

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Bachelorarbeit

**Entwurf eines Frameworks für komplexe
Events auf verteilten, reaktiven
Datenbanken und Triple-Stores**

Markus Freudenberg

28. April 2014

Advisors: Dipl.-Inf. Sebastian Tramp

Inhaltsverzeichnis

1. Einführung	6
1.1. Motivation und Ziele	6
2. Grundlagen	8
2.1. Resource Description Framework (RDF)	8
2.2. Events	9
2.3. Complex Event Processing (CEP)	9
2.4. ECA-Regeln	11
2.5. Reaktivität	11
2.6. SOAP	12
3. Anforderungen	13
3.1. Anwendungsfälle (AF)	13
3.1.1. AF1: Cloud-Events	13
3.1.2. AF2: Automatisiertes Dokumentenverwaltungssystem	13
3.1.3. AF3: Produktionsablaufüberwachung	13
3.1.4. AF4: Benutzerkontenabsicherung eines Onlineshops	14
3.1.5. AF5: Flexible Teilautovermietung	14
3.1.6. AF6: Automatische Aufgaben in sozialen Netzwerken	14
3.1.7. AF7: Zentrales Event-System einer Hochschule	14
3.1.8. AF8: Überwachung von Anlagen erneuerbaren Energie	15
3.1.9. AF9: Automatisierte Terminplanung	15
3.1.10. AF10: Lokale Auswertung von Wahlkampfstrategien	15
3.2. Benutzeranforderungen	15
3.2.1. BA1: Verteiltheit	15
3.2.2. BA2: Systemunabhängigkeit	15
3.2.3. BA3: Eigenständigkeit	16
3.2.4. BA4: Zuverlässigkeit bei großen Zugriffszahlen	16
3.2.5. BA5: Verlässlichkeit bei der Erkennung von Events	16
3.2.6. BA6: Echtzeit	16
3.2.7. BA7: physische Datenunabhängigkeit	16
3.2.8. BA8: Datenschutz und Sicherheit	16
3.2.9. BA9: freie Wahl und Definition von Events und Ressourcen	17
3.2.10. BA10: Verwaltung von Benutzern und Eventressourcen	17
3.2.11. BA11: RDF-Events	17
3.3. Auswertung	18
4. Bestandsaufnahme	19
4.1. Universale Complex Event Processing Systeme	19
4.1.1. Event-Stream-Processing-System: Odysseus	19
4.2. RDF-basierte CEP-Systeme	20
4.2.1. RDFTL	20
4.2.2. REWERSE I5	21
5. Spezifikation	24
5.1. Terminologie	24
5.2. Aufbau und Architektur	28

5.3.	Data-Layer	29
5.3.1.	Event Ontology	29
5.3.2.	Definition eines komplexen Events mit Hilfe von regulären Ausdrücken	33
5.3.3.	Definition eines komplexen Events am Beispiel	34
5.4.	Service-Layer	35
5.4.1.	Event-Service	35
5.4.2.	Client-Service	36
5.5.	Application-Layer	37
5.5.1.	Event-Framework-Control	37
6.	Implementierung	38
6.1.	Beispielszenario	39
6.1.1.	Lösungsansatz mit manuell definierten Datenbanktriggern	39
6.1.2.	Lösungsansatz auf Basis eines Triple-Stores	40
6.1.3.	Lösungsansatz mit dem Event-Framework	40
6.2.	Satellitendatenbanken und atomare Events	41
6.2.1.	Definition atomarer Events	41
6.2.2.	Auslösen atomarer Events	42
6.2.3.	Evaluation und Signalisierung eines atomaren Events	43
6.3.	Zentraldatenbank und komplexe Events	44
6.3.1.	Aktivierung komplexer Events	44
6.3.2.	Empfangen von Events	45
6.3.3.	Evaluation komplexer Events	46
6.3.4.	Initiierung von Conditions und Actions	48
6.4.	Event-Service	50
6.4.1.	Ausführung von Actions und Conditions	51
6.4.2.	Verbindungen mit Datenbanken	54
6.4.3.	Schnittstelle für den Client-Service	55
6.5.	Client-Service	56
6.6.	Event-Framework-Control	57
6.6.1.	Benutzer- und Rechteverwaltung	57
6.6.2.	Verwaltung von Satellitendatenbanken, Event- und Datenquellen . .	58
6.6.3.	Definition von Events auf Satellitendatenbanken und externen Datenquellen	58
6.6.4.	Erstellung/Registrierung von Actions, Conditions und Queries . . .	59
6.6.5.	Definition von komplexen Events	60
6.6.6.	Definition von EventSets aus Events, Actions, Conditions und Queries	61
6.7.	Systemanforderungen	61
7.	Evaluation und Diskussion	62
7.1.	Umsetzung von Benutzeranforderungen	62
7.1.1.	BA1: Verteiltheit	62
7.1.2.	BA2: Systemunabhängigkeit	62
7.1.3.	BA3: Eigenständigkeit	63
7.1.4.	BA4: Zuverlässigkeit bei großen Zugriffszahlen	63
7.1.5.	BA5: Verlässlichkeit bei der Erkennung von Events	63
7.1.6.	BA6: Echtzeit	63
7.1.7.	BA7: physische Datenunabhängigkeit	63
7.1.8.	BA8: Datenschutz und Sicherheit	64

7.1.9.	BA9: freie Wahl und Definition von Events und Ressourcen	64
7.1.10.	BA10: Verwaltung von Benutzern und Eventressourcen	64
7.1.11.	BA11: RDF-Events	64
7.2.	Lösungen für Anwendungsfälle	64
7.2.1.	AF1: Cloud-Events	64
7.2.2.	AF2: Automatisiertes Dokumentenverwaltungssystem	65
7.2.3.	AF3: Produktionsablaufüberwachung	65
7.2.4.	AF4: Benutzerkontenabsicherung eines Onlineshops	65
7.2.5.	AF5: Flexible Teilautovermietung	65
7.2.6.	AF6: Automatische Aufgaben in sozialen Netzwerken	65
7.2.7.	AF7: Zentrales Event-System einer Hochschule	66
7.2.8.	AF8: Überwachung von Anlagen erneuerbaren Energie	66
7.2.9.	AF9: Automatisierte Terminplanung	66
7.2.10.	AF10: Lokale Auswertung von Wahlkampfstrategien	66
7.3.	Beispiel: Einsatz eines Raspberry Pi als Eventquelle	67
7.4.	Bekannte Probleme und nächste Schritte	69
7.5.	Ausblick	70
Literatur		i
Abbildungsverzeichnis		iv
Tabellenverzeichnis		v
Appendix		vi
A. Tabellen Zentraldatenbank		vi
B. Tabellen Satellitendatenbank		x
C. Event-Data-Objects		xi
D. Interfaces		xii
Selbstständigkeitserklärung		xvii

Zusammenfassung

Diese Arbeit stellt ein Framework für komplexe Events auf Grundlage von Event-Condition-Action-Rules für verteilte Datenquellen vor. Dabei soll die Definition von atomaren und komplexen Events, dezentral über eine administrative Applikation, möglich sein. Zur Ausführung von Aktionen jeglicher Art, als Reaktion auf ein eingetretenes Ereignis, kann auf die Funktionalität von SOAP-Endpunkten zurückgegriffen werden. Durch die Präsentation einer einfachen, aktiven Hülle für passive Triple-Stores einer Virtuoso-Datenbank soll eine Möglichkeit zur Schaffung reaktiver Triple-Stores demonstriert werden.

1. Einführung

Mir kommt es immer vor, dass die Art- wie man die Ereignisse des Lebens nimmt - ebenso wichtigen Anteil an unserem Glück und Unglück hätte als diese Ereignisse selbst.

Wilhelm von Humboldt

Ereignisbasiertes Verhalten ist ein allgegenwärtiger Bestandteil unserer Umwelt und des menschlichen Verhaltens.

1. Eine Antilope ergreift die Flucht, da sie Bewegungen am Rand ihres Sichtfeldes wahrgenommen hat.
2. Eine Ampel schaltet auf Rot. Ein Autofahrer muss daraufhin bremsen.

Diese Beispiele sind komplizierter als sie auf den ersten Blick erscheinen. Die Antilope ergreift erst dann die Flucht, wenn ihr Gehirn das Gefahrenpotential der unbekannteren Bewegung als genügend hoch eingestuft hat. Nicht jeder Autofahrer muss beim Anblick einer roten Ampel bremsen. In den meisten Fällen tritt eine Reaktion erst dann ein, wenn bestimmte Ereignisse zusammenfallen oder diese unter bestimmten Bedingungen eintreten. Da die richtige Reaktion auf die Abfolge von bestimmten Ereignissen oft über Tod und Leben entscheidet, ist das menschliche Gehirn spezialisiert auf die Erkennung von Mustern in einer Flut von Ereignissen (vgl. [1] Kapitel 1.4. - Mustererkennung).

Eine Simulation dieses Verhaltens ist zentraler Bestandteil jedes Computersystems. Die meisten Prozesse, die durch einen Computer ausgeführt werden, können als Reaktionen auf zuvor eingetretene Ereignisse verstanden werden. Um auf komplexe Ereignismuster reagieren zu können, müssen diese in einem Strom vieler Ereignisse erkannt werden. Dafür werden in der Regel verschiedene Arten von Complex-Event-Processing (CEP) Verfahren eingesetzt.

1.1. Motivation und Ziele

Der Einsatz von Complex-Event-Processing-Systemen für viele verteilte Datenquellen ist ein wichtiges Ziel dieser Arbeit. Verschiedene Ansätze (vgl. Kapitel 4.) legen eine Abstraktion der Eventquellen nahe und konzentrieren sich auf das Erkennen und Verarbeiten komplexer Events. Andere Ansätze basieren auf bestimmten Eventquellen als Datenbasis von Ereignissen. Ein neuer Entwurf, der Events unabhängig von bestimmten Datenquellen erkennen kann und eine detaillierte Definition von Events ermöglicht, ist erklärtes Ziel dieser Arbeit. Die Definition von einfachen, datengebundenen Ereignissen auf herkömmlichen tabellengebundenen, wie RDF-basierten Datenstrukturen, bis hin zur ausführlichen Definition komplexer Ereignisse auf Grundlage verteilter Datenquellen, soll in einer nachvollziehbaren Weise umgesetzt werden.

Durch die wachsende Bedeutung des Semantic Webs [2] und die damit verbundene Ausweitung von Metadaten im RDF-Format (vgl. Kapitel 2.1.), müssen datengebundene Ereignisse nicht mehr nur in klassischen Datenbanken aufgezeichnet werden, sondern auch in Triple-Stores (eine Datenbank für RDF-Triple). Hierzu bedarf es neuer Konzepte, die Events auf diesen Datenquellen aufspüren, da nur wenige existierenden Triple-Stores eine derartige Funktionalität bieten (siehe Kapitel 2.5.).

Ausgangspunkt für dieses Projekt war es, eine reaktive Hülle für einen Virtuoso-Triple-Store (Teil des Virtuoso Universal Servers [3]) zu erstellen, welche die Reaktivität der SQL-Datenbank von Virtuoso (vgl. Kapitel 2.5.) auf den integrierten Triple-Store erweitert. Im Laufe der Entwicklung wurde daraus ein allgemeines Framework für komplexe Events, welches an keine bestimmte Datenbank als Datenquelle gebunden ist. Da Virtuoso als einziges Datenbanksystem in dieser Arbeit eingesetzt wird, ist der Name Virtuoso-Event-Framework weiterhin gerechtfertigt, wird aber der Einfachheit halber mit der gekürzten Form 'Event-Framework' bezeichnet.

Ein weiteres wichtiges Ziel ist, die einfache Anbindung von Anwendungen auf Client-Computern, welche die Ergebnisse des Frameworks nutzen. Dazu soll die Architektur des Event-Frameworks bis auf die Anwendungsebene erweitert werden, um dort einen einfachen Zugriff auf Daten und Funktionen des Frameworks zu gewährleisten.

Zur Realisierung eines solchen Frameworks für die Definition, Erkennung und Verarbeitung von komplexen Events, werden Anwendungsfälle und daraus resultierende Anforderungen zusammengetragen. Auf Basis der Anforderungen wird ein Entwurf der Software erarbeitet, um sie gemäß der Spezifikation umzusetzen. Zum Abschluss wird eine Evaluation der Fähigkeiten und Grenzen des entstandenen Entwurfes durchgeführt.

Der Source-Code und die Dokumentation für das Virtuoso-Event-Framework ist unter: <https://github.com/AKSW/VirtuosoEventFramework/> zu finden.

2. Grundlagen

Die in dieser Arbeit entwickelte Software greift auf Technologien des Sematic Web und reaktiver Datenbanken zurück. Grundlegende Konzepte und Technologien für das Verständnis der Problematik sollen in diesem Kapitel näher beschrieben werden.

2.1. Resource Description Framework (RDF)

Das Resource Description Framework ist grundlegender Bestandteil des Sematic Webs und wird zur Beschreibung von Ressourcen verwendet. Eine Ressource kann ein Dokument, eine Webseite, ein Bild oder jedes andere denkbare Ding sein, welches sich mit einem Uniform Resource Identifier (URI; [4]) eindeutig identifizieren lässt (Beispiel: `http://www.w3.org/RDF` - Ressource 'RDF' als eindeutige URI). Eine Ressource wird durch Eigenschaften (engl. properties) beschreiben. Eigenschaften einer Ressource drücken Merkmale und Besonderheiten aus oder zeigen Beziehungen zu anderen Ressourcen auf. Diese Eigenschaften werden durch Aussagen (engl. statements) formuliert, welche in Form von 3-Tupeln (engl. Triple) dargestellt werden. Diese Tripel besteht aus: URI einer Ressource (Subjekt), URI einer Eigenschaft (Prädikat) und dem Wert dieser Eigenschaft (Objekt), wobei das Objekt entweder eine Ressource mit URI oder ein Literal (etwa Zeichenfolge, vgl. [5]) sein kann. Dieses einfache linguistische Konstrukt machen Daten in Tripleform für Menschen und Computer leicht verständlich. Jede Ressource lässt sich so mit einer endlichen Menge von Tripeln beschreiben, welche zusammen einen Graphen bilden. Graphen können ebenfalls eine URI tragen, welche als zusätzliche Information einem Triple vorangestellt wird und so einen 4-Tupel oder Quad bilden.

Listing 1: Beispiel: ein 4-Tupel (Quad) als RDF-Datenformat

```
graph          subject          predicate          object
<http://www.w3.org> <http://www.w3.org/RDF> <http://SchemaX#creationDate> "10.04.2005"
die beschriebene Ressource 'RDF' (im Graph http://www.w3.org) wurde am 10.4.2005 erstellt
```

Die so dargestellten Daten können in Triple-Stores (auch Quad-Stores) gespeichert werden. Diese bauen entweder auf herkömmlichen Datenbanksystemen (in Tabellenform) oder auf graph-basierten Speicherkonzepten beruhen (z.B. AllegroGraph, Neo4j [6]) auf.

Zur Wiederverwendung von Ressourcen und Eigenschaften wird auf gemeinsame Schemas (oder Ontologie, Vokabular) zurückgegriffen. Ontologien beinhalten eine Beschreibung der Semantik von Ressourcen und Eigenschaften. Ressourcen und Eigenschaften einer Ontologie sind meist in einem gemeinsamen Namensraum definiert. Der gleich bleibende Teil einer URI kann mit Hilfe eines Präfixes abgekürzt werden (z. B. PREFIX w3: <http://www.w3.org/>).

Das RDF-Schema (RDFS) dient als Grundlage für die meisten Schemas. Mit Hilfe von RDFS kann man grundlegende semantische Beziehungen zwischen Ressourcen darstellen (z. B. Zugehörigkeit von Ressourcen zu Konzepten, Werte- und Definitionsbereich von Prädikaten usw.). Verschiedene Schemas können so als Schichten übereinander gelegt werden und zusammen eine genaue Definition der zu verwendenden Semantik in einem Graphen liefern. So basiert das in dieser Arbeit beschriebene Schema 'Event Ontology' (vgl. Kapitel 5.3.1.) auf OWL2, ein Schema dass auf OWL basiert, das wiederum auf RDFS aufbaut. In Anlehnung

an die Abfragesprache SQL für Datenbanken, wird zur Abfrage von Daten im RDF-Format auf die ähnlich strukturierte Abfragesprache SPARQL zurückgegriffen.

2.2. Events

Neben RDF als Datenformat möglicher Datenquellen, von denen bestimmte Ereignisse ausgehen können, muss der Begriff 'Ereignis' (im Kontext dieser Arbeit in der Regel als Event bezeichnet) genau beschrieben werden. Beim Umgang mit Events muss stets klar zwischen Eventdefinition und dem tatsächliche Auftreten eines zuvor durch eine Definition beschriebenen Events (Eventinstanz) unterschieden werden. Während die Eventdefinition alle möglichen Umstände beschreibt, unter denen ein Event eintreten kann, ist das tatsächliche Eintreten (oder Instanziierung) eines solchen Events als Signal zu verstehen, das über diesen Umstand informiert. Anzumerken ist, dass eine Eventinstanz durch eine Vielzahl von Eventdefinitionen beschrieben werden kann. Andererseits beschreibt die Definition eines Events selten alle Aspekte eines eingetretenen Events.

Events lassen sich weiterhin einteilen in atomare und komplexe Events. Erstere beschreiben Ereignisse, welche im Bezug auf ein betrachtetes System (z. B. eine Datenbank) nicht weiter in sinnvolle Teilereignisse zerlegbar sind. So könnte man das Einfügen eines neuen Tupels in eine Datenbanktabelle weiter unterteilen in folgende Schritte:

1. Erhalt eines SQL-Befehls
2. Auswertung dieses Befehls
3. Speicherallokation
4. Speicherbelegung mit neuen Werten

Diese Teilereignisse spielen aber im Allgemeinen für Anwendungen oder Nutzer einer Datenbank keine Rolle. Unter diesem Gesichtspunkt ist das Einfügen eines Tupels in eine Tabelle ein atomares Event.

Komplexe Events setzen sich aus mindestens einem atomaren oder komplexen Event zusammen, das durch passende Operatoren mit anderen Events verknüpft werden [7] (z. B. Boole'sche Operatoren) oder mit zusätzlichen Bedingungen versehen werden (z. B. Einschränkung der Tageszeit des Auftretens).

2.3. Complex Event Processing (CEP)

Events werden in der Computertechnik bereits seit über 50 Jahren verwendet. Am Anfang beschrieb man damit allein interne Events, wie den Wechsel zwischen zwei Threads. Mit der Zeit wurden aber auch externe Ereignisse, z. B. Events auf Netzwerkebene oder andere Stimuli in der Informatik unter dem Begriff Event zusammengefasst. Event Processing, also die Verarbeitung von Ereignissen, spielt in vier verschiedenen Bereichen eine wichtige Rolle [8]:

1. ereignisorientierte Simulation
2. Netzwerktechnik
3. Reaktive Datenbanken
4. Middleware

Ursprünglich stammt die Idee des Event Processing aus der Simulation. Die Idee war, ein wie auch immer geartetes System zu simulieren, in dem auf Eingabedaten mit so genannten Events reagiert wird, die die Interaktionen des Systems darstellen. Nach und nach nahmen andere Bereiche der Informatik diese Idee auf und entwickelten sie in verschiedene Richtungen weiter. Die vorliegende Arbeit greift im Besonderen auf die Ergebnisse aus den Bereichen Middleware und reaktiven Datenbanken zurück.

Einzelne, atomare Events sind oft zu primitiv, um aus ihnen eine Folgehandlung abzuleiten, da meist wichtige Informationen fehlen. Daher entwickelte sich die Thematik der Eventverarbeitung stetig weiter, um komplexe Events zu erkennen und auf diese automatisch reagieren zu können. Die gezielte Entwicklung des Complex Event Processing, zum Einsatz auf dem Markt, begann um das Jahr 2000.

In der IT-Branche entstand der Wunsch, event-basierte und global arbeitende Netzwerke zu realisieren. In einem solchen Netzwerk sollten Events jeden Levels weltweit und in Echtzeit überwacht und ausgewertet werden können. Dabei geht es natürlich nicht nur darum, das Eintreten einzelner Events zu erkennen, es sollen auch Muster und Zusammenhänge erkannt werden, die aus vielen verschiedenen Events bestehen, die zeitlich und örtlich weit auseinander liegen können. Natürlich soll auf das Erkennen eines solchen Musters auch entsprechend reagiert werden können. In einem solchen System besteht ebenfalls die Notwendigkeit, die Suchmuster und Suchstrategien on-the-fly und in Echtzeit zu ändern. Auch diese Fähigkeit sollte ein entsprechendes System haben. [9]

CEP-Systeme lassen sich grob in zwei Sparten einteilen:

1. Event-Stream-Processing-Systeme: Diese CEP-Systeme (kurz: ESP-Systeme) werten fortwährend einen Datenstrom (Eventstrom) mit Hilfe SQL-ähnlicher Abfragesprachen aus, um Muster komplexer Events darin zu erkennen.
2. Regelbasierte Systeme: Diese Systeme werten komplexe Events in mehreren Stufen aus, wobei vordefinierte Regeln das Muster der erwarteten Ereignisketten bestimmen.

Auch regelbasierte CEP-Systeme werten Eventströme aus, gehen dabei aber Schritt für Schritt vor, während ESP-Systeme in der Regel ganze Eventinstanzen suchen (d.h. nachdem diese bereits vollständig eingetreten sind).

2.4. ECA-Regeln

Ein weit verbreiteter regelbasierter Ansatz sind Event-Condition-Action-Rules (ECA-Regeln). Diese stellen eine detaillierte Anleitung in drei Teilen dar, welche die Konstruktion eines komplexen Events aus atomaren und komplexen Events, zusätzlichen Konditionen und auszuführenden Aktionen beschreibt. Wird ein Event erkannt, müssen alle beschriebenen Konditionen der ECA-Regel evaluiert werden. Sind alle Konditionen wahr werden alle Aktionen dieser Regel ausgeführt. Die ursprüngliche Struktur aus Event, Kondition und Aktion geht auf das HiPAC project [10] zurück, das reaktives Verhalten von Datenbanken untersuchte. Erweiterungen dieser grundlegenden Struktur werden bei den meisten Ansätzen zur Verarbeitung von Events vorgenommen. Die in dieser Arbeit verwendete Form von ECA-Regeln teilt die Kondition in 'Query' und 'Test' auf. Dadurch können zusätzliche Informationen, die nicht direkt mit dem erkannten Ereignis in Verbindung stehen, zur Auswertung einer Kondition herangezogen werden (siehe 5.1).

2.5. Reaktivität

Reaktivität ist eine wichtige Eigenschaft jedes modernen Datenbanksystems. Reaktives Verhalten von Datenbanken wurden als eine effiziente Erweiterung herkömmlicher relationaler oder objektorientierter Datenbanken eingeführt um automatisch auf Ereignisse reagieren zu können, welche intern oder außerhalb der Datenbank auftreten können. Reaktive Funktionalität eines Datenbanksystems (DBS) wird z. B. durch sogenannte Trigger und Constraints auf Tabellenebene bereitgestellt. Die Einführung von reaktivem Verhalten auf Datenbanken erweiterten deren Funktionalität und Einsatzmöglichkeiten deutlich. Datenbanken wurden von einem Sklaven der auf sie zugreifenden Anwendungen zu einem gleichberechtigten Partner, der nicht nur auf Anfragen reagiert, sondern durch das Anstoßen und Ausführen von Prozessen die Aktivitäten einer Anwendung unterstützen oder kontrollieren kann. Erste Arbeiten zu reaktiven Datenbanken beziehen sich hauptsächlich auf relationale Datenbanksysteme, bei denen Regeln für das Eintreten von Ereignissen als globale Restriktionen behandelt werden. [11, 12]

Weitere Vorschläge basierend auf Event-Condition-Action (ECA) Rules im Kontext von objektorientierten Datenbanksystemen findet man bei Chakravarthy [13], Berndtsson [14], Diaz [15] und Dittrich [16]. Einen umfassenden Überblick über reaktive Konzepte in Datenbanksystemen bieten die Bücher 'Active Database Systems: Triggers and Rules for Advanced Database Processing.' [17] und 'Active rules in database systems' [18].

