapAction="http://.../eventum/ESCStore/getEventSource" />
        <wsdl:input>
                <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
                <soap:body use="literal" />
        </wsdl:output>
</wsdl:operation>
<wsdl:operation name="delEventSource">
        <soap:operation
```

```
                soapAction="http://.../eventum/ESCStore/delEventSource" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
            <wsdl:fault name="fault">
                <soap:fault use="literal" name="fault" />
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getRuleEventSources">
            <soap:operation
                soapAction="http://.../eventum/ESCStore/getRuleEventSources" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getAllEventSources">
            <soap:operation
                soapAction="http://.../eventum/ESCStore/getAllEventSources" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getRuleByName">
            <soap:operation
                soapAction="http://.../eventum/ESCStore/getRuleByName" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getEventSourceByName">
            <soap:operation
                soapAction="http://.../eventum/ESCStore/getEventSourceByName" />
            <wsdl:input>
                <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ESCStore">
      <wsdl:port binding="tns:ESCStoreSOAP" name="ESCStoreSOAP">
        <soap:address location="http://localhost:8080/ESCStore"/>
```

```
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

ESEngine

WSDL zur Beschreibung des ESEngine Web Service.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:tns="http://www.informatik.uni-stuttgart.de/eventum/ESEngine/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       name="ESEngine"
           targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/ESEngine/"
       xmlns:cm="http://www.informatik.uni-stuttgart.de/eventum/ConfigurationMessage">
       <wsdl:types>
              <xsd:schema
                  targetNamespace="http://www.informatik.uni-stuttgart.de/eventum/ESEngine/">
                     <xsd:import schemaLocation="../xml/ConfigurationMessage.xsd"
                            namespace="http://www.informatik.uni-stuttgart.de/eventum/ConfigurationMessage"
                                />
                     <xsd:element name="createESEngineInstance">
                            <xsd:complexType>
                                   <xsd:sequence>
                                          <xsd:element name="in"
                                              type="cm:ConfigurationMessage" />
                                   </xsd:sequence>
                            </xsd:complexType>
                     </xsd:element>
                     <xsd:element name="createESEngineInstanceResponse">
                            <xsd:complexType>
                                   <xsd:sequence>
                                          <xsd:element name="out" type="xsd:string" />
                                   </xsd:sequence>
                            </xsd:complexType>
                     </xsd:element>
                     <xsd:element name="destroyESEngineInstance" type="xsd:string" />

                     <xsd:complexType name="destroyESEngineInstanceResponse"/>

                     <xsd:element name="getEngineType">
                            <xsd:complexType/>
                     </xsd:element>


                     <xsd:element name="getEngineTypeResponse" type="xsd:string" />

              </xsd:schema>
       </wsdl:types>
       <wsdl:message name="createESEngineInstanceRequest">
              <wsdl:part element="tns:createESEngineInstance" name="parameters" />
       </wsdl:message>
```

```
<wsdl:message name="createESEngineInstanceResponse">
       <wsdl:part element="tns:createESEngineInstanceResponse"
              name="parameters" />
</wsdl:message>
<wsdl:message name="destroyESEngineInstanceRequest">
       <wsdl:part name="parameters" element="tns:destroyESEngineInstance" />
</wsdl:message>
<wsdl:message name="destroyESEngineInstanceResponse"/>

<wsdl:message name="getEngineTypeRequest">
       <wsdl:part name="parameters" element="tns:getEngineType" />
</wsdl:message>
<wsdl:message name="getEngineTypeResponse">
       <wsdl:part name="parameters" element="tns:getEngineTypeResponse" />
</wsdl:message>

<wsdl:portType name="ESEngine">
       <wsdl:operation name="createESEngineInstance">
              <wsdl:input message="tns:createESEngineInstanceRequest" />
              <wsdl:output message="tns:createESEngineInstanceResponse" />
       </wsdl:operation>
       <wsdl:operation name="destroyESEngineInstance">
              <wsdl:input message="tns:destroyESEngineInstanceRequest" />
              <wsdl:output message="tns:destroyESEngineInstanceResponse" />
       </wsdl:operation>
       <wsdl:operation name="getEngineType">
              <wsdl:input message="tns:getEngineTypeRequest" />
              <wsdl:output message="tns:getEngineTypeResponse" />
       </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ESEngineSOAP" type="tns:ESEngine">
       <soap:binding style="document"
              transport="http://schemas.xmlsoap.org/soap/http" />
       <wsdl:operation name="createESEngineInstance">
              <soap:operation
                     soapAction="http://.../eventum/ESEngine/createESEngineInstance"
                         />
              <wsdl:input>
                     <soap:body use="literal" />
              </wsdl:input>
              <wsdl:output>
                     <soap:body use="literal" />
              </wsdl:output>
       </wsdl:operation>
       <wsdl:operation name="destroyESEngineInstance">
              <soap:operation
                     soapAction="http://.../eventum/ESEngine/destroyESEngineInstance"
                         />
              <wsdl:input>
                     <soap:body use="literal" />
              </wsdl:input>
              <wsdl:output>
                     <soap:body use="literal" />
              </wsdl:output>
       </wsdl:operation>
```

```
            <wsdl:operation name="getEngineType">
                  <soap:operation
                        soapAction="http://.../eventum/ESEngine/getEngineType" />
                  <wsdl:input>
                        <soap:body use="literal" />
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="ESEngine">
            <wsdl:port binding="tns:ESEngineSOAP" name="ESEngineSOAP">
                  <soap:address location="http://localhost:8080/ESEngine" />
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