Reaktivität in Triple-Stores ist dagegen weit weniger verbreitet und zur Zeit ein aktiver Forschungsgegenstand [19, 20] (vergleiche Kapitel 4). Reaktive Konzepte für für RDF Daten, werden in Ansätzen in verschiedenen Implementierungen der Storage And Inference Layer API (SAIL) des Triple-Stores 'Sesame' eingesetzt [21]. Im Gegensatz zu Triggern, welche als Erweiterung von SQL leicht einzusetzen sind, müssen diese Ereignisse direkt über die SAIL-API registriert werden und stehen nicht als Erweiterung einer Abfragesprache wie SPARQL zur Verfügung.

Anmerkung: Die Verwendung des Adjektives 'reaktiv' beschreibt das Verhalten einer Datenbank genauer als der ebenfalls verbreitete Term 'aktive' Datenbank, da stets auf eine zuvor definierte Art von Ereignissen reagiert wird.

Ein Ereignis, das durch einen Trigger signalisiert wird, soll im Kontext dieser Arbeit als atomares Event verstanden werden.

2.6. SOAP

Mit Hilfe des Netzwerkprotokolls SOAP (ursprünglich für: Simple Object Access Protocol), können Daten systemübergreifend ausgetauscht werden. Dieser industrielle Standard [22] des World Wide Web Consortium (W3C) stützt sich dabei auf eine XML-Repräsentation der Daten und den Internet-Protokollen HTTP, bzw. HTTPS zur Übertragung der Nachricht. Über SOAP-Endpunkte können Funktionen und Methoden veröffentlicht werden, von denen durch eine passende SOAP-Nachricht an den SOAP-Endpunkt von jedem berechtigten Benutzer im Netzwerk Gebrauch gemacht werden kann. Dadurch kann jede Art von Service über lokale Netzwerke oder das Internet verfügbar gemacht werden.

3. Anforderungen

Das in dieser Arbeit vorgestellte Framework für komplexe Events soll unterschiedlichen Anforderungen verschiedener Interessenten gerecht werden. Dazu wird im Folgenden, an Hand von verschiedenen Anwendungsfällen, eine Anforderungsanalyse durchgeführt. Aus den verschiedenen Anwendungsfällen (AF) sollen konkrete Benutzeranforderungen (BA) herausgearbeitet werden, die die Aufgaben dieses Frameworks genauer eingrenzen sollen. In einer zusammenfassenden Auswertung wird versucht, Benutzeranforderungen mit gesteigerter Priorität zu unterstreichen.

3.1. Anwendungsfälle (AF)

Anwendungsfälle sollen Einsatzmöglichkeiten des angestrebten Frameworks für komplexe Events darstellen, an Hand derer Anforderungen an das Framework abgeleitet werden.

3.1.1. AF1: Cloud-Events

Ein Cloud-Service will seine User möglichst genau und zeitnah über Änderungen an den von ihnen genutzten Ressourcen informieren. Dazu gehören Berichte über neue Daten, Änderungen an vorhandenen Dateien oder die Dokumentation über die Nutzung von bestimmten Funktionalitäten des Cloud-Service. Administratoren und andere Benutzer sollen definieren können, über welche Events sie informiert werden wollen und bei Bedarf automatisch auszuführende Aufgaben bestimmen, welche durch ein Event in der Cloud initiiert werden.

3.1.2. AF2: Automatisiertes Dokumentenverwaltungssystem

Auf Grundlage des unter AF1 beschriebenen Cloud-Service möchte ein Unternehmen ein umfassendes Dokumentenverwaltungssystem implementieren. Ziel ist es alle im Unternehmen entstehenden Dokumente automatisch zu erfassen. Textinhalte sollen automatisch mit Hilfe eines Name-Entity-Recognition-Verfahrens auf mögliche Schlagworte (engl. Tags) untersucht werden, die die Grundlage der Dokumentenverwaltung bilden. Mitarbeiter gehören dabei verschiedenen Gruppen an, die auf Dokumentenressourcen verschiedene Zugriffsrechte haben.

3.1.3. AF3: Produktionsablaufüberwachung

Der Produktionsablauf in einem Fertigungsbetrieb soll mittels Barcode-Scanner überwacht werden. Ziel ist dabei den Produktionsverlauf jedes einzelnen Artikels zu dokumentieren. Wichtige und unvorhergesehene Ereignisse sollen den zuständigen Mitarbeitern sofort über deren Tablet-Computer oder Smartphone angezeigt werden.

3.1.4. AF4: Benutzerkontenabsicherung eines Onlineshops

In einem Onlineshop sollen Veränderungen an Benutzerkontodaten automatisch überwacht werden um auf auffällige Aktivitäten rechtzeitig und angemessen reagieren zu können. Eine Vorkehrung besteht darin Änderungen an Passwörtern und E-Mail-Adressen eines Kontos zu registrieren und auf ungewöhnliche Änderungsaktivitäten zu achten. Werden Passwort und Adresse innerhalb weniger Tage geändert, könnte dies auf einen möglichen Eingriff eines unbefugten Akteurs hinweisen, der an das Passwort des Kontoinhabers gelangt ist und nun versucht das Konto ganz unter seine Kontrolle zu bringen. wird von Seiten des Anbieters die Vorsichtsmaßnahme getroffen, alle Nachrichten über die Änderung von Passwörtern an alle von diesem Nutzer bekannten E-Mail-Adressen zu senden, auch wenn diese vor weniger als einem Monat vom Nutzer gelöscht wurden. Damit würde im oben beschriebenen Betrugsszenario der richtiger Nutzer von den Vorgängen auf seinem Benutzerkonto informiert.

3.1.5. AF5: Flexible Teilautovermietung

Eine Teilautovermietung möchte ihr Betriebsmodell umstellen. Künftig sollen einige Fahrzeuge ihrer Flotte nicht nur an bestimmten Abgabestellen zu vorherbestimmten Zeiten verfügbar sein. Kunden bekommen flexible Zeitfenster eingeräumt und sollen die Möglichkeit haben das Fahrzeug überall (innerhalb der Stadt) abzustellen. Ein GPS-basiertes System überwacht dabei die zurückgelegten Strecken und Fahrzeiten, sowie die aktuelle Position des Fahrzeuges. Werden Fahrzeuge in einem bestimmten Teil der Stadt benötigt, sollen die Fahrer über eine Smartphone-Applikation darüber informiert werden. Über Preisnachlässe und Aufschläge sollen die Abgabeorte beeinflusst werden. Meldet sich ein Kunde ab, wird das Fahrzeug zur Vermietung freigegeben. Falls eine Reservierung für dieses Fahrzeug besteht, wird der entsprechende Kunde über die aktuelle Position informiert.

3.1.6. AF6: Automatische Aufgaben in sozialen Netzwerken

In einem sozialen Netzwerk (z. B. auf Grundlage von Xdox [23]) soll es jedem User möglich sein auf bestimmte Events eines Profils automatisch zu reagieren. Neben Benachrichtigungen über zuvor definierte Vorgänge auf eigenen oder fremden Profilen, könnte man z. B. alle Beiträge, die von einem andern User positiv bewertet wurden automatisch vermerken lassen oder immer positiv auf Freundesanfragen reagieren.

3.1.7. AF7: Zentrales Event-System einer Hochschule

Die Hochschule von Eisenhüttenstadt verfügt über mehrere Fakultäten und zahlreiche Institute. Veranstaltungen und Termine der einzelnen Fakultäten werden nicht zentral verwaltet, sondern von jeder Fakultät über eigene Webseiten und Informationsdienste veröffentlicht. Da keine Einigung auf ein einheitliches System zwischen den Fakultäten möglich scheint, wird das Institut für Informatik dazu angehalten alle bekannten Webseiten und Informationssysteme, die von Angehörigen der Hochschule zur Veröffentlichung von Informationen genutzt werden, automatisch zu überwachen, neue Informationen zu extrahieren und zentral über ein neues Event-System zu publizieren.

3.1.8. AF8: Überwachung von Anlagen erneuerbaren Energie

Ein Landwirt lässt auf einem Feld mehrere Windräder errichten. Da er meist keine Zeit hat die aktuellen Daten an einem PC zu überwachen, wäre er besonders für eine Applikation auf einem mobilen Endgerät dankbar, welche alle wichtigen Daten zusammenfasst und automatisch auf besondere Events aufmerksam macht. Ein betreuender Techniker wird automatisch über Betriebsausfälle oder andere Störungen informiert.

3.1.9. AF9: Automatisierte Terminplanung

Der Arbeitsplan von angestellten Ärzten soll in einem Krankenhaus automatisch gesteuert und optimiert werden. Dazu wird ein ständig aktualisierter Terminplan für den entsprechenden Arzt geführt, der den Behandlungsort, den Namen des Patienten, das Ergebnis der Vordiagnose und die dafür angestrebte Zeitspanne angibt. Erinnerungen oder Notfälle werden visuell und akustisch kommuniziert. Neue Patienten/Termine werden je nach Dringlichkeit, Spezialisierung des Arztes und vorhandenen Kapazitäten automatisch oder manuell in den aktuellen Plan eingeordnet.

3.1.10. AF10: Lokale Auswertung von Wahlkampfstrategien

Aus einer Vielzahl von Umfrageergebnissen in allen Wahlbezirken eines Landes sollen Trendwenden, starke Änderungsbewegungen und ähnlich wichtige Ereignisse gefiltert werden, um so frühzeitig auf lokale Wählerreaktionen eingehen zu können. Ein umfassendes Bild der lokalen Verhältnisse einer Wählerschaft soll es ermöglichen gezielt auf die Gunst des Wählers einzuwirken. Reaktionen auf ein bestimmtes Ereignis sollen schnell abgelesen werden und wenn nötig Gegenmaßnahmen getroffen werden.

3.2. Benutzeranforderungen

Die beschriebenen Anwendungsfälle haben verschiedene Anforderungen aufgezeigt, welche im Folgenden zusammengefasst werden.

3.2.1. BA1: Verteiltheit

Das angestrebte Framework soll für eine Vielzahl unterschiedlicher Datenbanken, Triple-Stores und anderen Eventquellen verfügbar sein, so dass komplexe Events aus atomaren Events verschiedener Quellen konstruiert werden können. Auch Konditionen und Aktionen sollen auf verschiedenen Servern ausgeführt, bzw. abgefragt werden.

3.2.2. BA2: Systemunabhängigkeit

Die Kommunikation zwischen verschiedenen Betriebs- und Datenbanksystemen ist zu gewährleisten. Dazu muss auf systemunabhängige Netzwerkprotokolle zurückgegriffen werden. Außerdem ist der Betrieb des Event-Frameworks auf unterschiedlichen Betriebssystemen sicherzustellen, um dessen Einsatzmöglichkeit so vielseitig wie möglich zu gestalten.

3.2.3. BA3: Eigenständigkeit

Das Event-Framework soll unabhängig von der Funktionalität externer Ressourcen operieren und damit eigenständig die Evaluation von komplexen Events vornehmen. D. h. es sollen Zeitverzögerungen durch zusätzliche Kommunikation mit Ressourcen (z. B. externe Rule-Engines), welche nicht auf dem selben Server zur Verfügung stehen, vermieden werden. Außerdem steigt so die Sicherheit beim Umgang mit vertraulichen Daten.

3.2.4. BA4: Zuverlässigkeit bei großen Zugriffszahlen

Die Zuverlässigkeit des Frameworks auch bei großen Zugriffszahlen ist entscheidend für die Verarbeitung vieler Events innerhalb eines kurzen Zeitfensters. Es muss sichergestellt sein, dass ein komplexes Event auch bei einer Vielzahl eingehender atomarer Events sicher erkannt wird und Konditionen sowie Aktionen zuverlässig ausgeführt werden.

3.2.5. BA5: Verlässlichkeit bei der Erkennung von Events

Das zuverlässige Erkennen von atomaren und komplexen Events ist eine zentrale Anforderung an das Framework. Wird ein zuvor registriertes Event signalisiert, so muss dies vom Framework erkannt und verarbeitet werden. Gleichsam muss ein vom Framework erkanntes Event auch tatsächlich eingetreten sein. Komplexe Events müssen in korrekter Reihenfolge bearbeitet und alle Konditionen ausgewertet werden.

3.2.6. BA6: Echtzeit

Tritt ein komplexes Event ein, sollte sichergestellt sein, dass dieses sofort erkannt wird und alle damit verbundenen Aktionen zeitnahe ausgeführt werden. Hierbei ist eine Verarbeitung in Echtzeit angestrebt, d. h. in klar definierten, möglichst kurzen Zeitfenstern.

3.2.7. BA7: physische Datenunabhängigkeit

Neben den von diesem Framework eingesetzten Datenbanken sollten alle Datenquellen (z. B. Triple-Stores) eine interne, physische Datenunabhängigkeit der durch Sie zur Verfügung gestellten Daten bieten. Dies ist Voraussetzung für eine fehlerfreie Auswertung von Konditionen und Events im Allgemeinen.

3.2.8. BA8: Datenschutz und Sicherheit

Der Schutz von übertragenen Daten sowie die Verifizierung von registrierten Daten- oder Eventquellen ist eine wichtige Anforderung für alle Kommunikationsvorgänge des Frameworks. Nur so kann eine fehlerfreie Auswertung von Events und eine Geheimhaltung von sensiblen Daten garantiert werden.

3.2.9. BA9: freie Wahl und Definition von Events und Ressourcen

Es darf keinerlei Einschränkung bei der Definition von Events und der Auswahl von Ressourcen geben, sofern es keinen semantischen oder syntaktischen Grund hat. Atomare und komplexe Events sollen bei Bedarf direkt mit dem Event-Framework zu definieren sein.

3.2.10. BA10: Verwaltung von Benutzern und Eventressourcen

Über die Einteilung in Administratoren und anderen Usern mit unterschiedlichen Berechtigungen soll das Framework auf die Anforderungen verschiedene Benutzer skalierbar gemacht werden. Berechtigungen für verschiedene Ressourcen (z. B. Datenquellen, Verwendung von bestimmten Events u.ä.) erlauben die Nutzung eines Event-Frameworks von verschiedenen Usern für unterschiedliche Aufgaben.

3.2.11. BA11: RDF-Events

Durch die wachsende Bedeutung des Semantic Webs [2], lohnt die Betrachtung aller unter 3.1. beschriebenen Szenarien aus einem neuen Blickwinkel. Wählt man eine RDF-basierte Datenbasis an Stelle von herkömmlichen Datenbank-basierten Lösungen, rückt das Problem der Reaktivität von Triple-Stores in das Zentrum der Aufmerksamkeit. Das angestrebte Framework sollte eine Möglichkeit für das Erkennen von atomaren Events auf einer RDF-basierten Datenbasis bieten.

3.3. Auswertung

Zur besseren Einschätzung der Tragweite von einzelnen Benutzeranforderungen sollen diese nun im Bezug auf jeden Anwendungsfall gewichtet werden. Dazu wurde eine Tabelle erstellt, welche für jedes Paar aus Anwendungsfall und Benutzeranforderung einen Wert zwischen 1 und 10 aufführt. Diese Werte geben eine subjektive Einschätzung des Autors über die Wichtigkeit einer Benutzeranforderung für den entsprechenden Anwendungsfall an. Die Summe aller Gewichtungen einer Benutzeranforderung soll ein klareres Bild für eine allgemeine Gewichtung der Benutzeranforderungen und damit die Prioritäten beim Entwurf des Event-Frameworks vorgeben.

	BA1: Verteiltheit	BA2: Systemunabhängigkeit	BA3: Eigenständigkeit	BA4: Zuverlässigkeit	BA5: Verlässlichkeit	BA6: Echtzeit	BA7: Datenunabhängigkeit	BA8: Datenschutz	BA9: Definitionsfreiheit	BA10: Verwaltung
AF1: Cloud-Events	10	8	5	9	9	8	5	9	6	9
AF2: Dokumentenverwaltungssystem	8	5	4	7	7	6	8	4	6	8
AF3: Produktionsablaufüberwachung	4	3	9	10	10	9	7	6	7	3
AF4: Benutzerkontenabsicherung	3	2	9	5	9	6	8	10	5	9
AF5: Teilautovermietung	9	8	4	9	9	6	3	7	7	10
AF6: Soziale Netzwerke	8	8	2	8	8	5	2	7	8	9
AF7: Zentrales Event-System	5	8	5	5	5	1	3	2	7	1
AF8: Anlagen für erneuerbare Energie	9	4	5	8	8	10	6	7	7	6
AF9: Terminplanung	6	5	4	8	9	9	5	8	8	9
AF10: Wahlkampfstrategien	4	4	3	6	7	4	9	8	9	6
Summe	66	55	50	75	81	64	56	68	70	70

Nach der Einschätzung des Autors liegt ein Schwerpunkt beim Entwerfen des Event-Frameworks auf der Zuverlässigkeit bei der Erkennung und korrekter Verarbeitung von Events, sowie der Ausführung aller Aktionen auch unter erschwerten Bedingungen (z. B. durch hohe Zugriffszahlen). Weiterhin besteht ein gesteigertes Interesse an einer einfachen Verwaltung und einer leichten Steuerung des Frameworks. Die Einbeziehung verschiedener Ressourcen als Eventquellen oder zur Ausführung von zuvor definierten Aktionen, die Definition von komplexen Events ohne Einschränkungen sowie die Sicherheit der übertragenen Daten haben ebenfalls eine hohe Priorität. RDF-Events (BA11) wird gesondert betrachtet und als eine Anforderung mit hoher Priorität aufgenommen.

4. Bestandsaufnahme

Erste Formen von Complex Event Processing (CEP) [24] Systemen existieren bereits seit den 80er Jahren, meist in klar umrissenen Domänen wie z. B. Datenbanken und Netzwerkumgebungen. Weiterreichende Ansätze, welche Events verschieden Ursprungs verarbeiten und sich frei durch Benutzer konfigurieren lassen, gehören in die jüngere Geschichte der Informatik. Im Jahr 2001 kam der Begriff des Business Activity Monitoring (BAM) auf, der den Bereich charakterisierte, auf den das CEP abzielte. Um die Arbeit mit CEP zu erleichtern, kamen zu dieser Zeit auch die ersten grafischen Werkzeuge auf, die es ermöglichten, dass auch Laien neue Muster für komplexe Ereignisse definieren konnten, ohne Programmcode schreiben zu müssen [8]. Den meisten CEP-Systemen ist gemein, dass Sie auf Daten-, bzw. Eventströmen (engl. event streams) beruhen, aus denen durch eigene Abfragesprachen komplexe Events 'gefiltert' werden. In diesem Zusammenhang spricht man auch von Event Stream Processing (ESP). Solche Systeme verwenden Complex Event Processing als Teil der Analyse von Datenströmen.

Neben universalen CEP-Systemen, werden in diesem Kapitel RDF-basierte CEP-Systeme gesondert betrachtet, da eine Generierung von atomaren Events in Triple-Stores zur Zeit nicht ohne Weiteres möglich ist und daher genauer betrachtet werden sollte. Datenbanken als Form eines CEP-Systems, bzw. reine Datenbank-basierte CEP-Systeme werden hier nicht weiter berücksichtigt.

4.1. Universale Complex Event Processing Systeme

Als universal werde alle CEP-Systeme verstanden welche auf keiner spezifischen Datenbasis operieren. Das heißt, dass diese CEP-Systeme keine bestimmte Datenbank oder Schnittstelle voraussetzen, die als Eventquelle dient. Vielmehr kann auf mehrere Quellen möglicher Events zurückgegriffen werden, die sich durch einen Anwender frei definieren lassen.

4.1.1. Event-Stream-Processing-System: Odysseus

Bei der Suche nach Lösungen für die in Kapitel 3. herausgearbeiteten Benutzeranforderungen, zeigt sich, dass umfassende Lösungsansätze zur Verarbeitung von komplexen Events existieren. Einen sehr weitreichenden Ansatz bietet das Odysseus Projekt der Abteilung für Informationssysteme an der Universität Oldenburg. Dabei handelt es sich um ein sogenanntes Datenstrommanagementsystem welches mit Hilfe eines CEP-Systems "...[ermöglicht], komplexe Ereignisabfolgemuster zu definieren und in Ereignisdatenströmen zu erkennen"[25]. Odysseus ist ein hoch modulares und erweiterbares System, welches durch Abfragesprachen auf Datenströmen verschiedener Eventquellen komplexe Events filtern und die resultierenden Ergebnisse an sogenannten Senken weitergibt. Senken können mit Anwendungen verbunden werden oder als Daten- bzw. Eventquelle einer weiteren Odysseus-Instanz dienen. Nach den Angaben über dieses Projekte [26] wird Odysseus vielen der in Kapitel 3. aufgeführten Benutzeranforderung gerecht (BA1,2,3,4,5,6,10), aber es gibt auch einige Einschränkungen. Da es sich bei Odysseus um eine Middleware handelt, existiert eine Datenebene nur im Bezug auf die Benutzerverwaltung. Die Konsistenz von Daten und Datenunabhängigkeit (BA7) liegen daher außerhalb der Domäne von Odysseus. Mit Odysseus werden in erster Linie Datenströme überwacht aus denen komplexe Events gefiltert werden. Zusätzliche

Konditionen, welche auf Informationen außerhalb dieser Datenströme zugreifen, müssen gesondert gehandhabt werden. Atomare Events werden als Bestandteile von Datenströmen vorausgesetzt und können nicht mit Odysseus definiert werden (BA9). Obwohl eine eigene Abfragesprache (Streaming SPARQL) für RDF-Datenströme bereitgestellt wird, umfasst Odysseus keinen inhärenter Ansatz zur Erstellung eines solchen Stromes (BA11). Die Bereitstellung atomarer Events als Eventstrom und dessen zuverlässige und sichere Übertragung liegt daher allein in der Hand des Nutzers.

Als weitere große CEP- bzw. ESP-Systeme sind die folgenden Projekte zu nennen:

1. Telegraph (UC Berkeley) [27]
2. CEPiL (University of Stuttgart, GeorgiaTech) [28]
3. STREAM und Rapide (Stanford University) [29, 30]

4.2. RDF-basierte CEP-Systeme

Neben der Entwicklung eines Systems zur Verarbeitung komplexer Events auf Basis von RDF-Metadaten besteht eine zusätzliche Schwierigkeit darin, einen Mechanismus zu konstruieren, der atomare Events auf der Ebene von RDF-Tripeln erkennt (z. B. das Einfügen eines neuen Tripels mit einer bestimmten Ausprägung) und diesen Vorgang als Event signalisiert. Dazu ist es außerdem nötig eine eigene Sprache zu definieren, bzw. vorhandene Sprachen wie SPARQL um eine eigene Syntax zur Definition dieser Events zu erweitern. Während man auf modernen Datenbanken mittels SQL eine Vielzahl von Möglichkeiten hat Events festzulegen (z. B. mittels Trigger oder Constraints), besitzt SPARQL (in Version 1.1) keine Syntax zur Definition von Events.

Listing 2: SQL-Syntax eines Update-Triggers in ECA-Struktur

Name des Triggers	>	CREATE TRIGGER AbtBudget
Event (vor jedem Update des Wertes 'Budget' auf der Tabelle 'Abteilung')	>	BEFORE UPDATE Budget ON Abteilung
	>	REFERENCING OLD ROW AS AltWert
Kondition (falls spaeter als 17 Uhr)	>	WHEN (CURRENT_TIME > TIME '17:00:00')
Aktion (loesche Tabelle 'AltWert')	>	DELETE FROM AltWert;

4.2.1. RDFTL

Ein Vorschlag für eine SQL-ähnliche Syntax auf Grundlage von von RDF-Daten wurde durch Peter T. Wood und Kollegen 2004 unter dem Begriff RDFTL (RDF Triggering Language) vorgestellt [31]. Außerdem wurde ein Framework präsentiert, welches alle technischen Voraussetzung zur Verarbeitung von Events bietet die durch RDFTL definiert wurden und damit eine aktive Hülle für passive (nicht reaktive) Triple-Stores schafft. Dadurch wird die Funktionalität eines Triple-Store um SQL-ähnliche Trigger erweitert und somit die Veröffentlichung von atomaren Events auf (verteilten) RDF-Metadaten realisiert.

Listing 3: RDFTL-Syntax eines Events in ECA-Struktur

Event (beim Update eines Triples)	> ON UPDATE (resource(),dc:description,->-)
Kondition (falls bestimmte Ressource Subjekt dieses Triples ist)	> IF \$delta/target(dc:subject) = resource(http://www.dcs.bbk.ac.uk/users/128) /target(ext1:interest)/element() /target(ext1:interest_typename)
Aktion	> DO LET \$updated_lo_list := resource(http://www.dcs.bbk.ac.uk/users/128) /target(ext3:messages)/target(ext3:updated_LOs) IN INSERT (\$updated_lo_list,seq++,\$delta);

Zentraler Bestandteil des vorgestellten Frameworks für zentrale und dezentrale Umgebungen ist eine lokale ECA-Engine die aus den folgenden Teilen besteht:

1. **RDFTL Language Interpreter** - interpretiert die RDFTL-Trigger Definition und registriert die damit verbundene ECA-Regel
2. **Event-Detector** - erkennt Events in den RDF-Metadaten und stößt die zuvor registrierten ECA-Rules an
3. **Condition Evaluator** - evaluiert die registrierten ECA-Rules und erkennt welche Regel erfüllt wurde
4. **Action Scheduler** - generiert eine Liste von auszuführenden Aktionen und regelt deren Ausführung
5. **Peer Connection Manager** - regelt die Kommunikation mit anderen ECA-Engines in einem Netzwerk

Diese Lösung ist kein eigentliches CEP-System, da es z. B. keine komplexen Events erkennt, die sich über mehrere Operationen erstrecken, sondern allein die Situation zum Zeitpunkt des Events auswerten kann (vergleichbar zu SQL-Triggern).

4.2.2. REVERSE I5

Das EU-finanzierte, multizentrische Projekt REVERSE (Reasoning on the Web with Rules and Semantics) befasst sich seit 2004 mit verschiedenen Forschungsschwerpunkten des Semantic Web. Die Arbeitsgruppe I5 'Evolution and Reactivity' hat sich zur Aufgabe gestellt "...[eine] deklarative Sprache, Methodik und Werkzeuge zur Spezifikation und Umgang mit Reaktivität, Evolution und Verbreitung von Veränderungen im Web zu entwickeln"[32]. In dem 2009 veröffentlichten Artikel 'Evolution and Reactivity in the Semantic Web' wird ein Großteil der von I5 erlangten Ergebnisse beschrieben[19]. Darin wird eine homogene Sprache (XChange) für ECA-Regeln im Semantic Web präsentiert, welche es ermöglicht komplexe Events in verteilten Umgebungen inklusive Konditionen und Aktionen zu definieren.

Listing 4: Eine XChange ECA-Regel: Der Passagier "John Q Public" soll per SMS bei dem Ausfall seines Fluges verständigt werden

ON	xchange : event {{
	flight - cancellation {{
	flight - number { var N },
	passenger {{
	name { " John Q Public " }
	}} }} }}
IF	in { resource { " http :// www . example . com/ flights .xml", " xml" },

```

    flights {{
      flight {{
        number { var N },
        from { var F },
        to { var T }
      }} }} }
DO and {
  xchange : event [
    xchange : recipient [ "http :// sms - gateway .org/us /206 -240 -1087/" ],
    text - message [
      "Hi , John! Your flight ", var N,
      " from ", var F, " to ", var T, " has been canceled ."
    ] ],
  in { resource { " http :// shuttle . com/ reservation.xml", " xml" },
    reservations {{
      delete shuttle -to - airport {{
        passenger { " John Q Public " },
        airport { var F },
        flight { var N }
      }} }} }
END

```

Besonderes Augenmerk gilt der Kondition dieser Regel (IF-Teil). Anstelle der Evaluation einer Kondition mit dem Ergebnis 'wahr' oder 'falsch' werden hier nur zusätzliche Informationen abgefragt. Der Abflugsort (from) und Destination (to) werden in die Variablen F und T zwischengespeichert und in der Definition der Aktion (DO-Teil) wiederverwendet. Die hier verwendeten ECA-Regeln sind eine Erweiterung der ursprünglichen Form *ON event IF condition THEN DO something*. Durch die Unterteilung der Kondition in 'Query', also das Beschaffen zusätzlicher Informationen, die nicht direkt im Zusammenhang mit dem Event stehen, und 'Test' (der eigentlichen Evaluation) kann die Spanne an möglichen Szenarien von komplexen Events deutlich erweitert werden.

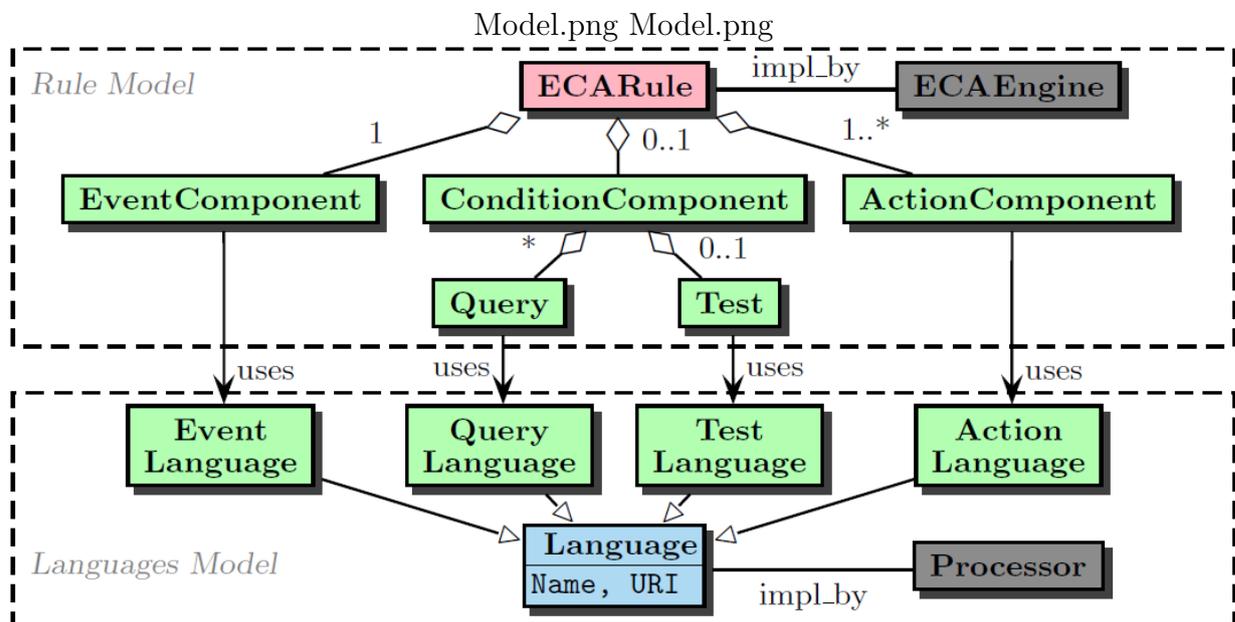


Abbildung 1: ECA-Regel-Komponenten und deren korrespondierenden Sprachen (aus [33])

Neben einer umfangreichen Ontologie, welche die Semantik von Events, Konditionen und Aktionen auf RDF-Ebene vorgibt, wurde aufbauend auf XChange ein Framework vorgestellt, dessen Grundansatz die Unterteilung einer ECA-Regel in deren Komponenten ist.

Die Umsetzung der Evaluation von Einzelkomponenten als Service im Framework, soll eine modulare Evaluation von komplexen Events ermöglichen, die sich dadurch verschiedener Evaluationsmodelle bedienen kann. Die respektive Sprache jeder Komponente basiert auf der Semantik von XChange (vergleiche Abbildung 1).

Das Zusammenspiel der einzelnen Komponenten lässt sich wie folgt zusammenfassen:

1. ein Client registriert eine neue Regel bei der **ECA-Engine**
2. mindestens eine **Composite Event Detection Engine** (für komplexe Events) evaluiert, mit Hilfe eines **Atomic Event Matchers** (zur Erkennung atomare Events), ECA-Regeln und gibt bei Eintreten das Ergebnis an die ECA-Engine zurück
3. die ECA-Engine ruft die zu verwendenden Komponenten für 'Query' und 'Test' auf
4. werden alle Konditionen mit 'wahr' beantwortet, wird eine **Action Engine** mit der Ausführung aller verbundenen Aktionen betraut

Dieser umfassende Ansatz zur Erkennung, Verarbeitung und Ausführung aller Komponenten einer ECA-Regel auf der Basis einer homogenen, deklarativen Sprache in einem Framework, stellt ein mächtiges Werkzeug für komplexe Events auf verteilten RDF-Datenquellen dar. Die Einteilung des Frameworks nach den Teilen einer ECA-Regel ermöglicht den Einsatz verschiedener Werkzeuge zur Erkennung und Verarbeitung von auftretenden Events. XChange als Sprache für die einheitliche Definition einer kompletten Regel verbirgt die eigentliche Modularität des Frameworks und vereinfacht den Umgang mit komplexen Events.

Das vorgestellte Framework wird vielen Benutzeranforderungen gerecht (BA 1,3,4,5,6,7). Da es sich um einen RDF-basierten Ansatz handelt, wird BA11 erfüllt. Die Definition von nicht RDF-basierte Events ist nicht angedacht (BA9). Es wurde keine Verwaltungs- / Benutzerebene implementiert (BA10). Die Definition von Events erfolgt über XChange. Daher ist eine Anwendung des Frameworks für Laien nicht ohne weiteres zu realisieren.

5. Spezifikation

Die in Kapitel 3. herausgearbeiteten Benutzeranforderungen dienen als Grundlage für die Spezifikation der in dieser Arbeit vorgestellten Software. Der Name Event-Framework unterstreicht den Schwerpunkt der Arbeit. Das in Kapitel 1.2. beschriebene Ziel (Umgang mit jeder Art von Event, in einer Umgebung verteilter Daten- und Eventquellen) ist daher bei der Spezifikation des Event-Frameworks als oberste Priorität zu betrachten.

Nach der Einführung in die verwendete Terminologie, wird der angestrebte Aufbau des Frameworks definiert. Verwendete Strukturen für persistente und nicht persistente Daten werden vorgestellt und die Anforderungen an einzelne Softwarekomponenten genauer erläutert.

5.1. Terminologie

Action (Aktion)

Eine Action wird optional als Reaktion auf den Eintritt eines komplexen Events ausgeführt. Dafür kommt jede denkbare Funktion, die über einen SOAP-Endpunkt veröffentlicht wurde in Frage.

Active-Complex Event (aktiv-komplexes Event)

Ein komplexes Event, das bei Eintritt mindestens eine Aktion auslöst, heißt aktiv-komplexes Event.

Atomic Event (Atomares Event)

Ein atomares Event beschreibt auf Tabellenebene der eingesetzten Satellitendatenbanken alle durch Trigger aufgezeichneten Ereignisse. D. h. alle Insert, Update- oder Delete-Vorgänge von Tupeln, die zuvor durch einen berechtigten Nutzer mit Hilfe des Event-Framework-Control als ein Auslöser für ein atomares Event spezifiziert wurden. Zusätzlich werden alle externen Events welche über das Event-Framework-Control registriert werden als atomares Event bezeichnet.

Central Database (Zentraldatenbank)

Die Zentraldatenbank in Verbindung mit dem Event-Service das Herzstück des Event-Frameworks. In ihr werden alle Daten für die Abwicklung von komplexen Events und alle administrativen Informationen gespeichert. Über Datenbankprozeduren wird ein Großteil der internen Funktionen des Event-Frameworks bereitgestellt.

Client-Service

Der Client-Service bildet die Grundlage für die Kommunikation mit dem Event-Service von einem Client-Computer aus. Alle Applikationen, die auf Daten des Event-Frameworks zugreifen sollen, müssen auf die Funktionalität einer lokalen Client-Service-Instanz zurückgreifen. Dies wird mittels Inter-Process-Communication-Protokollen ermöglicht.

Complex Event (Komplexes Event)

Das komplexe Event ist das Ergebnis der Verknüpfung von einem oder mehreren atomaren oder komplexen Events unter Verwendung erweiterten ECA-Regeln.

Condition (Kondition, Bedingung)

Die Condition beschreibt eine optionalen Bedingung die für das Eintreten eines komplexen Events Vorbedingung ist.

Datasource (Datenquelle)

Eine Datenquelle ist ein SOAP-, SPARQL- oder ein anderer Endpunkt, welcher Queries dieses Frameworks auswertet und beantwortet. Event- und Datenquelle können in einer Instanz vereint sein.

Event

Ein Event fasst atomare und komplexe Events zusammen.

Event-Framework-Control (EFC)

Das EFC ist eine Applikation, die zur Steuerung des gesamten Frameworks dient. Registrierung von Datenquellen, Erstellung und Registrierung von atomaren Events, Festlegung von Actions und Conditions sowie die Konstruktion und Aktivierung von komplexen Events werden über diese Software ermöglicht.

Event-Service

Der Event-Service wird parallel zur Zentraldatenbank ausgeführt (vorzugsweise auf dem selben Server). Er dient als Schnittstelle für die Kommunikation mit dem Client-Service und übernimmt zusätzlich alle internen Funktionen, welche nicht direkt über Datenbankprozeduren umgesetzt werden können. Die Kommunikation zwischen Service und Zentraldatenbank wird mittels einer ODBC-Verbindung ermöglicht. Die Zentraldatenbank kommuniziert mit SOAP-Nachrichten mit dem Event-Service.

Eventsource (Eventquelle)

Eine Eventquelle signalisiert das Eintreten eines Events in Form einer SOAP-Nachricht an die Zentraldatenbank.

Extended ECA-Rule (erweiterte ECA-Regel)

Erweiterte ECA-Regeln stellt eine detaillierte Anleitung dar, welche die Konstruktion eines komplexen Events aus atomaren oder weiteren komplexen Events, zusätzlichen Konditionen und auszuführenden Aktionen angibt. Die hier verwendete Definition für ECA-Regeln orientiert sich an dem 'general framework for Event-Condition-Action-Rules' von Alferes, Eckert und May [19]. Diese stellt eine Erweiterung der herkömmlichen ECA-Regeln (*ON event IF condition THEN DO something*) dar [10] und lässt sich wie folgt kurz zusammenfassen:

ON event AND additional knowledge IF condition THEN DO something [19]

1. **Event** (ON event) - ein oder mehrere Events (atomar oder komplex), welche durch Boole'sche Operatoren verknüpft werden
2. **Query**(AND additional knowledge) Datenanfragen deren Ergebnisse zur Konstruktion der eigentlichen Konditionstestfrage verwendet werden
3. **Test** (IF condition) dieser Test wertet eine Konditionsanfrage aus und gibt einen Boole'schen Wert zurück

4. **Action** (THEN DO something) falls alle verlangten Events eingetreten sind und alle Konditionen erfüllt sind (alle Konditionsanfragen ergeben 'True'), führe keine oder mehrere Aktionen aus

Der wichtigste Unterschied zwischen herkömmlichen ECA-Regeln und der hier verwendeten erweiterten Form, ist die Unterteilung der Condition in Query und Test. Dies erlaubt eine einfache dynamische Gestaltung von Conditions und Actions. Alle Parameter die für das Ausführen von einer Condition benötigt werden, können im ersten Schritt (Query) von unterschiedlichen Datenquellen abgefragt werden. Darüber hinaus bietet das Event-Framework die Möglichkeit Parameter mit Werten aus atomaren Events zu versehen.

External Event (Externes Event)

Ein Externe Event ist ein atomares Event, welches nicht mit Hilfe des Event-Framework-Control erstellt wurde, sondern lediglich als solches registriert ist. Implementierung, Erstellung der SOAP-Nachricht, sowie deren Übertragung an die Zentraldatenbank müssen durch den entsprechenden Provider sichergestellt werden.

Linked-Data-View

Linked-Data-View ist ein Begriff von Virtuoso-Datenbanken [34], der die Abbildung von Tabelleninhalten der relationalen Datenbank Virtuoso auf Daten im RDF-Format umschreibt. Dadurch können die meisten Inhalte von Tabellen einer Virtuoso-Datenbank direkt als RDF-Triple eingesetzt werden.

Passive-Complex Event (passiv-komplexes Event)

Ein komplexes Event welches bei Eintritt keine Aktion auslöst, heißt passiv-komplexes Event.

Provider

Ein Provider ist ein Server (oder allg. Computer) der über einen SOAP-Endpunkt Actions, Conditions oder Queries zur Verfügung stellt. Unter Umständen kann auch die Entität gemeint sein, die einen solchen Server betreibt.

Query (Abfrage)

Als Query wird im Kontext dieser Arbeit der erste Teil jeder Condition (nach erweiterten ECA-Regeln) verstanden. In diesem Schritt werden zusätzliche Informationen (z. B. durch eine Datenbankabfrage) organisiert, welche z. B. zur Auswertung der eigentlichen Condition benötigt werden.

Satellite-Database (Satellitendatenbank)

Neben der Zentraldatenbank kann das Event-Framework auch um Satellitendatenbanken erweitert werden. Eine Satellitendatenbank vereinigt Aktionprovider, Daten- und Eventquelle in sich. Außerdem können Satellitendatenbanken direkt vom Event-Framework-Control aus verwaltet werden, so dass keine manuelle Definition von Triggern auf Datentabellen von Nöten ist. Alle Satellitendatenbanken erhalten einen SOAP-Endpunkt, welcher die Kommunikation mit der Zentraldatenbank und dem Event-Service ermöglicht.

Test

Als Test wird im Kontext dieser Arbeit der erste Teil jeder Condition (nach erweiterten ECA-Regeln) verstanden. Die Condition wird durch Resultate vorhergehender Queries

komplettiert und auf einem zuvor definierten Endpunkt ausgeführt. Als Antwort wird ein Boole'scher Wert erwartet, der das Ergebnis der Konditionsevaluation angibt.

Trigger

Dieser Begriff aus reaktiven Datenbanken zur Beschreibung eines Auslösers für Events auf einer Tabelle, wird in dieser Arbeit als Synonym für ein atomares Event auf einer Datenbank verwendet, da jedes Event einer eingesetzten Datenbank im Ursprung auf einen Trigger zurückzuführen ist.

5.2. Aufbau und Architektur

Das hier vorgestellte Event-Framework lässt sich als Schichtenarchitektur beschreiben, die sich vertikal von einem Data-Layer über eine Service-Layer bis hin zu Anwendungen auf Client-Computern (Application-Layer) verteilt. Horizontal soll das Event-Framework auf Grundlage mehrerer verteilter Datenbanken und anderer Daten- bzw. Eventquellen basieren und von verschiedenen Anwendungen auf unterschiedlichen Client-Computern genutzt werden können. Zur Kommunikation zwischen verschiedenen Computern im Framework sind Schnittstellen in Form von SOAP-Endpunkten vorgesehen. Das eingesetzte SOAP-Protokoll (vgl. Kapitel 2.6.) soll die Unabhängigkeit von Datenbanksystemen und Betriebssystemen (BA2 vgl. Kapitel 3.2.2.) in dieser verteilten Umgebung sicherstellen.

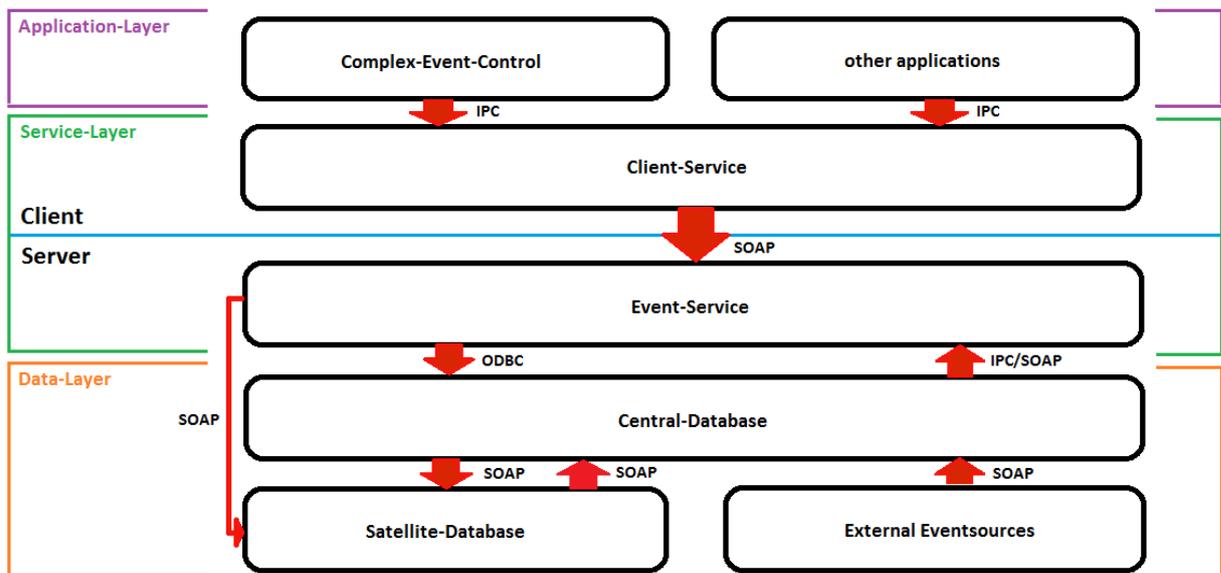


Abbildung 2: Event-Framework-Layer-Model

Da Zentraldatenbank und Event-Service funktional eng miteinander verbunden sind, sollten beiden Entitäten auf einem Server operieren, um Kommunikationsverzögerungen beim gegenseitigen Aufruf von Funktionen so gering wie möglich zu halten.

5.3. Data-Layer

Die Datenebene unterteilt sich in eine Zentraldatenbank sowie eine Vielzahl von Satellitendatenbanken und anderen Event- bzw. Datenquellen (siehe Abbildung 5.1). Die Zentraldatenbank ist das Herzstück des Event-Frameworks und übernimmt, neben der Speicherung aller Daten des Frameworks, einen Großteil der funktionalen Aufgaben. Diese werden über Datenbankprozeduren und Trigger bereitgestellt und ermöglichen damit einen effizienten Umgang mit großen Datenmengen, da Zeitverluste durch zusätzliche Kommunikation mit dem Event-Service (welcher diese Aufgaben sonst übernehmen müsste) vermieden werden. Das Empfangen einer SOAP-Nachricht über das Eintreten eines atomaren Events auf einer Satellitendatenbank, die anschließende Kontrolle auf komplexen Events, die auf das empfangene atomare Event 'warten' sowie alle Folgeprozesse, werden automatisch durch Prozeduren der Zentraldatenbank verarbeitet.

Als Datenspeicher wird die Zentraldatenbank für alle Daten des Frameworks eingesetzt (Benutzerinformationen, Eventdefinitionen, Eventlog...). Einfache, statische Datenobjekte, wie die Definitionen von atomaren Events, werden in Tabellenform abgespeichert und als Linked-Data-Views zusätzlich im RDF-Triple-Format zur Verfügung gestellt. Komplexe Datenobjekte, wie die Definitionen von komplexen Events (ECA-Rules), werden als RDF-Triple einem Triple-Store gespeichert. Das verwendete Datenbanksystem muss daher einen integrierten Triple-Store bieten oder die Möglichkeit haben einen externen Triple-Store einzubeziehen. Eine schematische Darstellung der verwendeten Ontologie als Grundlage für Definitionen komplexer Events ist im nächsten Unterkapitel zu finden.