Literaturverzeichnis

[AESW08] R. von Ammon, C. Emmersberger, F. Springer, C. Wolff. Event-Driven Business Process Management and its Practical Application Taking the Example of DHL. *Complex Event Processing blog*, 2008. (Zitiert auf den Seiten 23, 29 und 69)

[BDG07] A. Barros, G. Decker, A. Grosskopf. Complex events in business processes. In *Business Information Systems*, pp. 29–40. Springer, 2007. (Zitiert auf den Seiten 23, 24, 32, 33, 60 und 61)

[BDK10] F. Burger, P. Debicki, F. Kötter. Vergleich von Complex Event Processing-Ansätzen für Business Activity Monitoring. 2010. (Zitiert auf den Seiten 19 und 38)

[BE07] F. Bry, M. Eckert. Rule-based composite event queries: The language XChange eq and its semantics. *Web Reasoning and Rule Systems*, pp. 16–30, 2007. (Zitiert auf Seite 17)

[CLS+05] F. Curbera, F. Leymann, T. Storey, D. Ferguson, S. Weerawarana. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, 2005. (Zitiert auf Seite 19)

[Dey01] A. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001. (Zitiert auf Seite 20)

[DGB07] G. Decker, A. Grosskopf, A. Barros. A graphical notation for modeling complex events in business processes. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, p. 27. IEEE, 2007. (Zitiert auf den Seiten 5, 21, 22 und 70)

[DKGZ10] M. Döhring, L. Karg, E. Godehardt, B. Zimmermann. The Convergence of Workflows, Business Rules and Complex Events—Defining a Reference Architecture and Approaching Realization Challenges. In *ICEIS '10*. 2010. (Zitiert auf den Seiten 22, 23 und 29)

[FRH10] J. Freund, B. Rücker, T. Henninger. *Praxishandbuch BPMN*. HANSER, 2010. (Zitiert auf den Seiten 5, 12 und 13)

[HCC05] J. Han, Y. Cho, J. Choi. Context-aware workflow language based on web services for ubiquitous computing. *Computational Science and Its Applications–ICCSA 2005*, pp. 1008–1017, 2005. (Zitiert auf Seite 25)

[HCKC06] J. Han, Y. Cho, E. Kim, J. Choi. A ubiquitous workflow service framework. *Computational Science and its Applications-ICCSA 2006*, pp. 30–39, 2006. (Zitiert auf Seite 25)

[LK06] B. List, B. Korherr. An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1532–1539. ACM, 2006. (Zitiert auf Seite 12)

[LL06] J. Ludewig, H. Lichter. Software Engineering-Grundlagen. *Menschen, Prozesse, Techniken. dpunkt. verlag*, 2006. (Zitiert auf Seite 29)

[LR00] F. Leymann, D. Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, 2000. (Zitiert auf Seite 11)

[LS08] D. Luckham, R. Schulte. Event processing glossary-version 1.1. *Event Processing Technical Society, Tech. Rep*, 2008. (Zitiert auf den Seiten 19 und 20)

[Luc02] D. Luckham. *The Power of Events*. Addison-Wesley, 2002. (Zitiert auf den Seiten 7 und 15)

[Men11] F. Menge. Workshop: Activiti BPM Platform. Web, 2011. (Zitiert auf den Seiten 5 und 37)

[MIK08] M. zur Muehlen, M. Indulska, K. Kittel. Towards integrated modeling of business processes and business rules. In *19th Australian Conference on Information Systems ACIS*. 2008. (Zitiert auf den Seiten 5, 20, 22, 67 und 70)

[MLM10] J. Ma, W. Liu, P. Miller. Event Modelling and Reasoning with Uncertain Information for Distributed Sensor Networks. *Scalable Uncertainty Management*, pp. 236–249, 2010. (Zitiert auf den Seiten 26, 69 und 70)

[OMG10] OMG. Business Process Model and Notation (BPMN), Version 2.0. 2010. (Zitiert auf Seite 12)

[SCCY07] K. Shin, Y. Cho, J. Choi, C. Yoo. A workflow Language for Context-Aware Services. 2007. (Zitiert auf Seite 25)

[SN03] G. Steinke, C. Nickolette. Business rules as the basis of an organization's information systems. *Industrial Management & Data Systems*, 103(1):52–63, 2003. (Zitiert auf Seite 20)

[WfM99] G. WfMC. Terminology and Glossary. *Document No WFMC-TC-1011. Workflow Management Coalition. Winchester*, 1999. (Zitiert auf Seite 11)

[WGET08] S. Wasserkrug, A. Gal, O. Etzion, Y. Turchin. Complex event processing over uncertain data. In *Proceedings of the second international conference on Distributed event-based systems*, pp. 253–264. ACM, 2008. (Zitiert auf den Seiten 26 und 70)

[WHR09] H. Wolf, K. Herrmann, K. Rothermel. Modeling Dynamic Context Awareness for Situated Workflows. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, pp. 98–107. Springer, 2009. (Zitiert auf den Seiten 25 und 30)

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, F. Leymann. Towards context-aware workflows. In *CAiSE07 Proc. of the Workshops and Doctoral Consortium*, volume 2. Citeseer, 2007. (Zitiert auf Seite 25)

[WMKL09] M. Wieland, D. Martin, O. Kopp, F. Leymann. SOEDA: A Methodology for Specification and Implementation of Applications. 2009. (Zitiert auf den Seiten 8, 23, 29 und 70)

Alle URLs wurden zuletzt am 28.06.2011 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Sascha Julien Retter)