Satellitendatenbanken dienen als eine Erweiterung des Event-Frameworks, um zusätzliche Datenbanken. Sie dienen als Event- und Datenquellen, die direkt über das Event-Framework-Control (EFC) verwaltet werden können. D.h. atomare Events müssen nicht manuell im Umfeld der entsprechenden Datenbank erstellt werden, sondern können dezentral mittels eines EFC verwaltet werden und nach Möglichkeit auch von Laien bedienbar sein. Datenbankprozeduren übernehmen das automatische Versenden von SOAP-Nachrichten an die Zentraldatenbank und dokumentieren alle Kommunikationsvorgänge in einer eigenen Tabelle der Zentraldatenbank. Andere Event- und Datenquellen liegen dagegen außerhalb der Domäne dieses Frameworks und müssen die geforderte Funktionalität, wie die Übertragung von richtig formatierten SOAP-Nachrichten an die Zentraldatenbank, selber implementieren.

5.3.1. Event Ontology

Zur semantischen Beschreibung von komplexen Events im RDF-Format kommt die folgende Ontologie zum Einsatz. Zentral ist darin die Unterteilung komplexer Events in Stages (Phasen), die als Teilevent zu verstehen sind und einen zeitlichen Rahmen haben. Wurde eine Stage erfolgreich beendet (alle erhofften Events sind eingetreten und alle Conditions sind erfüllt), dann wird automatisch die nächste Stage initiiert (Stage-Transition). Nach dem Abschluss der letzten Stage können alle definierten Actions eingeleitet werden. EventSets (inkl. InitialEventSets) in Verbindung mit Operatoren bilden die Grundlage für die Anordnung von Events in einer Stage. Durch die Boole'schen Operatoren kann klar ausgedrückt werden, welche Events in welchem Zusammenhang erwartet werden (oder nicht erwartet werden). Ein Zeitlicher Rahmen dafür wird durch die Zeitrestriktion einer Stage vorgegeben. Im Folgenden werden alle Konzepte und Eigenschaften der abgebildeten Ontologie kurz vorgestellt.

complex event ontology

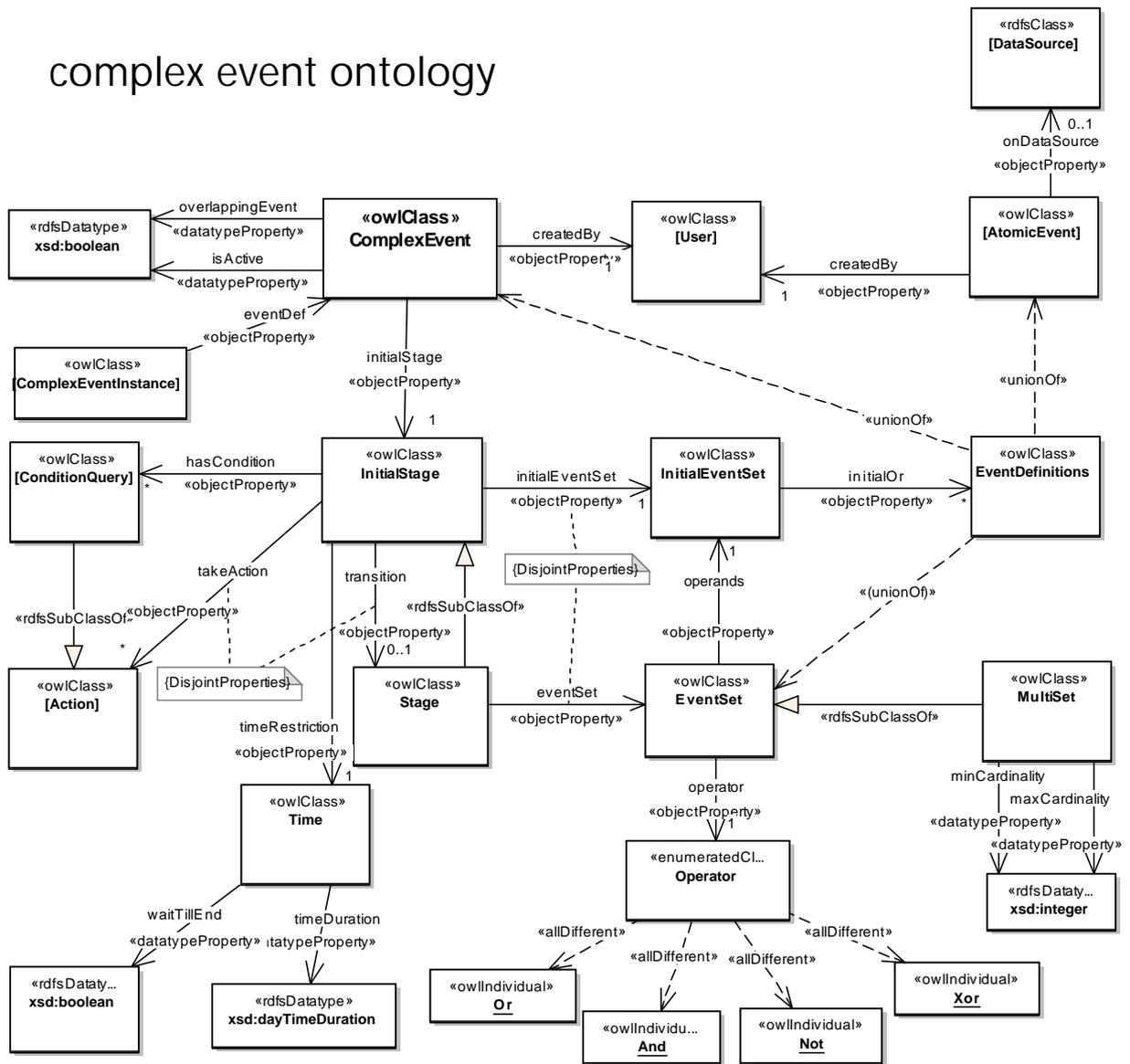


Abbildung 3: Complex-Event-Ontology

Listing 5: Definition der verwendeten Namespace-Abkürzungen

PREFIX shma :	<http://EventFramework/Schema/>	die Complex Event Ontology
PREFIX link :	<http://EventFramework/LinkedData/>	alle Daten via Linked-Data-View
PREFIX :	<http://EventFramework/Stages/>	alle Eventdefinitionen

Konzepte und Eigenschaften:

Anmerkung: Konzepte welche durch eckige Klammern markiert sind stellen ihre Instanzen via Linked-Data-Views zur Verfügung (siehe 5.1.). Konzepte sind durch einen initialen Großbuchstaben von Eigenschaften zu unterscheiden. Hier werden nur solche Eigenschaften und Konzepte beschrieben, die für das Verständnis dieser Ontologie entscheidend sind.

Action (Aktion - Kurznotation: A)

Die formale Definition der unter 5.1. beschriebenen Action. In diesem Framework werden Actions, Conditions und Queries im Allgemeinen auf Endpunkten unter Verwendung von SOAP-Nachrichten ausgeführt. Parameter, die für des Ausführen einer Action von Nöten sind, werden statisch bereitgestellt oder mittels verschiedener Queries auf SOAP- oder SPARQL-Endpunkten abgefragt. Actions sollen nur an die letzte Stage eines komplexen Events gekoppelt werden, um der Definition von ECA-Rules gerecht zu werden.

Atomic Event (atomare Event - Kurznotation: AE)

Die formale Definition des unter 5.1. beschriebenen atomaren Events.

ComplexEvent (komplexes Event - Kurznotation: CE)

Die formale Definition des unter 5.1. beschriebenen komplexen Events.

ConditionQuery (Condition oder Query - Kurznotation: C, Q)

Die formale Definition der unter 5.1. beschriebenen Condition. Hierbei werden für den Test einer Condition, alle nötigen Parameter statisch oder mittels verschiedener Queries bereitgestellt. Anschließend wird die Condition genau wie eine Action auf einem SOAP-Endpunkt ausgeführt und als Ergebnis ein Wahrheitswert erwartet.

(Anmerkung: Eine Condition ist formal eine Query welche als Rückgabewert einen Wahrheitswert verlangt. Condition und Query werden daher als ConditionQuery zusammengefasst)

EventInstance (Event-Instanz)

Wird ein komplexes Event aktiviert, so wird eine neue Instanz dieses komplexen Events angelegt, unabhängig davon ob alle Bedingungen für das Eintreten dieses Events erfüllt werden können. Eine Event-Instanz wird verworfen, sobald ein Event nicht wie vorgegeben eintritt oder eine Kondition nicht erfüllt werden kann.

EventSet (Kurznotation: ES)

Ein EventSet besteht aus einer Menge von Events oder weiteren EventSets, die als Operanden dienen und einem Boole'schen Operator aus der Menge: {OR, AND, XOR, NOT}. Standardmäßig wird jedes Event-Set mit einem OR initiiert. Alle enthaltenen Elemente dieses Event-Sets sind mit diesem Operator verknüpft. Durch die Schachtelung weiterer EventSets in anderen können komplexe Boole'sche Strukturen abgebildet werden und dienen der Anordnung von Events in einer Stage.

Listing 6: Beispiel: EventSet

```
XOR(AE1, AND(CE1, NOT(AE5)))
```

Das äußere Event-Set besteht aus dem atomaren Event AE1 und einem zweiten Event-Set, verbunden durch ein XOR. Das zweite Event-Set verbindet das komplexe Event CE1 und das Nicht-Eintreten vom atomaren Event AE5 mit einem AND und könnte wie folgt ausgedrückt werden: Entweder AE1 tritt ein, oder CE1 tritt ein während AE5 nicht eintreten darf.

InitialEventSet (Kurznotation: IES)

Ein InitialEventSet enthält nur atomare oder komplexe Events als Operanden. Der Operator ist stets ein OR.

InitialStage (Kurznotation: IS)

Die erste Stage eines jeden komplexen Events ist obligatorisch und muss ein Initial-Event-Set enthalten. Dadurch können Events der ersten Event-Stage nur durch ein OR verknüpft werden. Jedes einzelne Event in der ersten Phase eines komplexen Events beendet diese Phase und initiiert die zweite Phase (falls vorhanden). Es wird dadurch mindestens eine Stage-Transition erzwungen, welche den zeitlichen Verlauf eines komplexen Events genauer dokumentieren soll und eine Möglichkeit bietet initiale Konditionen abzufragen.

MultiSet (Kurznotation: MS)

Das MultiSet ist eine Sonderform des EventSet. Es enthält genau ein atomares oder komplexes Event als Operand und OR als Operator. Durch eine Minimal- und Maximalkardinalität ist es möglich die Anzahl des Auftretens des enthaltenen Events zu bestimmen. Ein so angegebenes Event muss in der aktuellen Stage x-mal auftreten, wobei gilt:

$$\text{Minimalcardinalität} \leq x \leq \text{Maximalcardinalität} \quad (1)$$

overlapping EventInstance (überlappende Event-Instanz)

Wird ein komplexes Event als 'overlapping' definiert, wird bei jeder Transition von der ersten in die zweite Stage eine neue EventInstance des gleichen komplexen Events angelegt. Zu jedem Zeitpunkt ist eine EventInstance in der InitialStage dieses Events aktiv.

Stage oder EventStage (Phase, Teilevent - Kurznotation: S)

Jedes komplexe Event setzt sich aus einer oder mehreren Stages zusammen. Sie unterteilen das Event in Zeitintervalle, in denen alle für diese Stage vorgesehenen Events eintreten müssen.

timeRestriction (Zeitrestriktion - Kurznotation: T)

Jede Stage kann optional mit einer Zeitrestriktion versehen werden. Diese besteht aus einer Zeitspanne und einem Boole'schen Wert, welcher angibt, ob die vorgegebene Zeitspanne unter allen Umständen einzuhalten ist (WaitTillEnd). Ist keine Zeitrestriktion für eine Stage definiert, kann diese Stage nicht nach dem Ablauf einer einer bestimmten Zeit beendet werden. Ist eine Zeitrestriktion für eine Stage definiert müssen alle im EventSet beschriebenen Events im vorgesehenen Zeitintervall eintreten. Die aktuelle Instanz eines komplexen Events wird verworfen falls die beschriebene Zeitspanne abgelaufen ist, ohne dass alle vorgesehenen Events eingetreten sind. Es kann vorkommen, dass alle Events, die zur Erfüllung dieser Phase eintreten müssen, bereits eingetreten sind. Nun ist noch zu kontrolliert, ob ein Event eintritt welches laut Event-Definition nicht eintreten darf.

Listing 7: Beispiel: EventSet mit Event das nicht eintreten darf

```
AND(AE1, CE2, NOT(AE3)) - AE1, CE2 sind eingetreten
```

Ist in diesem Fall die WaitTillEnd-Konstante mit dem Wert 'False' versehen, kann sofort mit der Kontrolle auf zusätzliche Konditionen fortgefahren werden und letztlich in die nächste Phase übergegangen werden. Andernfalls wird bis zum Ende der angegebenen Zeitspanne gewartet um sicher zu gehen, dass AE3 nicht eintritt.

transition (Stage-Transition - Kurznotation: $\rightarrow[C^*,T^?]-\rightarrow$)

Beschreibt den Übergang von einer Stage in die nächste unter der Voraussetzung, dass alle Bedingungen der Ausgangs-Stage (Events und Conditions) erfüllt sind und eine mögliche Zeitrestriktion nicht überschritten wurde.

5.3.2. Definition eines komplexen Events mit Hilfe von regulären Ausdrücken

Auf Grundlage der Event Ontology und den eingeführten Bezeichnungen (bzw. Kurznotationen) lassen sich komplexe Events mittels regulären Ausdrücken beschreiben. Dazu wird eine induktive Definition [?] der einzelnen Bestandteile vorgenommen:

Anmerkung zur Notation: Tupel werden mit eckigen Klammern umschlossen. Stagetransitionen werden mit $\rightarrow[C^,T^?]-\rightarrow$ dargestellt. Diese Notation verdeutlicht die möglichen Bedingungen unter denen eine Stagetransition eintreten darf: 1. alle Conditions (C^*) sind erfüllt, 2. falls eine Zeitrestriktion vorhanden ist ($T^?$), so ist diese zu berücksichtigen.*

5.3.2.1. Definition: InitialEventSet

- eine Menge von atomaren, bzw. komplexen Events oder weiteren EventSets (der Operator OR wird implizit angenommen)

$$IES := (AE|CE|ES)^+$$

5.3.2.2. Definition: EventSet

- ein Tupel aus einem InitialEventSet und einem Operator aus der Menge {OR, XOR, AND, NOT}

$$ES := [IES, (OR|XOR|AND|NOT)]$$

5.3.2.3. Definition: InitialEventStage

- ein Tupel aus einem InitialEventSet, einer Menge von Conditions, der nächsten Stage oder einer Menge aus Actions und optional einer Zeitrestriktion

$$IS := [IES, C^*, T^?, (S|A^*)]$$

5.3.2.4. Definition: EventStage

- ein Tupel aus einem EventSet, einer Menge von Conditions, der nächsten Stage oder einer Menge aus Actions und optional einer Zeitrestriktion

$$S := [ES, C^*, T^?, (S|A^*)]$$

5.3.2.5. Definition: ComplexEvent

Komplexe Events lassen sich unterteilen in aktive und passive Events. Der Übergang von einer Stage zu einer anderen wird mittels des Stage-Transition Operators $->[C^*,T?]->$ beschrieben.

passive komplexe Events:

$$CE := IS(- > [C^*, T?]- > S)^*$$

aktive komplexe Events:

$$CE := IS(- > [C^*, T?]- > S) * - > [C^*, T?]- > A^+$$

5.3.3. Definition eines komplexen Events am Beispiel

Zum besseren Verständnis der in der 'Complex Event Ontology' formal ausgedrückten Ideen wird ein Beispiel der Definition eines komplexen Events im RDF-Format präsentiert.

Listing 8: Beispiel: Definition eines komplexen Events in RDF (Auszug)

```

PREFIX shma: <http://EventFramework/Schema/>
PREFIX link: <http://EventFramework/LinkedData/>
PREFIX : <http://EventFramework/Stages/>

:InitialStage1      a                shma:InitialStage
:InitialStage1      shma:transition  :Stage5
:InitialStage1      shma:stageID    1
:InitialStage1      shma:initialEventSet  :InitialEventSet8

:InitialEventSet8   a                shma:InitialEventSet
:InitialEventSet8   shma:initialOr   link:AtomicEvent10
:InitialEventSet8   shma:initialOr   link:AtomicEvent5
:InitialEventSet8   shma:setId      8

:Stage5             a                shma:Stage
:Stage5             shma:stageID    5
:Stage5             shma:takeAction  link:Action7
:Stage5             shma:timeRestriction  nodeID://b10118
:Stage5             shma:initialEventSet  :EventSet9

:EventSet9          a                shma:EventSet
:EventSet9          shma:operands   :InitialEventSet9
:EventSet9          shma:operator   shma:Xor
:EventSet9          shma:setId      9

:InitialEventSet9   a                shma:InitialEventSet
:InitialEventSet9   shma:initialOr  :EventSet10
:InitialEventSet9   shma:initialOr  link:ComplexEvent2
:InitialEventSet9   shma:setId      9

nodeID://b10118    a                shma:Time
nodeID://b10118    shma:timeDuration  3000
nodeID://b10118    shma:waitTillEnd  0

```

1. **InitialStage1** Die erste Stage dieses Events. Alle Events dieser Stage sind unter :InitialEventSet8 zu finden, die nachfolgende Stage wird mit einer Stage-Transition zu Stage5 initiiert(durch Eigenschaft shma:transition ausgedrückt).
2. **InitialEventSet8** InitialStage1 wird durch AtomicEvent5 oder AtomicEvent10 erfüllt (Eigenschaft shma:initialOr bindet alle Events des InitialEventSets).

3. **Stage5** - Nach erfolgreichem Eintreten aller verlangten Events aus EventSet9, in der Zeitspanne beschrieben unter nodeID://b10118 (shma:timeRestriction), wird Action7 (shma:takeAction) ausgeführt.
4. **EventSet9** beschreibt eine Menge von Events in InitialEventSet9 (shma:operands), welche durch den Operator Xor (shma:operator) verknüpft werden und in diesem Beispiel alle Events der Stage5 darstellen.
5. **InitialEventSet9** - Diese Eventmenge besteht aus dem komplexen Event ComplexEvent2 (d.h. diese komplexe Event muss in der vorgegebenen Zeitspanne von Stage5 auftreten) und allen weiteren Events im EventSet10 (hier nicht aufgeführt).
6. **nodeID://b10118** - Dieser Blank-Node ist eine Instanz des Konzepts Time. Es stehen 3000 Sekunden zur Verfügung die verknüpfte Stage (Stage5) abzuschließen (alle Events müssen eingetreten und alle Conditions erfüllt sein). Sind alle Events erfüllt, dann wird mit nicht gewartet bis 3000 Sekunden vergangen sind, sondern sofort mit der Auswertung von Conditions begonnen (shma:waitTillEnd = 0, also 'false')

5.4. Service-Layer

Das Service-Layer des Frameworks unterteilt sich in einen serverseitigen und einen clientseitigen Service. Der Event-Service, der auf einem Server parallel zur Zentraldatenbank ausgeführt wird, dient als Schnittstelle für die Kommunikation mit dem Client-Service und übernimmt zusätzlich alle internen Funktionen, welche nicht direkt über Datenbankprozeduren umgesetzt werden können. Der Client-Service stellt die Verbindung zwischen Client-Applikation und Event-Service her. Die Datenaustausch zwischen diesen beiden Kommunikationspartnern erfolgt mittels SOAP-Nachrichten über mögliche Systemgrenzen hinweg. Wie in solchen verteilten Umgebungen üblich werden die durch Services veröffentlichten Funktionen und Methoden, sowie die zum Datenaustausch verwendeten Datenkonstrukten über Kommunikationsverträge (Communication-Contracts) definiert, welche dem Event-Service, wie dem Client-Service in der aktuellen Version zur Verfügung stehen müssen. Üblicherweise werden Service-Contracts zur Definition von Schnittstellen und Data-Contracts zur Definition der ausgetauschten Datenobjekte eingesetzt.

5.4.1. Event-Service

Dieser serverseitige Service wird parallel zur Zentraldatenbank eingesetzt hat 3 Hauptaufgaben:

1. Veröffentlichung aller Methoden und Funktionen die Clientanwendungen für die Nutzung des Event-Frameworks benötigen und zentrale Schnittstelle für die Kommunikation mit jedem Client-Service
2. Datenbankkommunikation zum Aufruf von Prozeduren und der Abfrage von Daten
3. Erweiterung der Funktionalität der Zentraldatenbank

zu 1.

Der Event-Service ist der direkte Kommunikationspartner für jeden Client-Service und damit indirekt für jede Anwendung die auf Informationen dieses Frameworks zugreifen will. Dabei wird eine Kommunikation stets vom Client-Service per SOAP-Request-Nachricht angeregt und vom Event-Service nach der Ausführung der angeforderten Funktion oder Methode mit einer SOAP-Response-Nachricht, welche die angeforderten Informationen enthält, beantwortet.

Die angebotene Funktionalität des Services für Clientanwendungen lässt sich in die folgenden Teilbereiche gliedern:

- a) Benutzerverwaltung und Benutzersessions (Log in von Benutzern, Hinzufügen von Nutzern...)
- b) Verwaltung von Satellitendatenbanken, Event- und Datenquellen (Satellitendatenbank registrieren...)
- c) Bereitstellung von Informationen (z. B. welche Events in einer bestimmten Zeitspanne aufgetreten sind...)
- d) Erstellung von Actions und Conditions
- e) Definition von atomaren und komplexen Events
- f) Weitere Funktionen die speziell für bestimmte Programme erstellt werden

zu 2.

Die Verbindung zwischen Event-Service und Zentraldatenbank wird über eine ODBC-Verbindung hergestellt [35]. Der Event-Service kommuniziert daher ausschließlich mittels SQL-Kommandos mit der Zentraldatenbank.

zu 3.

Die Zentraldatenbank löst die meisten Aufgaben zur Abwicklung und Verwaltung von Events automatisch mittels Datenbankprozeduren und Trigger. Einige wenige Aufgaben, wie das Ausführen von Actions und Conditions, müssen aber aus Gründen der Komplexität in den Event-Service ausgelagert werden. Hierzu greift die Zentraldatenbank mit einer SOAP-Nachricht auf die entsprechende Methode des Event-Service zu.

5.4.2. Client-Service

Dieser clientseitige Service muss für die Nutzung des Frameworks durch eine Anwendung auf dem selben Computer ausgeführt werden und soll eine einheitliche Schnittstelle für die Kommunikation mit dem Event-Service bieten und die Anmeldung von Benutzern beim Event-Service steuern. Die Kommunikation zwischen Client-Service und Anwendungen soll über IPC-Protokolle (IPC; [36] z. B. Net-Pipe [37]) erfolgen. IPC stellt sicher, dass sich beide Kommunikationsendpunkte auf dem selben Computer befinden und eignet sich daher für einen schnellen und sicheren Austausch von Daten innerhalb eines Computersystems.

5.5. Application-Layer

Die Applikationsebene umfasst alle Programme, die direkt auf die Funktionen und Methoden des Client-Service zugreifen. Das im Zuge dieser Arbeit implementierte Framework soll neben den Datenbanken, Event- und Client-Service zusätzlich eine administrative Applikation bereitstellen (Event-Framework-Control siehe 5.2.3.1), welche die komplette Definition von atomaren und komplexen Events, Event- und Datenquellen, Aktionen und Konditionen sowie die Verwaltung von Benutzern des Frameworks erlaubt. Jede weitere Applikation die zur Lösung der beschriebenen Probleme der Anwendungsfälle (vergleiche Kapitel 3.1.) eingesetzt wird, greift auf die Funktionalität zurück, die über den Client-Service zur Verfügung gestellt wird. Im Kapitel Evaluation (Kapitel 7.) wird noch einmal näher auf die Möglichkeiten weiterer Applikationen im Event-Framework eingegangen.

5.5.1. Event-Framework-Control

Das Event-Framework-Control ist eine administrative Anwendung zur Steuerung des Event-Framework und zur Verwaltung von Benutzern und deren Rechte. Die wichtigsten Aufgabebereiche dieser Anwendung sind:

1. Benutzer- und Rechteverwaltung
2. Verwaltung von Satellitendatenbanken, Event- und Datenquellen
3. Definition von atomaren Events auf Satellitendatenbanken
4. Registrierung atomarer Events externer Eventquellen
5. Erstellung/Registrierung von Actions, Conditions und Queries
6. Zuteilung von Parametern zur Ausführung von Actions, Conditions und Queries
7. Definition von komplexen Events aus Events, Actions, Conditions und Queries
8. (De-)Aktivierung und Kontrolle von Instanzen komplexer Events

Über ein einfaches Rechtesystem mit Administratoren und Benutzern mit verminderten Rechteumfang soll der Zugriff und auf Event- und Datenquellen beschränkt werden. Eine Benutzeroberfläche, die durch einfache Bedienung und Übersichtlichkeit auch Laien einen Zugang zum Thema komplexe Events ermöglicht, ist anzustreben.

6. Implementierung

Im Rahmen dieser Arbeit wurde ein komplettes Event-Framework nach den Spezifikationen unter 5. implementiert und steht in einer Open-Source Version zur Verfügung (Link: <https://github.com/AKSW/VirtuosoEventFramework/>). In diesem Kapitel wird diese Implementierung, beginnend mit dem Data-Layer, beschrieben und auf grundlegende Entwurfsentscheidungen näher eingegangen.

Für die gesamte Entwicklung dieses Frameworks wird Virtuoso als Datenbanksystem in der Version 6.1.7 eingesetzt, sowohl als Zentraldatenbank, als auch als Beispiel für Satellitendatenbanken. Virtuoso bietet einen integrierten Triple-Store basierend auf einer Quad-Tabelle [38]. Damit genügt Virtuoso den zuvor gestellten Anforderungen an ein Datenbanksystem (vergleiche Kapitel 5.2.). Es wurde versucht, den Großteil der internen Funktionalität des Frameworks als SQL-Prozeduren direkt in den verwendeten Datenbanken zu implementieren, um Zeitverluste, die bei der Kommunikation mit dem Event-Service auftreten, zu umgehen. Bis auf die Ausführung von Actions und Conditions, ist dies umgesetzt worden. Da das Ausführen einer Action (oder Condition) das dynamische Erstellen und Kompilieren eines eigenständigen SOAP-Clients verlangt, konnte diese Aufgabe nicht als SQL-Prozedur umgesetzt werden.

Für die Implementierung der Bestandteile von Service- und Applikationsschicht wurde unter der Verwendung der Programmiersprache C# auf der plattformübergreifenden CLR-Implementierung des Mono-Projekts [39] zurückgegriffen. Diese Maßnahme erlaubt zum jetzigen Zeitpunkt den Einsatz von Event- und Client-Service unter Linux-Distributionen und Windows. Der Einsatz der Event-Framework-Control Applikation unter Linux scheitert zur Zeit noch an verschiedenen Unstimmigkeiten zwischen den Versionen von Mono auf Linux und Windows.

Zur Veranschaulichung der Verarbeitung von Events im Event-Framework, beginnend mit einem atomaren Events auf einer Satellitendatenbank, wird ein Beispiel bemüht. Anhand dieses Beispiels wird der Kontrollfluss im Framework bis hin zur Vollendung eines komplexen Events und der anschließenden Ausführung von vordefinierten Aktionen genau beleuchtet. Dies soll einen genaueren Einblick in die automatische Prozesse geben, welche durch das Auftreten eines Events angeregt werden. Im Rahmen des schriftlichen Teils dieser Bachelorarbeit wird bewusst auf eine ausführliche Dokumentation aller Prozesse, Prozeduren, Datenbanktabellen, Funktionen der Serviceebene und den Aufbau der Event-Framework-Control Applikation verzichtet. Da der Umfang des implementierten Frameworks den Arbeitsaufwand einer Bachelorarbeit bereits deutlich überzogen hat, würde eine derartige Beschreibung zusätzlich den vorgegebenen Rahmen der schriftlichen Ausarbeitung weit überschreiten. Außerdem ist die veröffentlichte Version zum Zeitpunkt der Fertigstellung dieser Arbeit noch keineswegs final. Eine ausführliche Dokumentation wird im Zuge der Fortführung dieser Implementierung begleitend zum Source-Code veröffentlicht (<https://github.com/AKSW/VirtuosoEventFramework/>).

Anmerkung zum Pseudocode: Der in diesem Kapitel verwendete Pseudocode soll unübersichtlich SQL-Prozeduren vereinfacht zusammenfassen. Dabei wird versucht sich an der ursprünglichen SQL-Syntax zu orientieren. Der Einfachheit halber wird von dieser Praxis teilweise abgesehen. Außerdem müssen einige Teile in Kommentaren zusammengefasst werden. Alle genannten Tabellen werden in Appendix A und B beschrieben.

6.1. Beispielszenario

Das hier verwendete Szenario sollte dem Leser als Anwendungsfall AF4 aus Kapitel 3.1. bekannt sein. Es werden verschiedene Lösungsansätze präsentiert, die den Umgang mit einem solchen Szenario unter herkömmlichen Voraussetzungen beschreiben und einem Lösungsansatz unter Verwendung des Event-Frameworks gegenübergestellt.

In einem Onlineshop sollen Veränderungen an Benutzerkontodaten automatisch überwacht werden um auf auffällige Aktivitäten rechtzeitig und angemessen reagieren zu können. Eine Vorkehrung besteht darin Änderungen an Passwörtern und E-Mail-Adressen eines Kontos zu registrieren und auf ungewöhnliche Änderungsaktivitäten zu achten. Werden Passwort und Adresse innerhalb weniger Tage geändert, könnte dies auf einen möglichen Eingriff eines unbefugten Akteurs hinweisen, der an das Passwort des Kontoinhabers gelangt ist und nun versucht das Konto ganz unter seine Kontrolle zu bringen. wird von Seiten des Anbieters die Vorsichtsmaßnahme getroffen, alle Nachrichten über die Änderung von Passwörtern an alle von diesem Nutzer bekannten E-Mail-Adressen zu senden, auch wenn diese vor weniger als einem Monat vom Nutzer gelöscht wurden. Damit würde im oben beschriebenen Betrugsszenario der richtige Nutzer von den Vorgängen auf seinem Benutzerkonto informiert.

Es wird angenommen, dass die Datenbasis dieses Szenarios in Form von RDF-Tripeln zur Verfügung steht. Das Vorgehen bei Definition und Auswertung von atomaren Events auf Grundlage von Datenbanktabellen ist sehr ähnlich, da der Triple-Store von Virtuoso direkt auf einer Datenbanktabelle beruht. Aus Zeitgründen wird auf diese Möglichkeit aber nur oberflächlich eingegangen.

6.1.1. Lösungsansatz mit manuell definierten Datenbanktriggern

Das beschriebene Szenario lässt sich nur mit einem komplexen Event beschreiben. Ein Versuch, dieses Verhalten mit Hilfe eines einfachen Datenbanktriggers zu modellieren (und somit als atomares Event), würde die folgenden Probleme aufwerfen:

Eine Aktion, welche mehrere E-Mails versendet, ließe sich mit einem einzelnen Trigger umsetzen. Allerdings müssten neben den zur Zeit hinterlegten E-Mail-Adressen eines Benutzerkontos alle Adressen, die in den letzten 30 Tagen gelöscht wurden, ausfindig gemacht werden. Dies würde eine gesonderten Speicherort für gelöschte Adressen voraussetzen. Außerdem müsste für das Löschen von E-Mail-Adressen ein weiterer Datenbanktrigger definiert werden, welcher die gerade gelöschten Adressen an diesen Ort abspeichert. Dieses Vorgehen entspricht einem aufwendig implementierten komplexen Events, welches eine Sonderlösung für ein spezielles Problem darstellt und keinerlei Wiederverwendbarkeit bietet. Jede Änderung dieses Events wäre mit hohen Kosten verbunden. Ähnliche Probleme existieren beim Einsatz von CEP-Systemen wie Odysseus (vergleiche Kapitel 4.1.). Beide beschriebenen atomaren Events müssen als Trigger definiert werden und zusätzlich als Event-Stream zur Verfügung gestellt werden.

6.1.2. Lösungsansatz auf Basis eines Triple-Stores

Die meisten Triple-Stores bieten keinerlei reaktive Funktionalität und eignen sich daher nicht zur Lösung des beschriebenen Problems. Greift man z. B. auf einen Sesame Triple-Store mit entsprechendem Storage and Interface Layer (SAIL) zurück (vergleiche Kapitel 2.4.), müssen atomare Events direkt über die SAIL-API registriert werden. Ein reaktiver Lösungsansatz ist daher nur sehr begrenzt möglich und mit hohen Kosten bei der Umsetzung verbunden. Der Einsatz dieses Triple-Stores in Verbindung mit dem beschriebenen Framework wäre allerdings durchaus denkbar.

6.1.3. Lösungsansatz mit dem Event-Framework

Wie durch Benutzeranforderung BA9 (freie Wahl und Definition von Events und Ressourcen) gefordert, kann man mit dem Event-Framework unabhängig von der zugrundeliegenden Datenstruktur atomare Events definieren. Da eine Virtuoso-Datenbank Daten sowohl in Form von Tabellen, als auch in RDF-Triplen speichern kann, wird bei der Implementierung des Frameworks auf Instanzen dieses Datenbanksystems zurückgegriffen. Die Schritte zur Definition eines komplexen Events, wie es als Beispielszenario beschrieben wurde, lassen sich grob zusammenfassen:

1. Definition eines atomaren Events, welches das Update eines Passworts registriert (AE1)
2. Definition eines atomaren Events, welches das Löschen / Update einer E-Mail-Adresse registriert und als optionaler Wert die gerade gelöschte E-Mail-Adresse speichert (AE2)
3. Definition der Aktion 'Emails senden an' (ACT1)
4. Definition eines komplexen Events (CE1) nach erweiterten ECA-Regeln:
 - a) Initial-Stage: Initial-Event-Set: {AE2} Eine Instanz dieses Events wird durch das Löschen einer E-Mail-Adresse ausgelöst und tritt danach in die nächste Phase (Stage 1) ein.
 - b) Stage 1, Event-Set: {AE1}, Zeitrestriktion – 30 Tage, Aktionen {ACT1}: Innerhalb der nächsten 30 Tage wird mit jeder Änderung des Nutzerpasswortes diese Phase beendet. Da es die letzte Phase des Events ist, würde damit das komplexe Event erfüllt und die Action {ACT1} ausgeführt werden. Die nötigen Parameter, wie E-Mail-Adressen, sind durch entsprechende Queries zu erlangen. Die bereits gelöschte Adresse wird mittels eines Event-Value-Mappings (vergleiche Kapitel 6.4.1.1.) aus dem atomaren Event AE2 extrahiert.
5. Aktivierung des komplexen Events CE1

Vor der Ausführung der beschriebenen Aktion könnten noch zusätzliche Bedingungen über das Ausführen von Conditions geprüft werden. Man könnte z. B. kontrollieren, ob der betreffende Nutzer zur Gruppe der Administratoren gehört. In diesem Fall könnte das Passwort automatisch wieder auf das ursprüngliche Passwort gesetzt werden.

Es wird davon ausgegangen, dass alle Schritte zur Definition des komplexen Events korrekt ausgeführt würden und dass alle nötigen Daten in der unter 5.2.1.1. beschriebenen Datenstruktur vorliegen.

6.2. Satellitendatenbanken und atomare Events

Das Event-Framework bietet zwei Möglichkeiten atomare Events zu definieren. Man kann ein atomares Event mit Hilfe eines eindeutigen Namens registrieren. In diesem Fall wird einfach eine neue ID für dieses Event vergeben und die erwartete Form der SOAP-Nachricht, welche dieses Event auf der Zentraldatenbank als eingetreten meldet, angezeigt. Die Implementierung und Auslösung, sowie die Übertragung der SOAP-Nachricht, liegt dann allein in den Händen des Event-Providers.

Die zweite Möglichkeit ist das Erstellen eines atomaren Events auf einer Satellitendatenbank mit Hilfe des Event-Framework-Control. Hierbei kann der User zwischen einem atomaren Event auf einer beliebigen Datenbanktabelle, oder für den in Virtuoso integrierten Triple-Store entscheiden. Je nach Trigger-Typ (INSERT, DELETE, UPDATE) existieren drei verschiedene Trigger, die bei Bedarf für die gewählte Tabelle angepasst und eingefügt werden.

Im Folgenden wird das Auftreten und die Verarbeitung von atomaren Events auf RDF-Metadaten der Satellitendatenbanken demonstriert.

6.2.1. Definition atomarer Events

Benutzer des Event-Frameworks haben die Möglichkeit, mit dem Event-Framework-Control atomare Events auf Datenbanktabellen und dem integrieren Triple-Store einer Virtuoso-Datenbank, zu definieren. Dazu wird im ersten Fall vom Benutzer die Ergänzung des WHERE-Teils einer einfachen SELECT-FROM-WHERE-Anfrage in SQL erwartet, welche genau die Tupel herausfiltern soll, die in Zukunft das atomare Event auslösen sollen. Außerdem muss eine Trigger-Option (INSERT, UPDATE, DELETE) ausgewählt werden. Wird ein solches Event ausgelöst, werden alle Werte dieses Tupels in einem 'Row-Vektor' gespeichert und den Informationen über dieses Event angefügt. Diese Werte kann man z. B. als Parameter für Actions oder Conditions wieder verwenden (vergleiche Kapitel 6.6.1.1. - Parametermapping).

Listing 9: Beispiel Kondition eines atomaren Events in EventFrameworkTriggerConditions

TriggerName	TriggerType	TableName	ParamArray	Condition
DELETETRIGGER_1_13	DELETE	A_TEST_TABLE	0,2	SELECT CASE WHEN (? = 88 AND ? < 100) THEN 1 ELSE 0 END

Das Beispiel zeigt ein einfaches atomares Event auf der Tabelle A_TEST_TABLE. Die Spalte 'ParamArray' gibt an, welche Werte des Row-Vectors an Stelle des '?' der parametrisierten Kondition einzusetzen sind.

Soll dagegen ein atomares Event auf dem Virtuoso-Triple-Store definiert werden, erfolgt dies mit der Ergänzung des WHERE-Teils einer SPARQL-Anfrage durch einen Benutzer. Als 'Row-Vektor' wird der komplette Quad gespeichert. In beiden Fällen wird die so definierte Bedingung für das neue atomare Event zuerst parametrisiert und dann in der Tabelle 'EventFrameworkTriggerConditions', der jeweiligen Satellitendatenbank, abgespeichert. Da SPARQL bei Virtuoso auf SQL aufbaut, ist es möglich, SPARQL-Konstrukte direkt als Kondition in SQL-Anfragen zu übernehmen. Eine Übersetzung von SPARQL nach SQL ist daher nicht erforderlich. Damit besteht eine einfache Möglichkeit atomare Events auf der Basis von RDF-Metadaten zu definieren. Benutzeranforderung BA11 (vgl. Kapitel 3.2.) ist im Bezug auf verwendete Virtuoso-Datenbanken erfüllt.

6.2.2. Auslösen atomarer Events

Wir nehmen an, dass in dem betrachteten Beispiel der folgende Triple soeben gelöscht wurde:

```
<http://testGraph/UserID00023> <http://testGraph/hasEmail> "noRealEmail@testGraph.net"
```

D.h. die E-Mail-Adresse noRealEmail@testGraph.net des Benutzers mit ID 23 wurde gelöscht. Durch die Definition des atomaren Events AE2, der einen solchen Vorgang als atomares Event meldet, wurde der folgende Trigger der RDF_QUAD-Tabelle hinzugefügt. (*Anmerkung: Dieser Trigger gilt für alle atomaren Events des Typs 'DELETE' auf dieser Tabelle. Es werden pro Tabelle maximal drei Trigger benötigt - INSERT, DELETE, UPDATE*).

- 1: TRIPLE_DELETE_TRIGGER BEFORE DELETE ON RDF_QUAD REFERENCING OLD AS O
- 2: trigg := (SELECT TriggerName, Condition FROM EventFrameworkTriggerConditions WHERE TableName = 'VirtuosoTripleStore' AND TriggerType = 'DELETE');
- 3: rowVector := vector(O.Graph,O.Subject,O.Predicate,O.Object);
- 4: aq := async_queue (20); ▷ neue async_queue für Multithread-Aufgaben
- 5: **for all** $i \in trigg$ **do**
 - ▷ Jede gefundene Bedingung wird nun in einem neuen Thread mit den Werten des Row-Vektors überprüft.
- 6: aq_request (aq, 'CHECK_TRIGGER_CONDITION', vector ('VirtuosoTripleStore', now(), trigg[i][0], trigg[i][1], rowVector, 0));
- 7: **end for**

RDF_QUAD ist die Basistabelle des integrierten Triple-Stores in Virtuoso und beinhaltet alle Quads (Graph, Subjekt, Prädikat, Objekt - vergleiche Kapitel 2.1.), die in diesem Triple-Store hinterlegt werden. Entscheidend ist, dass dieser Trigger für alle atomaren Delete-Events dieses Triple-Stores anwendbar ist. Dieser Trigger wird beim Entfernen eines Triples aus dem Virtuoso-Triple-Store ein mal ausgeführt und löst das Überprüfen aller atomaren Events aus, die als DELETE-Events dieses Triple-Stores definiert worden. D.h. jedes gelöschte Tripel wird mit Hilfe vieler paralleler Threads auf das Zutreffen einer (oder mehrerer) vordefinierten Bedingung geprüft, welche in der Tabelle 'EventFrameworkTriggerConditions' auf der jeweiligen Satellitendatenbank hinterlegt sind. Der Vorteil ist, dass beim Löschen, Einfügen oder Update jedes Triples maximal ein Trigger ausgelöst wird. Das Auslagern der Kontrolle auf das eigentliche Eintreten eines atomaren Events (hier als Methode

'CHECK_TRIGGER_CONDITION') erlaubt das Verteilen dieser Aufgabe auf mehrere Threads. Dadurch wird die Zeitverzögerungen beim Umgang mit vielen atomaren Events (in Form von Datenbanktriggern) auf einer Tabelle reduziert.

6.2.3. Evaluation und Signalisierung eines atomaren Events

In diesem Schritt wird mit Hilfe der Methode 'CHECK_TRIGGER_CONDITION' überprüft, ob der eingefügte Tripel die Bedingungen eines atomaren Events erfüllt. Die Bedingung für das im Beispiel angedachte atomare Event AE2 könnte wie folgt aussehen:

Listing 10: SPARQL-Abfrage als Bedingung eines atomaren Events

```
sparql
PREFIX shma: <http://EventFramework/Schema/>
PREFIX : <http://EventFramework/Stages/>
ASK
FROM <http://EventFramework/Stages>
WHERE
{
graph ?graph
{
?subj ?pred ?obj.
?subj a shma:User.
FILTER(?pred = shma:hasEmail)
}
FILTER(?graph IN (<http://EventFramework/Stages>))
FILTER(str(?subj) = ??)
FILTER(str(?pred) = ??)
FILTER(str(?obj) = ??)
}
```

Das Muster ?subj ?pred ?obj ist in jeder Kondition atomarer Events vertreten, um die gesuchte Tripelstruktur festzulegen. Durch die Filter auf ?subj, ?pred und ?obj wird das gesuchte Triple eindeutig auf das soeben gelöschte Triple festgelegt, indem die Parameter '??' durch die Werte des gelöschten Triples ersetzt werden.

Jedes atomare Event, das so erkannt wird, löst das Versenden einer SOAP-Nachricht an die Zentraldatenbank aus, welche neben einer Identifikation des Events, mit dem übertragenen Row-Vektor alle Daten des auslösenden Tupels liefert. Diese Prozedur wird, wie im vorhergehenden Kapitel beschrieben, mehrmals parallel ausgeführt, falls mehrere atomare Events gleichen Typs auf dieser Tabelle existieren.

Parameter: tableNa - Tabellename, occurrence - Zeitpunkt des Auftretens, triggerNa - Name des atomaren Events, triggerCondition - Syntax der Kondition, rowVector - Row-Vektor-Werte

- 1: **procedure** CHECK_TRIGGER_CONDITION
- 2: (tableNa VARCHAR, occurrence DATETIME, triggerNa VARCHAR, triggerCondition VARCHAR, rowVector ANY)
- 3: params := getParams(rowVector); ▷ Parameter werden extrahiert
- 4: eval := EXECUTE(triggerCondition, params); ▷ die Kondition wird mit den Werte des auslösenden Triples überprüft
- 5: **if** eval = 1 **then** ▷ Kondition wurde erfüllt
- 6: resp := SEND_EVENT_AS_SOAP(vector(instance, tableNa, triggerNa, occurrence, rowVector), endpoint);

```

    ▷ Nachricht über das Eintreten eines atomaren Events wird an den SOAP-Endpunkt
      der Zentraldatenbank gesendet
7:      if resp = 1 then                                ▷ SOAP-Nachricht wurde erfolgreich übertragen
8:          INSERT INTO EventFrameworkAtomicEvents VALUES(1, occurrence, trig-
              gerNa, tableNa, rowVector);
9:      else                                             ▷ keine erfolgreiche Übertragung
10:         INSERT INTO EventFrameworkAtomicEvents VALUES(0, occurrence, trig-
              gerNa, tableNa, rowVector);
11:     end if
12: end if
13: end procedure

```

Die unter EventFrameworkAtomicEvents eingefügten Tupel legen einerseits ein Log aller aufgetretenen atomaren Events einer Satellitendatenbank an und können andererseits im Fall des Misserfolgs (erster Wert ist '0') zu einem späteren Zeitpunkt erneut übertragen werden.

Die Funktion 'SEND_EVENT_AS_SOAP' leitet den erstellten Parametervektor zusammen mit den Informationen über Auftrittsort und Name des atomaren Events an die Zentraldatenbank weiter, welche über den SOAP-Endpunkt mit der dort veröffentlichten Funktion 'INSERT_NEW_EVENT' angenommen und weiter verarbeitet werden.

6.3. Zentraldatenbank und komplexe Events

In diesem Abschnitt wird auf die Funktionalität der Zentraldatenbank im Bezug auf die Verarbeitung von Events genau eingegangen. Besonders die Erkennung von komplexen Events aus dem Strom eingehender atomarer Events, bis hin zur Ausführung von Actions wird genau betrachtet.

6.3.1. Aktivierung komplexer Events

Ist ein komplexes Event mit dem Event-Framework-Control definiert worden, kann es jederzeit aktiviert bzw. deaktiviert werden. Wird ein komplexes Event aktiviert, wird eine neue Instanz dieses Events angelegt.

Anmerkung: Für genauere Informationen über einzelne Tabellen konsultieren Sie bitte Appendix I.

Listing 11: Beispiel: neue Eventinstanz in Tabelle EventFrameworkComplexEventInstances

EventID	CEID	EventUri	FirstStageUri	CurrentStage	Started	Finished
25	33	link:ComplexEvent1	:InitialStage1	0	2013-12-08	<DB-NULL>

Darüber hinaus werden in die Tabelle 'EventFrameworkAwaitingEvent' (Eventwarteschlange) alle Events eingetragen, welche in der aktuellen Stage (nach der Aktivierung die InitialStage) von Bedeutung sind. Im Beispiel würden AE2 als atomares Event für die InitialStage und AE1 nach Aktivierung von Stage1 eingetragen werden.

Listing 12: Beispiel: Events in Tabelle EventFrameworkAwaitingEvent

CEID	TriggerID	SourceCE	Recurrences	Started	Until
<DB NULL>	2	25	0	2013-12-01	<DB NULL>

Tritt nun ein Event der gelisteten Events ein, so wird der Recurrence-Wert aller Einträge um eins erhöht, bei denen der Until-Wert größer ist als der aktuelle Zeitpunkt. Darf ein bestimmtes Event nicht eintreten, so muss der Recurrence-Wert dieses Events '0' sein. An Hand dieser Tabelle lässt sich schnell bestimmen, ob und wie oft ein bestimmtes Event seit der Initiierung einer Stage aufgetreten ist.

6.3.2. Empfangen von Events

Events werden in der Regel als SOAP-Nachrichten von einer Eventquelle an die Zentraldatenbank signalisiert. Eventquellen können neben Satellitendatenbanken andere externe Quellen sein (jede Netzwerkentität welche eine SOAP-Nachricht versenden kann). Dabei kann jede Instanz atomare sowie komplexe Events versenden. Da komplexe Events in der vorliegenden Version nur über eine Zentraldatenbank evaluiert werden, ist die Quelle für komplexe Events in der Regel die Zentraldatenbank selber (in diesem Fall kann auf das Senden einer SOAP-Nachricht natürlich verzichtet werden). In dem von uns betrachteten Beispiel aus 6.1. wird dieser Vorgang anhand eines atomaren Events nachvollzogen. Das Empfangen von komplexen Events läuft aber in exakt gleicher Weise ab.

Die durch eine Satellitendatenbank mit dem Aufruf der Funktion SEND_EVENT_AS_SOAP initiierte SOAP-Nachricht (vergleiche Kapitel 6.2.3.) hat in dem betrachteten Beispiel die folgende Form:

Listing 13: SOAP-Request-Nachricht an die Zentraldatenbank ausgelöst durch das Eintreten eines atomaren Events

```
<?xml version='1.0' ?>
<SOAP:Envelope
  xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/1999/XMLSchema'
  xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'
  xmlns:dt='urn:schemas-microsoft-com:datatypes'>
  <SOAP:Body>
    <cli:INSERT_NEW_EVENT xmlns:cli='services.wsdl'>
      <dsInstance>2</dsInstance>
      <internalSource>VirtuosoTripleStore</internalSource>
      <name>DELETETRIGGER_1_13</name>
      <occurrence>2013-12-13T23:01:22.000002+01:00</occurrence>
      <rowVector>
        <variant>http://testGraph</variant>
        <variant>http://testGraph/UserID00023</variant>
        <variant>http://testGraph/hasEmail</variant>
        <variant>"noRealEmail@testGraph.net"</variant>
      </rowVector>
    </cli:INSERT_NEW_EVENT>
  </SOAP:Body>
</SOAP:Envelope>
```

Der Row-Vektor fasst, wie bereits beschrieben, das auslösende Triple (bzw. Quad) dieses atomaren Events. Die anderen Parameter für die Funktion INSERT_NEW_EVENT sind: dsInstance - die ID der Datenquelle, die Datenbank-interne Bezeichnung des Ursprungsorts (in diesem Fall der Triple-Store), name - der Name des atomaren Events, occurrence - der genaue Zeitpunkt des Auftretens. Die Funktion lässt sich verkürzt so darstellen:

```

1: procedure INSERT_NEW_EVENT(dsInstance INT, internalSource VARCHAR,
    name VARCHAR, occurrence DATETIME, rowVector ANY)
2:   triggerID := (SELECT triggerID FROM EventFrameworkTriggers WHERE trigger-
    Name = name OR AlternativeName = name);
    ▷ bestimme ID des Events an Hand des Namens
3:   if triggerID IS NOT NULL AND triggerID > 0 then
4:     INSERT INTO EventFrameworkEvents VALUES(dsInstance, internalSource,
    triggerID, occurrence, rowVector);
    ▷ Abspeichern des Events in zentraler Event-Tabelle
5:     UPDATE_EVENTS(0, triggerID, id, occurrence);    ▷ siehe nächstes Kapitel
6:   end if
7: end procedure

```

Das Auftreten des empfangenen Events wird in der Tabelle 'EventFrameworkEvents' vermerkt. Die Tabelle 'EventFrameworkEvents' dient als Eventhistorie, da alle auftretenden Events hier gespeichert werden. So ist es möglich auch nachträglich komplexe Events zu erkennen. Mit dem Aufruf der Methode UPDATE_EVENTS wird die Evaluation von komplexen Events eingeleitet.

6.3.3. Evaluation komplexer Events

Die Evaluation von komplexen Events wird stets gleichzeitig für alle aktiven Eventinstanzen abgewickelt. Wird ein Event empfangen, das von mindestens einer Instanz in der Tabelle 'EventFrameworkAwaitingEvent' hinterlegt wurde, muss kontrolliert werden, ob durch das Eintreten dieses Events der Status der aktuelle Stage verändert wird. Wird eine Stage abgeschlossen, müssen etwaige Conditions geprüft werden und schließlich eine neue Stage initiiert werden. Handelt es sich um die letzte Stage müssen definierte Actions ausgeführt werden. Dieser Prozess wird durch die Prozedur UPDATE_EVENTS eingeleitet.

Parameter: complex - gibt an, ob es sich um ein atomares oder komplexes Event handelt, eventID - ist die ID des eingetretenen Events, occurrence - ist der Zeitpunkt des Auftretens

```

1: procedure UPDATE_EVENTS(complex SMALLINT,eventID INT,occurrence DA-
    TETIME)
2:   if not complex then                                ▷ falls auslösendes Event nicht komplex ist
3:     awaiting:= getAwaitingEvents(eventID, occurrence);
    ▷ extrahiere alle Events aus EventFrameworkAwaitingEvent mit gleicher EventID,
    die noch nicht verflossen sind und erhöhe den Recurrence-Wert um 1 (siehe
    6.3.1.)
4:   else if complex then
5:     [...]
6:   end if
7:   for all i in awaiting do
8:     updateEventToInstanceMapping(eventID);                ▷ siehe unten
9:     UPDATE_STAGE(awaiting[1], awaiting[2]);                ▷ zweiter Teil der Evaluation
10:  end for
11: end procedure

```

Mit der Methode 'updateEventToInstanceMapping' wird in der Tabelle 'EventFrameworkEventToInstanceMapping' eine Abbildung zwischen Eventinstanzen, bzw. deren Stages

und den eingetretenen Events, die während dieser Stage 'erwartet' wurden, vorgenommen. Damit kann man im Nachhinein feststellen, welches Event eine bestimmte Veränderung der Eventinstanz verursacht hat.

Durch die Methode UPDATE_STAGE wird kontrolliert, ob eine Veränderung einer Stage in einer Eventinstanz aufgetreten ist. Alle damit verbundenen Änderungsmaßnahmen werden durch Sie veranlasst.

Parameter: instanceID - die ID der untersuchten EventInstanz, started - Zeitpunkt zu dem diese Instanz initiiert wurde

```

1: procedure UPDATE_STAGE(instanceId INT, started DATETIME)
2:   instance := (SELECT FirstStageUri, CurrentStage FROM EventFrameworkComplexEventInstances WHERE EventID = instanceId);
3:   duration := GET_DURATION_OF_STAGE(CAST(instance[0][0] as VARCHAR), CAST(instance[0][1] as INTEGER));
4:   waitTillEnd := duration[0][2];
5:   time := duration[0][3];
   ▷ Informationen über zu untersuchende Eventinstanz (instance) und die aktuelle Stage dieses Instanz (duration) werden abgerufen.
6:   eval := EVALUATE_SET (instanceId, CAST(instance[0][1] AS VARCHAR), started);   ▷ Evaluiere das EventSet der aktuellen Stage und test ob es mit dem gerade eingetretenen Event erfüllt wird.
7:   if eval = 1 then   ▷ EventSet ist erfüllt
8:     eval := CHECK_CONDITIONS(instanceId, CAST(duration[0][1] as INTEGER));   ▷ prüfe ob alle Conditions zu 1 (wahr) ausgewertet werden
9:   end if
10:  if eval = 1 then   ▷ EventSet und Conditions sind erfüllt
11:    if not waitTillEnd OR time = 0 OR DATEDIFF(started, now()) > time then
   ▷ Falls die aktuelle Stage keine Zeitrestriktion hat, oder diese noch nicht überschritten ist, oder waitTillEnd = 1 ist
12:      DELETE FROM EventFrameworkAwaitingEvent WHERE SourceCE = instanceId;   ▷ lösche alle Events dieser Instanz aus der Eventwarteschlange
13:      nextStage:=GET_DURATION_OF_STAGE(CAST(instances[0][0] as VARCHAR), CAST(instance[0][1] as INTEGER)+1);   ▷ prüfe ob es eine nächste Stage gibt
14:      if LENGTH(nextStage) > 0 then   ▷ falls es eine nächste Stage gibt
15:        UPDATE EventFrameworkComplexEventInstances SET CurrentStage = CAST(instance[0][1] as INTEGER) +1 WHERE EventID = instanceId;   ▷ Update Instanz-Einträge
16:        INSERT_AWAITING_EVENTS(instanceId, CAST(instance[0][0] as varchar), CAST(instance[0][1] as INTEGER)+1); ▷ füge die Events der neuen Stage in die Eventwarteschlange ein
17:      end if
18:      if LENGTH(nextStage) = 0 then   ▷ die aktuelle Stage ist die letzte
19:        UPDATE EventFrameworkComplexEventInstances SET Finished = now()

```

```

                WHERE EventID = instanceId;           ▷ Update Instanz-Einträge
20:             INSERT_NEW_EVENT( 1, ", ", now(), ", instanceId);   ▷ löse neues
                (komplexes) Event aus und führe alle Actions dieses Events aus
21:             TAKE_ACTIONS(instanceId);             ▷ führe alle Actions aus
22:             end if
23:         end if
24:     end if
25: end procedure

```

Die eigentliche Evaluation komplexer Events wird offensichtlich in die beiden Funktionen `EVALUATE_SET` und `CHECK_CONDITIONS` ausgelagert. Da die Funktion `CHECK_CONDITIONS` vergleichbar zu `TAKE_ACTIONS` ist, werden beide in einem Zusammenhang im nächsten Abschnitt erläutert. `EVALUATE_SET` evaluiert das EventSet einer Stage, indem bisher eingetretene Events unter Berücksichtigung des zugehörigen Operators ausgewertet werden. Dazu wird auf die Werte der 'Eventwarteschlange' (`EventFrameworkAwaitingEvent`) zurückgegriffen und durch eine rekursive Anwendung dieser Funktion auf interne EventSets eine Lösung erzielt. Die Funktion `GET_DURATION_OF_STAGE(Uri InitialStageUri, INT StageNr)` liefert wichtige Informationen über die ausgewählte Stage einer komplexen Eventdefinition, was durch eine SPARQL-Abfrage realisiert wird. Als Rückgabe erhält man das folgende Array: (Uri der Stage, Uri des EventSets dieser Stage, `waitTillEnd`-Konstante dieser Stage, Zeitrestriktion in Sekunden).

6.3.4. Initiierung von Conditions und Actions

Wird das EventSet einer Stage positiv ausgewertet, d.h. alle Events dieses Sets sind vorschriftsmäßig eingetreten oder ausgeblieben, müssen nachfolgend alle Conditions ausgewertet werden. Handelt es sich um die letzte Stage eines komplexen Events werden zusätzlich alle Actions ausgeführt. In beiden Fällen muss dabei auf die Funktionalität des Event-Service zurückgegriffen werden.

Da Conditions und Actions auf beliebigen SOAP-Endpunkten ausgeführt werden, müssen die entsprechenden SOAP-Nachrichten dynamisch erstellt werden (siehe Kapitel 6.4.1.). Dieses Vorgehen ist nicht vollständig durch Virtuoso unterstützt und wird daher auf den Event-Service ausgelagert. Die in diesem Kapitel beschriebenen Methoden, bzw. Funktionen dienen daher nur der Initiierung dieser Vorgänge von der Zentraldatenbank aus.

CHECK_CONDITIONS wird mit den Parametern eventInstanceID, welche die Eventinstanz und damit das komplexe Event eindeutig definiert, und der aktuellen Stage-Nummer dieser Instanz aufgerufen. Als Resultat wird ein Integer-Wert aus {0,1} erwartet, der angibt, ob *alle* Conditions positiv ausgewertet wurden.

```

1: procedure CHECK_CONDITIONS(eventInstanceID INT,stageNr INT)
2:   actions := (SELECT DISTINCT ActionNr, ActionID FROM EventFrameworkParameterMappings WHERE StageNr = stageNr AND CEID = (SELECT CEID From EventFrameworkComplexEventInstances WHERE EventID = eventInstanceID) ORDER BY ActionNr); ▷ rufe alle Actions dieses komplexen Events ab
3:   aq := async_queue (20);   ▷ lege neue async_queue für Multithread-Aufgaben an
4:   for all i in actions do
5:     result := aq_request (aq, 'CHECK_INNER_CONDITIONS', vector (eventInstanceID, CAST(actions[i][1] as INTEGER)));
   ▷ Führe asynchrone Prozeduraufrufe der Konditionsprüfungen aller Conditions durch.
6:   end for
7:   return evaluateResult(result);           ▷ werte Result-Array aus
8: end procedure

```

Da die Auswertung von Conditions über den Event-Service abgewickelt wird, treten zeitliche Verzögerungen durch das Übertragen von SOAP-Request- und SOAP-Antwer-Nachrichten und der Kompilierung eines SOAP-Clients (vergleiche Kapitel 6.4.1.) auf. Daher werden alle Conditions parallel abgearbeitet und in die integrierten Multithread-Queue von Virtuoso eingereiht. Conditions werden durch den Aufruf der Funktion CHECK_INNER_CONDITIONS parallel (bis zu 20) durch den Event-Service verarbeitet. Das Array 'result' fasst alle Resultate der einzelnen Evaluierungen (mittels '0' bzw. '1'). Mit *evaluateResult(result)* soll eine einfach Auswertung dieses Arrays erfolgen, welche die Methode CHECK_CONDITIONS nur dann mit 'wahr' terminiert, falls alle Konditionsevaluierungen ebenfalls in 'wahr' resultierten.

CHECK_INNER_CONDITIONS wird mit den Parametern eventInstanceID, welche die Eventinstanz und damit das komplexe Event eindeutig definiert, und actionID, die ID der auszuführenden Aktion, aufgerufen. Alle nötigen Parameter zur Evaluierung einer Kondition wird durch eine SQL-Abfrage aus der Tabelle 'EventFrameworkParameterMappings' extrahiert und in einem String-Array, zusammen mit Typinformationen, an die Funktion CheckCondition des Event-Service übergeben. Das Senden einer passenden SOAP-Nachricht wird wie bei Satellitendatenbanken durch die Methode SEND_EVENT_AS_SOAP erzielt.

Die Funktionen TAKE_ACTIONS und TAKE_ACTION zur Ausführung mehrerer Aktionen eines komplexen Events sind fast deckungsgleich zu CHECK_CONDITIONS und CHECK_INNER_CONDITIONS. Ein Unterschied ist, dass kein Resultat erwartet wird, was eine Evaluierung von Einzelresultaten durch evaluateResult(result) überflüssig macht. Außerdem wird die Funktion CallForAction des Event-Service aufgerufen. Die SOAP-Nachricht zum Versenden einer E-Mail von einem Action-Provider aus, könnte wie folgt aussehen:

Listing 14: SOAP-Request-Nachricht an den Event-Service zur Ausführung einer Action

```
<?xml version='1.0' ?>
<SOAP:Envelope
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:SOAP='http://www.w3.org/2003/05/soap-envelope'
  xmlns:SOAP-ENC='http://www.w3.org/2003/05/soap-encoding'
  xmlns:dt='urn:schemas-microsoft-com:datatypes'?>
  <SOAP:Header></SOAP:Header>
  <SOAP:Body>
    <cli:CallForAction SOAP:encodingStyle='http://www.w3.org/2003/05/soap-encoding'
      xmlns:cli='http://tempuri.org/' >
      <controlID xsi:type="xsd:int" dt:dt="int">1340379783</controlID>
      <actionID xsi:type="xsd:int" dt:dt="int">1</actionID>
      <eventInstance xsi:type="xsd:int" dt:dt="int">5</eventInstance>
      <parameters SOAP-ENC:arraySize='4' SOAP-ENC:nodeType='array' SOAP-ENC:itemType
        ='xsd:string'>
        <item>System.String</item>
        <item>noRealEmail@testGraph.net; anOther@email.com</item>
        <item>System.String</item>
        <item>Aenderung Ihres Passwortes</item>
        <item>System.String</item>
        <item>Das Passwort ihres Benutzerkontos wurde geaendert. Bitte bestaetigen
          Sie diesen Vorgang innerhalb der naechsten 7 Tage. Das Log-In mit
          Ihrem alten Passwort ist in dieser Zeit weiterhin moeglich. Ist diese
          Passwortaenderung nicht durch Sie veranlasst worden, koennen Sie hier
          [...] einen Missbrauchsversuch Ihres Benutzerkontos melden.</item>
      </parameters>
    </cli:CallForAction>
  </SOAP:Body>
</SOAP:Envelope>
```

Zum Aufruf dieser Action sind drei Parameter nötig. Parameter 1 fasst alle bekannten E-Mail-Adressen, inklusive der bereits gelöschten Adresse 'noRealEmail@testGraph.net'. Es folgt der Betreff und die Nachricht, der zu versendenden E-Mail. Außerdem wird der Datentyp jedes Parameters angeführt, da alle Parameter als String abgespeichert werden.

Kann eine Action oder Condition nicht ausgeführt werden (z. B. der Provider ist nicht erreichbar, oder Parameter sind falsch), wird dieser Umstand durch das Framework nicht besonders behandelt. Der Provider von Actions oder Conditions allein ist verantwortlich für die Ausführung dieser. In späteren Versionen dieses Frameworks soll die Möglichkeit bestehen, ein Standardergebnis für Evaluierungen einer Condition anzugeben, um für solche Fälle vorzusorgen.

6.4. Event-Service

Der Event-Service nimmt zusammen mit der Zentraldatenbank alle Aufgaben zur Verarbeitung von Events auf sich. Darüber hinaus werden von diesem Service weitere Aufgabenbereiche (wie unter 5.2.2.1. beschrieben) übernommen:

1. Veröffentlichung aller Methoden und Funktionen die Clientanwendungen für die Nutzung des Event-Frameworks benötigen und zentrale Schnittstelle für die Kommunikation mit jedem Client-Service
2. Datenbankkommunikation zum Aufruf von Prozeduren und der Abfrage von Daten
3. Erweiterung der Funktionalität der Zentraldatenbank

Um dem unter 6.1. eingeführten Beispiel besser folgen zu können, wird in diesem Kapitel zuerst auf den dritten Punkt eingegangen. Anschließend wird auf die verschiedenen Schnittstellen, welche vom Event-Service implementiert werden, näher eingegangen.

Daten zwischen Event- und Client-Service sowie den Instanzen des Application-Layers werden zu einem Großteil mit speziellen Datenobjekten ausgetauscht, die den Data-Contract für die in diesem Kapitel genannten Service-Contracts stellen. Deren Struktur orientiert sich dabei eng an der Complex Event Ontology und kann in Appendix C eingesehen werden.

6.4.1. Ausführung von Actions und Conditions

Die Erweiterung der Funktionalität der Zentraldatenbank wird über einen eigenen Service-schnittstelle des Event-Service realisiert. Diese dient allein der passiven Kommunikation mit der Zentraldatenbank, d.h. nur die Zentraldatenbank darf Gebrauch von diesem SOAP-Endpunkt machen. Diese Schnittstelle implementiert den folgenden Service-Contract:

Listing 15: Schnittstellenbeschreibung IVirtuosoExtentionService für den Zugriff der Zentraldatenbank

```
[ServiceContract]
public interface IVirtuosoExtentionService
{
    [OperationContract]
    void CallForAction(int controlID, int actionID, int eventInstance, string[]
        parameters);

    [OperationContract]
    object CheckCondition(int controlID, int actionID, int eventInstance, string[]
        parameters);

    [OperationContract]
    void SendSmtplibMail(string smtpHost, int port, string username, string password,
        string from, string to, string subject, string body);
}
```

Für die Ausführung einer Action ist die Methode 'CallForAction' vorgesehen. Zur Evaluation einer Condition dient 'CheckCondition', welche im Gegensatz zu 'CallForAction' den Rückgabewert der extern ausgeführten Condition an die Zentraldatenbank weitergibt. Die Parameter sind in beiden Fällen: controlID - ein Integer-Wert der die Authentizität des Kommunikationspartners sicherstellen soll, actionID - die ID der auszuführenden Action, bzw. Condition, eventInstance - die ID der Eventinstanz die diese Action (Condition) ausgelöst hat, parameters - ein Stringarray mit den benötigten Parametern und Datentypinformationen. Beide Operation sind vom Aufbau her gleich und können daher im Bezug auf den Prozessablauf zusammengefasst werden.

6.4.1.1. Paramertmapping

An dieser Stelle muss zuvor auf einen Punkt eingegangen werden. Um Action- und Condition-Aufrufe so dynamisch wie möglich zu gestalten muss es neben der statischen Zuteilung von Parametern zur Definitionszeit eines komplexen Events, die Möglichkeit bestehen, Parameter zum Zeitpunkt der Ausführung zu beziehen. Dazu sollen zwei Möglichkeiten geschaffen werden.

1. eine Query an einen bestimmten Endpunkt (z. B. SOAP-, oder Sparql) zu senden (wie unter erweiterten ECA-Regeln vorgesehen, vgl. Kapitel 5.1.)
2. einen bestimmten Wert des Row-Vektors eines zuvor eingetretenen Events zu extrahieren (vgl. Kapitel 6.2.3.) um so auf Ergebnisse von Events zuzugreifen, aus denen das komplexe Event aufgebaut ist

Queries lassen sich ganz ähnlich zu Actions und Condition direkt über das Event-Framework-Control anlegen und sollen eine Abfrage auf einen bestimmten Endpunkt ausführen, mit dessen Ergebnis als Parameter fortgefahren wird. Hierzu ist die ID und alle Parameter dieser Query notwendig (vergleichbar zu Actions und Conditions). Allerdings könnte auch in diesem Fall nur statische Parameter für Queries im Voraus verwendet werden. Diesem Problem wird mit dem sogenannten Event-Value-Mapping begegnet. Ziel ist es einen Wert des Row-Vektors eines zuvor eingetretenen Events zu extrahieren und als Parameter einzusetzen. Dazu wurde eine eigene Syntax entwickelt. Da Row-Vektoren zur Zeit nur für atomare Events unterstützt werden, muss vom Ursprung des komplexen Events immer zu einem atomaren Event und dem enthaltenen Row-Vektor-Wert navigiert werden.

Listing 16: Beispiel: Event-Value-Mapping

```
##CE1/STAGE2/ES77/MS6/5/AE32/VALUE(2)##  
  
Syntax:  
AE - Atomic Event  
CE - Complex Event  
STAGE - die zu untersuchende Stage eines komplexen Events  
ES - EventSet  
MS - MultiSet (die darauffolgende Zahl (MS6/5) bestimmt die zu betrachtende Recurrence  
)  
VALUE(1) - extrahiert den Wert an Position 1 des Row-Vektors dieses Events  
## - grenzt die Event-Value-Map deutlich von anderem Fliesstext ab
```

Da sich ein solches Mapping immer auf die aktuelle Eventinstanz bezieht lässt sich das betrachtete Beispiel wie folgt formulieren: Gesucht wird der Wert auf Stage2 im EventSet77. Dieses EventSet enthält ein MultiSet (MS6) welches das atomare Event AE32 mehrmals erwartet. Die fünfte Wiederholung dieses atomaren Events enthält den gesuchten Wert an Row-Vektor-Position 2.

Genau wie statische Parameter werden Event-Value-Mappings in die Tabelle 'EventFrameworkParameterMappings' aufgenommen und mit Hilfe der Abbildungen aus Eventinstanzen und aufgetretenen Events in Tabelle 'EventFrameworkEventToInstanceMapping' (vgl. Kapitel 6.3.3.) zur Ausführungszeit von Actions und Konditionen ausgewertet.

Voraussetzung ist dass das gesuchte Event Teil des gleichen komplexen Events ist und bereits eingetreten ist. Diese Event-Value-Mappings lassen sich direkt in Abfragetext (unabhängig von der Abfragesprache) einer Query aufnehmen, um Queries mit dynamischen Parametern zu nutzen.

6.4.1.2. Beispiel: Aufruf von CallForAction

Am Beispiel von 'CallForAction' soll das Ausführen einer Action, welche auf einem externen SOAP-Endpoint veröffentlicht wurde demonstriert werden.

Listing 17: CallForAction aus VirtuosoExtentionService

```
public void CallForAction(int controlID, int actionID, int eventInstance, string []
    parameters)
{
    DataTable t = eventService.GetActionById(actionID);
    if ((int)(eventService.GetControlId(1)) == controlID && t.Rows.Count > 0)
    {
        EventAction action = new EventAction(t, 0);
        string [] forwardParams = new string[parameters.Length / 2];

        for (int i = 0; i < parameters.Length; i++)
        {if (i % 2 == 0)
            {if (parameters[i] == "query")
                {
                    DataTable q = eventService.GetActionById(int.Parse(parameters[i + 1].
                        ToString()));
                    EventAction query = new EventAction(q, 0);
                    if (query.SparqlQuery.Contains("##"))
                    {
                        MatchCollection matches = valMapCheck.Matches(query.SparqlQuery);
                        foreach (Match mat in matches.OfType<Match>().OrderByDescending(x
                            => x.Index))
                        {
                            query.SparqlQuery = query.SparqlQuery.Remove(mat.Index, mat.
                                Length);
                            query.SparqlQuery = query.SparqlQuery.Insert(mat.Index,
                                evaluateValueMap(eventInstance, mat.Value));
                        }
                    }
                    forwardParams[i / 2] = query.InvokeSparqlQuery().ToString();
                }else if (parameters[i] == "valueMap")
                {
                    MatchCollection matches = valMapCheck.Matches(parameters[i + 1].
                        ToString());
                    string res = parameters[i + 1];
                    foreach (Match mat in matches.OfType<Match>().OrderByDescending(x => x
                        .Index))
                    {
                        res = res.Remove(mat.Index, mat.Length);
                        res = res.Insert(mat.Index, evaluateValueMap(eventInstance, mat.
                            Value));
                    }

                    forwardParams[i / 2] = res;
                } else //static value
                    forwardParams[i / 2] = parameters[i + 1];
            }
        }
        action.InvokeRemoteMethod(forwardParams);
    }
}
```

In dieser Methode müssen zuerst das Parameterarray ausgewertet werden. Es gilt neben statischen Werten, etwaige Queries auszuführen, bzw. Event-Value-Mappings in tatsächliche Parameter umzuwandeln. Queries werden direkt durch den Aufruf von *query.InvokeSparqlQuery()* ausgeführt (Anmerkung: nicht nur für Sparql-Abfragen). Müssen Event-Value-Mappings ausgewertet werden, wird die Funktion *private string evaluateValueMap(int eventInstance, string eventValueMap)* ausgeführt, welche den gesuchten Parameterwert als String extrahiert. Durch *action.InvokeRemoteMethod(forwardParams)* wird die die Kompilierung eines SOAP-Clients zur Ausführung der gewünschten Methode auf einem externen SOAP-Endpoint unter Verwendung der extrahierten Parameter eingeleitet. Dazu muss das Wsdl-Dokument

[40] des SOAP-Endpunktes analysiert werden, um daraus automatisch einen SOAP-Client zu erstellen, der alle auf diesem SOAP-Endpunkt veröffentlichten Methoden als Schnittstelle implementiert. Dazu wird auf WCF Dynamic Proxy zurückgegriffen [41] - eine bekannte Beispielbibliothek von Microsoft, für eben diese Aufgabe. Methoden dieser dynamisch erstellten Klasse lassen sich dann mittels *public Object Invoke(Object obj, Object[] parameters)* unter C# aufrufen [42]. Bei der Ausführung einer Condition auf einem Externen SOAP-Endpunkt wird ein Integer als Rückgabewert erwartet {0,1}, welcher zurück an die Zentraldatenbank gereicht wird. In dem betrachteten Beispiel würde eine Methode aufgerufen werden, welche drei Parameter als Strings erwartet und eine E-Mail an all im ersten Parameter gelisteten Empfänger sendet, welche neben dem Betreff (zweiter Parameter) um die Nachricht (dritter Parameter) ergänzt wird. Damit ist das unter 6.1. angestoßene Beispiellevent abgeschlossen. Darüber hinaus gibt es aber weitere Möglichkeiten vom Event-Framework Gebrauch zu machen.

6.4.2. Verbindungen mit Datenbanken

Der Zugriff auf Daten und Funktionalität der Zentraldatenbank vom Event-Service erfolgt durch eine ODBC-Verbindung, ausschließlich mit Hilfe von SQL-Kommandos. Einige Aufgaben verlangen einen direkten Zugriff auf Satellitendatenbanken. Dies wird über die SOAP-Endpunkte der jeweiligen Datenbank abgewickelt, welche die verlangte Funktionalität über SQL-Prozeduren auf diesen bereitgestellt haben.

6.4.2.1. ODBC-Verbindung zur Zentraldatenbank

Die ODBC Verbindung mit der Zentraldatenbank wird beim Start des Service aufgebaut und erst beim Beenden des Service wieder verworfen (es wird davon ausgegangen, dass Zentraldatenbank und Event-Service auf einem Server operieren).

Listing 18: Beispiel: Ausführung eines SQL-Kommandos in C#

```
public int GetControlId(int dsInstance)
{
    IDbCommand dbcmd = StaticHelper.dbcon.CreateCommand();
    dbcmd.CommandText = "SELECT ControlID FROM EventFrameworkDataSources WHERE DSInstance"
        + dsInstance.ToString();
    int zw = (int)dbcmd.ExecuteScalar();
    dbcmd.Dispose();
    dbcmd = null;
    return zw;
}
```

Die Instanz dbcon der Klasse OdbcConnection wird statisch bereitgestellt. Werden nicht-skalare Werte als Antwort erwartet, werden diese in einen DataTable übernommen und an den Client-Service weitergegeben.

6.4.2.2. Verbindungen mit Satellitendatenbanken

Muss direkt vom Event-Service auf eine Satellitendatenbank zugegriffen werden, z. B. um diese zu registrieren, oder wie im Beispiel unten, deren Tabellennamen abzurufen, wird ein neuer SOAP-Client für einen solchen Endpunkt erstellt (EventFrameworkProceduresDocLiteralPortTypeClient). Dazu wird die Endpunktadresse dieser Satellitendatenbank von der Zentraldatenbank abgefragt. Anschließend wird die gewünschte Methode mit Hilfe des erstellten Clients ausgeführt und das Ergebnis an den Clientservice weitergegeben.

Listing 19: Beispiel: Aufruf einer über den SOAP-Endpunkt einer Satellitendatenbank bereitgestellten Funktion (GET_SCHEMA_TABLES) in C#

```
public string[] GetSchemaTables(int dsInstance = 1)
{
    EndpointAddress addr = new EndpointAddress(GetRemoteProcedureEndpoint(dsInstance));

    EventFrameworkProceduresDocLiteralPortTypeClient virtuosoSoapClient = new
        EventFrameworkProceduresDocLiteralPortTypeClient(bind, addr);
    string[] zw = null;

    zw = virtuosoSoapClient.GET_SCHEMA_TABLES((int)(GetControlId(dsInstance)));

    virtuosoSoapClient.Close();
    virtuosoSoapClient = null;
    return zw;
}
```

6.4.3. Schnittstelle für den Client-Service

Die Hauptaufgabe des Event-Service ist die Funktionalität des Event-Frameworks über den Client-Service für Anwendungen zu veröffentlichen. Dazu wird das folgende Interface implementiert (das komplette Interface ist unter Appendix D hinterlegt):

Listing 20: Auszug: IVirtuosoEventService - Schnittstelle für den Client-Service

```
[ServiceContract]
public interface IVirtuosoEventService
{
    [OperationContract]
    DataTable GetEvents(int minutes);

    [OperationContract]
    DataTable GetDatabases(string type);

    [OperationContract]
    DataTable GetAllTriggers(string likeName);

    [OperationContract]
    string DeleteActionsOrConditions(User admin, int actionID);
    [...]
}
```

Aus Zeitgründen ist es dem Autor nicht möglich auf jede der fast 50 Funktion einzugehen. Zusammenfassend lässt sich aber sagen, dass die unter 5.2.2.1. angeführten Aufgabengebiete abgedeckt werden und dadurch besonders die Anforderungen des Event-Framework-Control erfüllt werden. Eine Erweiterung dieser Funktionalität auch für andere Anwendungen und deren Anforderungen lässt sich über diese oder neue Schnittstellen jederzeit ergänzen.

6.5. Client-Service

Der Client-Service ist das Tor zum Event-Framework für jede Anwendung auf Client-Computern und stellt den ersten Client-seitigen Teil des Frameworks dar. Dieser Service ist die Grundlage für den Zugriff auf das Event-Framework, was eine einheitliche und sichere Schnittstelle zwischen Client- und Event-Service garantieren soll. Dieser Service soll in zukünftigen Versionen in der Lage sein, Programme aktiv mit Daten des Event-Frameworks zu versorgen und nicht nur Anfragen an das Framework an den Event-Service weiterleiten. Der Client-Service implementiert das folgende Interface (das komplette Interface ist unter Appendix D hinterlegt):

Listing 21: Auszug: IEventClient - Schnittstelle mit dem Event-Service

```
[ServiceContract(Namespace = "http://EventFrameworkClient/Operations")]
public interface IEventClient
{
    [OperationContract]
    DataTable GetEvents(int minutes);

    [OperationContract]
    User LogIn(User currentUser);

    [OperationContract]
    bool CreateNewAccount(User newUser);

    [OperationContract]
    bool ResetUserpassword(string userName, string newPass);

    [OperationContract]
    bool UpdateUserAccount(User user);
    [...]
}
```

Diese Schnittstelle ist nahezu deckungsgleich mit der IVirtuosoEventService-Schnittstelle des Event-Service. Allerdings werden durch den Client Service zusätzliche Informationen über den aktuellen Nutzer und Session an den Event-Service übergeben. Zur Nutzung des Frameworks ist daher eine Anmeldung des Benutzers über den Client-Service von Nöten. Diese Anmeldung wird über das Event-Framework-Control manuell vorgenommen und soll für einmal registrierte, bzw. bekannte Anwendungen automatisch erfolgen. Mit der erfolgreichen Anmeldung hat ein Benutzer (und dessen Anwendungen), entsprechende Rechte (siehe 6.6.1.1.) vorausgesetzt, Zugriff auf die komplette Funktionalität des Event-Frameworks.

Anders als die bisherigen Kommunikationsmethoden wird der Austausch von Daten zwischen Client-Anwendungen und Client-Service durch eine Inter Process Communication (IPC) hergestellt. Dazu wird auf Pipe-Binding des .NET-Frameworks zurückgegriffen [37]. Die relativ kostspielig Übertragung von XML-basierten Daten mittels SOAP ist im Umfeld eines Computer-Systems überflüssig.

6.6. Event-Framework-Control

Für die Anwendungsebene wurde eine Anwendung zur Administration des Event-Framework implementiert, welches bereits vorher unter dem Namen Event-Framework-Control Erwähnung fand. Diese Anwendung wurde als ein Windows-Forms Projekt in C# implementiert. Da die vorliegende Version nur einen Zwischenstand der Entwicklung darstellt und mittels Mono noch keine Lösung geschaffen wurde das Event-Framework-Control auf anderen Betriebssystemen als Windows einzusetzen, wird in diesem Kapitel weniger auf die eigentliche Implementierung, als auf die bereits abgedeckten Anforderungen (siehe Kapitel 5.2.3.1.) sowie den Grundlegenden Aufbau der Anwendung eingegangen.

6.6.1. Benutzer- und Rechteverwaltung

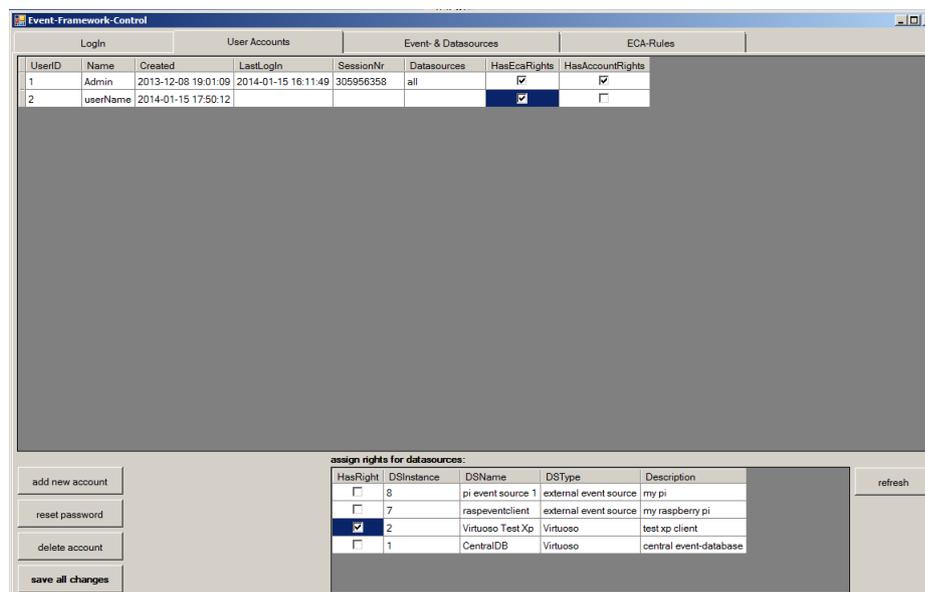


Abbildung 4: Useraccounts und Rechte

Ein einfaches Rechtssystem soll eine genaue Zuteilung von bestimmten Funktionalitäten auf Benutzer ermöglichen. Zur Zeit wurden Beispielphaft drei Rechte aufgenommen:

UserAccount Rechte - diese Benutzer dürfen neuen Benutzer registrieren und eigene Rechte weitergeben

ECA Rechte - diese User dürfen komplexe Events erstellen/ändern und alle dafür nötigen Aufgaben ausführen

Datenquellen Rechte - ein User hat das Recht atomare Events auf den von ihm registrierten Datenquellen zu definieren, dieses Recht kann an andere Nutzer weitergegeben werden

Diese Rechte lassen sich leicht erweitern, bzw. feinkörniger gestalten.

6.6.2. Verwaltung von Satellitendatenbanken, Event- und Datenquellen

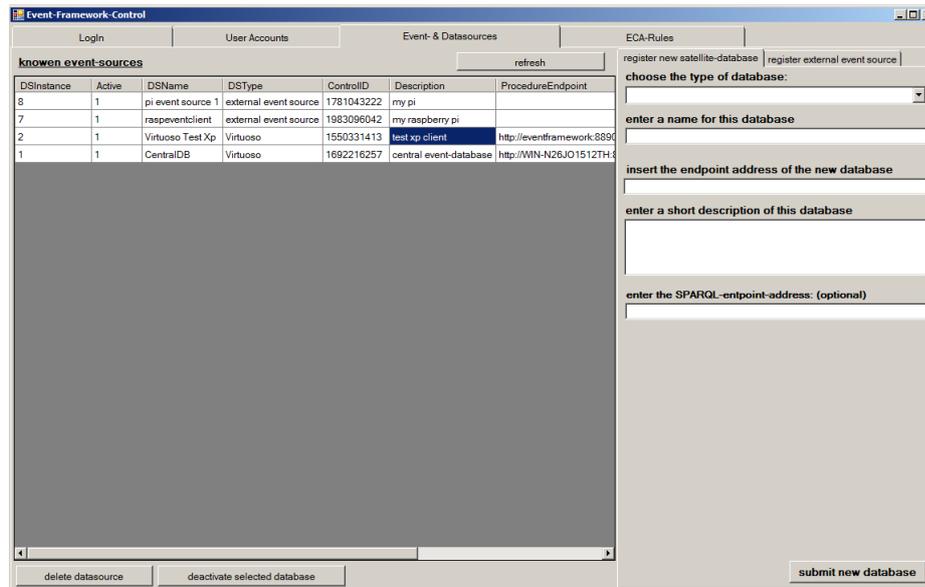


Abbildung 5: Verwaltung von Satellitendatenbanken und anderen Datenquellen

Neben Satellitendatenbanken können andere, externe Event- und Datenquellen registriert werden. Jede Datenquelle kann vorübergehend deaktiviert werden, so dass keine atomaren Events von dieser Quelle berücksichtigt werden, bzw. diese erst gar nicht ausgelöst werden.

6.6.3. Definition von Events auf Satellitendatenbanken und externen Datenquellen

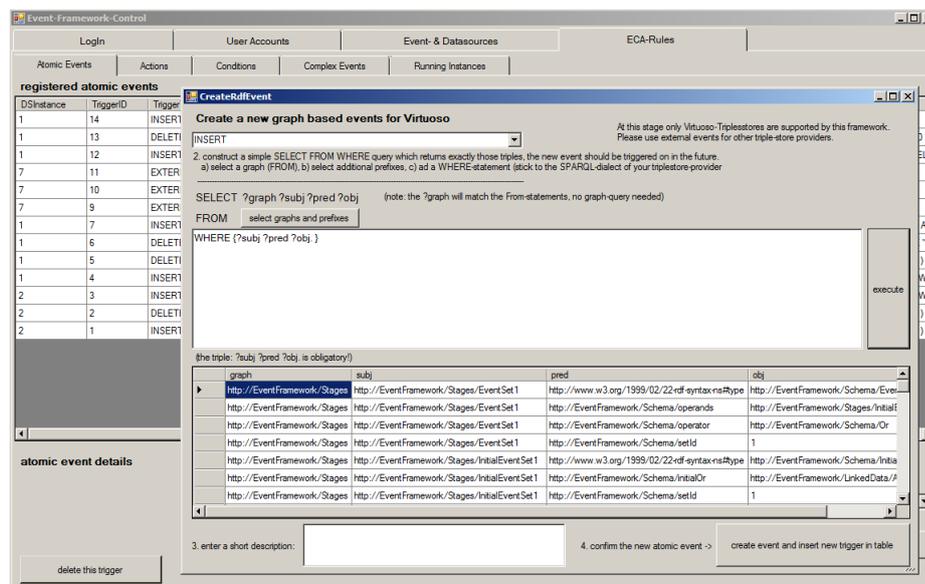


Abbildung 6: Definition eines atomaren Events für einen Triple-Store

Atomare Events lassen sich direkt auf Satellitendatenbanken als Trigger von Tabellen oder für RDF-Daten für den integrierten Triple-Store von Virtuoso definieren. Außerdem können externe Events als atomare Events registrieren.

6.6.4. Erstellung/Registrierung von Actions, Conditions und Queries

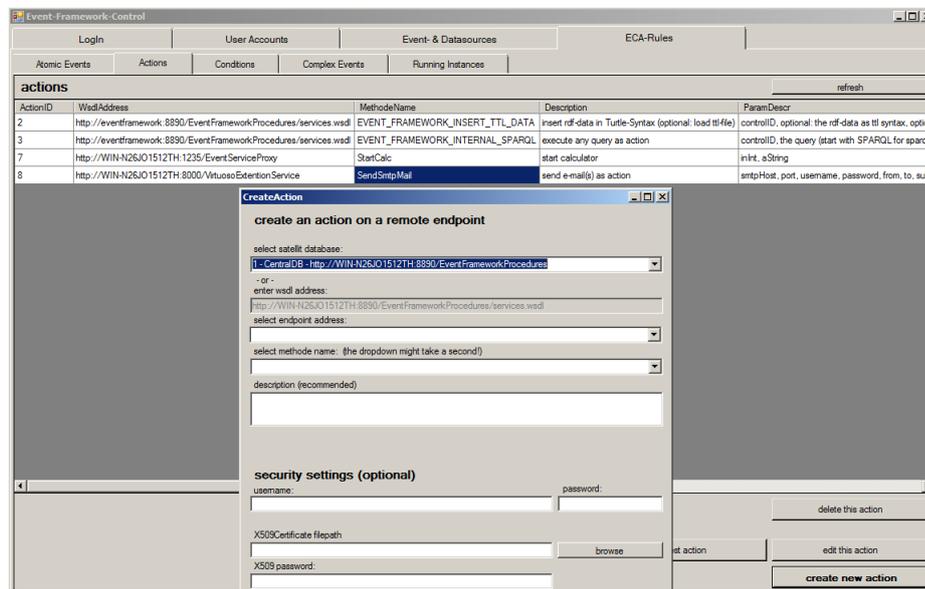


Abbildung 7: Definition einer Action für einen SOAP-Endpoint

Actions, Conditions und Queries lassen sich über die hier gezeigte Maske auf SOAP-Endpunkten definieren. Dafür werden mindestens die folgenden Parameter benötigt:

1. Adresse des Wsdl-Dokuments dieses Endpunktes
2. Adresse des zu verwendenden Endpunktes
3. Name der auszuführenden Methode
4. zu verwendende Parameter für diese Methode (statisch, als Query oder als Event-Value-Mapping)

Zusätzlich soll in zukünftigen Versionen auch die Verwendung von Endpunkten möglich sein, die einer Authentifizierung bedürfen.

Die so erstellten Actions können hier jederzeit getestet oder geändert werden.

6.6.5. Definition von komplexen Events

The screenshot shows the 'Event- Framework-Control' application window. The main area is titled 'Event Stages' and contains three numbered steps: 1. Define events of this stage (with buttons for 'Initial Stage' and 'Event-Set'), 2. Select additional Conditions (with a 'Conditions' button), and 3a. Select actions which will be taken after all events and conditions are met (with an 'Actions' button). There is also a '3b. or - create a new Stage:' section with a 'Next Stage' button and a 'Remove Stage' button. On the left, the 'New Complex Event' section includes fields for 'Name', 'Description', and 'activate now', along with a table for 'manage existing events'.

CEID	Name	Description	IsActive	IsOverlapping	Recurrences	Period
1	a complex test	some complex testing	1	0	1	PT
2	a new test complex event	some more testing	1	0	1	PT
3	action test	sgfewgrewer	1	0	1	PT
4	poster chile	poster	1	0	1	PT

Abbildung 8: Definition eines komplexen Events

Die Definition von komplexen Events erfolgt über die abgebildete Maske. Neben Namen und Beschreibung des Events ist die Definition der einzelnen Stages von Bedeutung. Dazu wird dieser Vorgang aufgeteilt in:

1. Definition des EventSets
2. Zuteilung von etwaigen Conditions
3. Zuteilung von Actions falls die letzte Stage betrachtet wird

Bereits definierte Events können geladen und geändert werden. Außerdem lässt sich hier mit dem Button 'activate event now' eine Instanz dieses Events anlegen. Eine Instanziierung von Events zu bestimmten Zeitpunkten, bzw. in periodischen Abständen wird ebenfalls unterstützt.

6.6.6. Definition von EventSets aus Events, Actions, Conditions und Queries

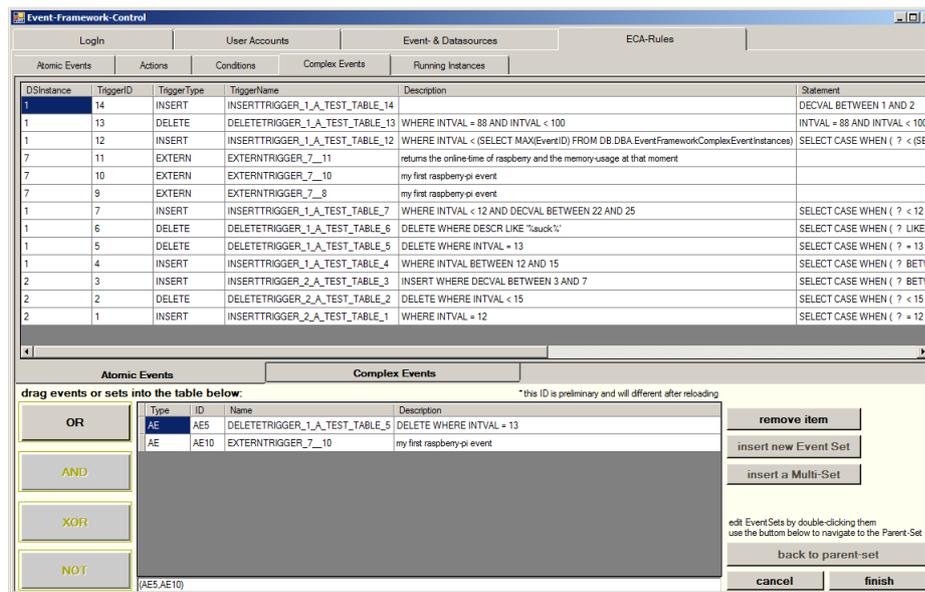


Abbildung 9: Definition eines EventSets

Zur Definition eines EventSets können atomare und komplexe Events von den oberen Tabellen, einfach in das EventSet gezogen werden. Außerdem können neue EventSets bzw. MultiSets eingefügt werden.

6.7. Systemanforderungen

Entwickelt wurde dieses Framework mit Hilfe eines Windows-Server 2008. Für den Betrieb der Zentraldatenbank und dem Event-Service sollte eine ähnliche Serverumgebung dienen. Der Test für Linux Distributionen wurden auf einer Ubuntu 12 (Desktop) Instanz erfolgreich vorgenommen, andere Linux Distributionen als zentraler Server sind aber denkbar. Mono in Version 2.10 oder höher ist Voraussetzung für das Ausführen des Event- und Client-Service unter Linux. Unter Windows können beide Serviceinstanzen direkt ausgeführt werden. Als Zentraldatenbank muss Virtuoso in Version 6.1.6, 6.1.7 oder 7.0.0 zum Einsatz kommen. Das Ausführen der Event-Framework-Control Applikation ist zur Zeit nur unter Windows mit .NET Framework 4.0 (oder höher) möglich, soll aber in späteren Versionen auch für den Betrieb unter Linux konfiguriert werden. Satellitendatenbank sind als Instanzen von Virtuoso 6.1.6 oder 6.1.7 angedacht. Weitere Datenbanksysteme sind hier theoretisch möglich. Dafür müssen alle verwendeten Prozeduren für das entsprechende DBS angepasst werden. Anmerkung: Virtuoso 7.0.0 enthält einen Fehler, daher ist von der Verwendung dieser Version als Satellitendatenbank abzuraten.

Externe Datenquellen, bzw. Provider von Actions, Conditions oder Queries welche über einen SOAP-Endpunkt verfügen, müssen diesen mit einem Wsdl-Dokument ausstatten. Andere Endpunkte zur Ausführung von Queries (z. B. SPARQL) müssen über HTTP zugänglich sein.

7. Evaluation und Diskussion

In diesem Kapitel wird eine kritische Analyse des bisher Erreichten vorgenommen. Da das Event-Framework zum Zeitpunkt dieser Ausarbeitung noch keinen finalen Zustand erreicht hat existieren noch einige Baustellen und Unzulänglichkeiten, welche Nach der Beendigung dieser Arbeit anzugehen sind. Im Verlauf von Recherche und Implementierung sind außerdem neue Anforderungen, fehlende Funktionalitäten und Verbesserungsmöglichkeiten aufgezeigt worden. Diese sollen hier diskutiert werden, um Ziele für zukünftige Weiterentwicklungen auszuloten.

Zuerst wird auf die unter Kapitel 3. aufgestellten Benutzeranforderungen eingegangen und deren Umsetzung beurteilt. In diesem Zusammenhang werden auch alle Anwendungsfälle erneut aufgegriffen und auf Lösungswege mit dem Event-Framework aufgezeigt. Anschließend soll ein Ausblick auf zukünftige Herausforderungen dieses Kapitel abschließen.

7.1. Umsetzung von Benutzeranforderungen

Die im Kapitel 3. herausgearbeiteten Benutzeranforderungen sollen auf deren Umsetzung durch das Event-Framework beurteilt werden.

7.1.1. BA1: Verteiltheit

Ziel dieser Anforderung war eine möglichst große Basis an zu verwendenden Event- und Datenquellen, sowie aus vielen möglichen Providern für Actions, Conditions und Queries wählen zu können. Da das Framework keine Einschränkungen bezüglich bestimmter Entitäten macht, sondern alle Quellen von Eventressourcen akzeptiert, wenn diese eine passende SOAP-Nachricht an den SOAP-Endpunkt der Zentraldatenbank senden, wird dieser Punkt erfüllt. Darüber hinaus kann sogar der Triple-Store einer Virtuoso-Datenbank als aktive Eventquelle genutzt werden. Bei dem beschriebenen Mangel an reaktiven Triple-Stores, ist dies besonders hervorzuheben. Als Provider für Actions oder Condition kann jeder SOAP-Endpunkt eingesetzt werden. Außerdem können Queries und Conditions auf verschiedenen Query-Endpunkten, wie z. B. SPARQL-Endpunkten, ausgeführt werden. Diese Möglichkeiten erlauben einen Einsatz des Event-Frameworks auf verteilten Datenquellen.

7.1.2. BA2: Systemunabhängigkeit

Eine Systemunabhängigkeit wird durch verschiedene Aspekte hergestellt. Durch die Verwendung von SOAP als Netzwerkprotokoll zur Nachrichtenübertragung zwischen einzelnen Teilen des Frameworks ist eine systemunabhängige Übertragung von Daten garantiert. Unabhängigkeit von bestimmten Datenbanksystemen ist nur im Ansatz umgesetzt. Satellitendatenbanken wurden zwar als Virtuoso-Instanzen vorgesehen, können aber durch eine gesonderte Implementierung der verwendeten Prozeduren auf anderen Datenbanksystemen auf diese erweitert werden. Ansätze dazu wurden bereits unter TSQL-Datenbanken realisiert. Durch die Verwendung von Mono [39] als ein Surrogat für NET-Framework-Bibliotheken, ist es gelungen, den kompletten Service-Layer auf Windows und Linux-Distributionen zum Einsatz zu bringen. Aus Zeitgründen wurde noch keine funktionierende Implementierung des Event-Framework-Control unter Linux-Distributionen erarbeitet.

Wichtige Schritte in Richtung Systemunabhängigkeit sind vollzogen worden. Weitere Anstrengungen sind nötig, aber unter den gegebenen Voraussetzungen in greifbarer Nähe.

7.1.3. BA3: Eigenständigkeit

Alle Voraussetzungen für eine selbstständige Auswertung von komplexen Events werden durch Zentraldatenbank und Event-Service bereitgestellt. Externe Module zur Lösung von Aufgaben werden nicht eingesetzt.

7.1.4. BA4: Zuverlässigkeit bei großen Zugriffszahlen

Diese Anforderung bedarf einer genauen Evaluation, unter klar definierten Testbedingungen, die zum jetzigen Zeitpunkt im Hinblick auf den Stand der Implementierung noch nicht vollzogen wurde. Fehler die auf das gleichzeitige Ausführen mehrerer paralleler Eventevaluationen zurückzuführen sind, wurden bisher noch nicht beobachtet.

7.1.5. BA5: Verlässlichkeit bei der Erkennung von Events

Dieser Punkt war stets erster Ansatzpunkt bei der Kontrolle von Funktionalität und Zuverlässigkeit des Event-Frameworks. Nach langwierigen Tests, kann von einem hohen Grad an Zuverlässigkeit bei der Verarbeitung von komplexen Events, bis hin zur Ausführung von Actions, ausgegangen werden. Aber auch hier gilt, dass weitere Tests unter klar definierten Bedingungen nötig sind, um ein eindeutiges Bild zu erhalten.

7.1.6. BA6: Echtzeit

Echtzeit dient auch in diesem Zusammenhang eher als Schlagwort und nicht als klar definierter Begriff. Die schnelle Ausführung aller verbundenen Aufgaben bei der Verarbeitung von komplexen Events wurde bei der Implementierung des Event-Frameworks stets berücksichtigt. Zeitliche Verzögerungen können bei der Ausführung von Conditions und Actions auftreten, da zuerst ein SOAP-Client erstellt und kompiliert werden muss. Dabei können bis zu fünf Sekunden vergehen. Besonders in Bezug auf Conditions kann dies zu Problemen bei sehr kurzlebigen komplexen Events führen. Eine Überarbeitung des Frameworks im Bereich der Ausführung von externen Actions und Conditions ist daher vorgesehen.

7.1.7. BA7: physische Datenunabhängigkeit

Durch den Einsatz externer Event- und Datenquellen ist eine Garantie für physische Datenunabhängigkeit, der durch diese Quellen zur Verfügung gestellten Daten, nicht möglich. Alle Daten, welche durch die Zentraldatenbank oder Satellitendatenbanken bereitgestellt werden, unterliegen dem Grundsatz jeder Datenbank, die logische Struktur aller Daten unabhängig von der physischen Struktur und deren Veränderungen zu präsentieren.

7.1.8. BA8: Datenschutz und Sicherheit

Dieser Anforderung eine hohe Priorität in Kapitel 3.3. zugesprochen. Aus Zeitgründen konnte bisher auf Datenschutz und Sicherheit bei der Übertragung von Daten nicht eingegangen werden. SOAP-Nachrichten werden über das Netzwerkprotokoll HTTP und nicht mittels HTTPS ausgetauscht. Jeder Nutzer mit ECA-Rechten (vgl. Kapitel 6.6.1) hat Zugriff auf alle Events, deren Definitionen sowie Actions, Condition und Queries. Dieser Umstand lässt sich durch eine Verfeinerung des in Ansätzen vorhandenen Rechtesystems beseitigen. Bei den weiteren Arbeiten am Event-Framework muss daher diese Anforderung in den Fokus der Entwicklung rücken.

7.1.9. BA9: freie Wahl und Definition von Events und Ressourcen

Durch die Verwendung von externen Event- und Datenquellen sowie der möglichen Ausführung von Actions, Conditions und Queries auf jeder denkbaren Netzwerkentität, wurde dieser Anforderung Rechnung getragen. Bei der Definition von Events sind keine besonderen Einschränkungen zu beachten.

7.1.10. BA10: Verwaltung von Benutzern und Eventressourcen

Das Event-Framework-Control bietet umfangreiche Möglichkeiten zur Verwaltung von Benutzern und deren Rechte. Event- und Datenquellen können ebenfalls mit dieser Applikation verwaltet werden.

7.1.11. BA11: RDF-Events

Durch die Erweiterung der Reaktivität der verwendeten Virtuoso-Datenbanken auf den Triple-Store von Virtuoso (vgl. Kapitel 6.2.1.) können atomare Events auf RDF-Metadaten erkannt und signalisiert werden. Diese Benutzeranforderung ist daher in Bezug auf Virtuoso-Datenbanken erfüllt.

7.2. Lösungen für Anwendungsfälle

Hier sollen mögliche Lösungsansätze der unter Kapitel 3.1. angeführten Anwendungsfälle mit Hilfe des Event-Frameworks aufgezeigt werden.

7.2.1. AF1: Cloud-Events

Unter der Annahme, dass der Cloud-Service auf Grundlage einer Satellitendatenbank des Event-Frameworks operiert, ließen sich die geforderten Funktionen leicht umsetzen. Atomare Events, wie Änderungen am Datenbestand eines Nutzers, ließe sich über einfache Trigger mit dem Event-Framework-Control definieren. Über komplexe Events könnten diese in Berichten zusammengefasst werden, die mit einer definierten Action eine Zusammenfassung an die Endgeräte des Nutzers senden oder andere automatische Aufgaben übernehmen.

7.2.2. AF2: Automatisiertes Dokumentenverwaltungssystem

In einem ersten Schritt wird ein komplexes Event erstellt, welches beim Einfügen oder dem Update von Dokumenten den enthaltenen Text mit dem zu verwendenden Name-Entity-Recognition-Verfahren auswertet. Dazu wurde dieses Verfahren als Funktion auf einem SOAP-Endpunkt veröffentlicht, die nun als Action eines komplexen Events verwendet werden kann. Die Ergebnisse dieses Verfahrens (d. h. die Schlagworte) werden ebenfalls auf einer Datenbank hinterlegt. Das Einfügen dieser Tupel wird in einem zweiten Schritt als atomare Events definiert, welche als Grundlage für weitere komplexe Events dienen, deren Actions auf die Funktionalität des neuen Dokumentenverwaltungssystems zurückgreifen.

7.2.3. AF3: Produktionsablaufüberwachung

Das Einscannen eines Barcodes an allen Produktionsstationen wird als atomares Event über Event-Framework-Control definiert (Anmerkung: zur reinen Dokumentation dieser Vorgänge ist die Definition von komplexen Events nicht nötig). Bei besonderen Ereignissen, die entweder als atomare Events definiert wurden oder sich aus komplexe Events schließen lassen, können Nachrichten an Mitarbeiter über zuvor definierte Actions gesendet werden. Oder es müssen sofortige Gegenmaßnahmen ergriffen werden (z. B. automatisches Abschalten einer Produktionslinie).

7.2.4. AF4: Benutzerkontenabsicherung eines Onlineshops

Eine ausführliche Lösung dieses Szenarios wurde in Kapitel 6. diskutiert.

7.2.5. AF5: Flexible Teilautovermietung

Anmerkung: Voraussetzung für dieses Szenario sind Autos die nicht mit herkömmlichen Zündschlüssel, sondern mit einem elektronischen Authentifizierungsverfahren des Fahrers gestartet werden. Der Lösungsansatz des beschriebenen Szenarios soll hier nur in Auszügen vorgestellt werden

Meldet sich ein Fahrer bei einem Auto ab, werden alle Interessenten in der Umgebung des Abstellortes über diesen informiert. Hierzu wird das atomare Event der Abmeldung mit der Aktion des Informierens von anderen Nutzern in der Nähe des Fahrzeuges kombiniert. Ein anderer Nutzer sichert sich dieses Fahrzeug online oder per SMS und holt es ab. Während der Fahrt wird der Fahrer über Rabatte für das Abstellen des Fahrzeuges in bestimmten Stadtvierteln informiert. Dazu werden über Queries Stadtviertel ermittelt, die zur Zeit mit Fahrzeugen unterbesetzt sind und über einfache Actions an die mobilen Endgeräte der Nutzer weitergegeben.

7.2.6. AF6: Automatische Aufgaben in sozialen Netzwerken

Über Definitionen von atomaren Events (z. B. das Bewerten eines Artikels) und Actions (z. B. das Senden einer Bestätigung einer Freundschaftsanfrage), lassen sich einfache komplexe Events erstellen, welche diese beiden Gruppen entsprechend verbinden und so automatische Reaktionen in sozialen Netzwerken realisieren.

7.2.7. AF7: Zentrales Event-System einer Hochschule

Hierzu müssen die entsprechenden Quellen regelmäßig auf neue Einträge geprüft werden. Externe atomare Events könnten über diese Umstände informieren und die entsprechenden Updates mit dem Row-Vektor des atomaren Events übertragen. Ein komplexes Event wertet jedes Update aus (z. B. Queries und Conditions) und fügt mit einer Action ein neues Update im zentralen Event-System der Hochschule ein.

7.2.8. AF8: Überwachung von Anlagen erneuerbaren Energie

Ganz ähnlich zu AF2, bzw. AF3 könnten hier atomare und komplexe Events verwendet werden, um allgemeine Informationen über die Anlagen zu senden oder besondere Vorkommnisse zu berichten.

7.2.9. AF9: Automatisierte Terminplanung

Nach der Definition atomarer Events, welche Notfallaufnahmen und Vordiagnosen dokumentieren, übernimmt ein komplexes Events die Einordnung eines neuen Falles in den Terminplan eines Arztes. Dazu wird mit einer Query die Priorität einer Behandlung ermittelt und die Terminpläne von zuständigen Ärzten verglichen. Informationen über neue Notfälle werden mit einfachen komplexen Events ausgelöst, deren Actions das Senden der Notfallinformation an die Endgeräte der Ärzte übernehmen.

7.2.10. AF10: Lokale Auswertung von Wahlkampfstrategien

Atomare Events müssen über neue Umfrageergebnisse berichten. Deren Auswertung könnte als Ergebnis einer Query zurückgegeben werden, die in einem komplexen Event automatisch nach dem Erscheinen neuer Umfragen ausgeführt werden. Mit den Ergebnissen dieser Auswertung können verschiedene Actions als Reaktion ausgeführt werden.

7.3. Beispiel: Einsatz eines Raspberry Pi als Eventquelle

Als ein einfaches Beispiel zum Einsatz und Erzeugen eines externen, atomaren Events wird der kreditkartengroßer Computer 'Raspberry Pi' verwendet [43]. Da ein Raspberry Pi üblicherweise mit einer Linux Distribution betrieben wird (z. B. Raspbian, Arch Linux), kann das beschriebene Verfahren unter jeder anderen Linux Distribution angewendet werden.

Vorgehensweise:

1. Eine externe Eventquelle wird unter 'Event- & Datasources' in the Event-Framework-Control registriert.

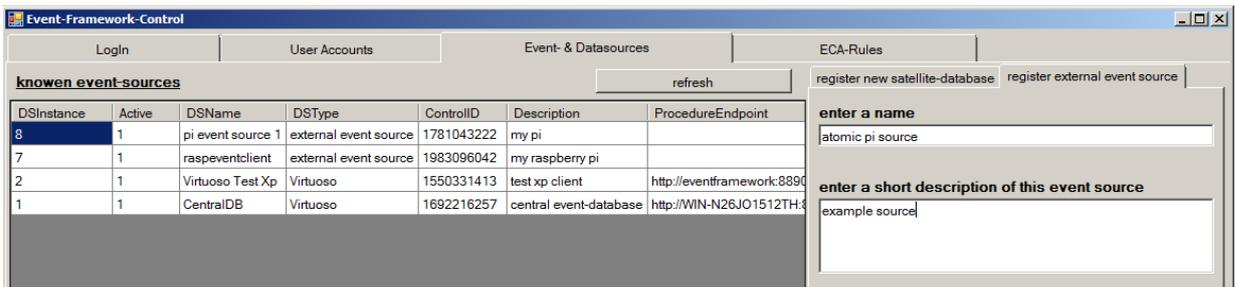


Abbildung 10: Registrierung einer externen Eventquelle

2. Ein externes Event wird unter Verwendung dieser Eventquelle registriert. Nach der Registrierung wird die SOAP-Nachricht angezeigt, welche zum Auslösen des neuen Events auf der Zentraldatenbank, von der Eventquelle versendet werden muss.

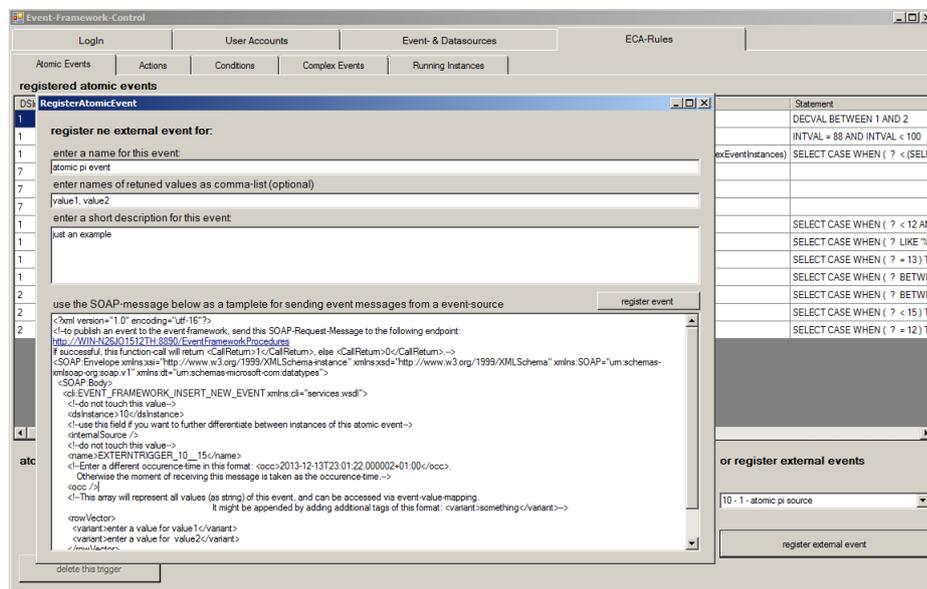
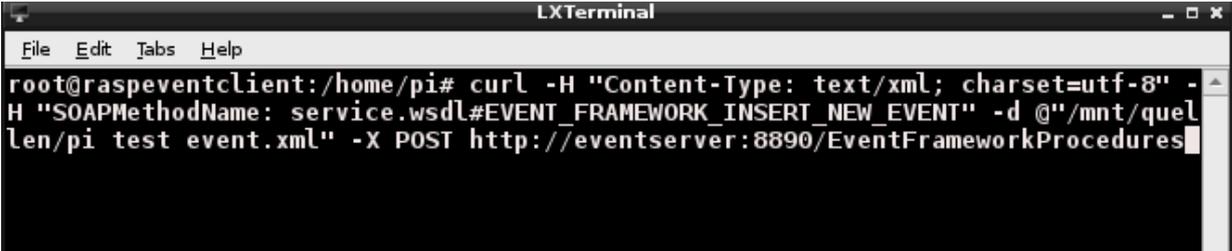


Abbildung 11: Registrierung eines externen Events

Listing 22: SOAP-Nachricht für ein externes atomares Event

```
<?xml version="1.0"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http
://www.w3.org/1999/XMLSchema" xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1" xmlns
:dt="urn:schemas-microsoft-com:datatypes">
  <SOAP:Body>
    <cli:EVENT_FRAMEWORK_INSERT_NEW_EVENT xmlns:cli="services.wsdl">
      <dsInstance>10</dsInstance>
      <internalSource />
      <name>EXTERNTRIGGER_10__15</name>
      <occurrence />
      <rowVector>
        <variant>enter a value for value1</variant>
        <variant>enter a value for value2</variant>
      </rowVector>
    </cli:EVENT_FRAMEWORK_INSERT_NEW_EVENT>
  </SOAP:Body>
</SOAP:Envelope>
```

3. Nachdem man die angezeigte SOAP-Nachricht in einer XML-Datei auf dem Raspberry Pi abgespeichert hat, kann diese jederzeit mit dem folgenden Terminal-Kommando an die Zentraldatenbank gesendet werden und somit das registrierte atomare Event auslösen:



```
LXTerminal
File Edit Tabs Help
root@raspeventclient:/home/pi# curl -H "Content-Type: text/xml; charset=utf-8" -
H "SOAPMethodName: service.wsdl#EVENT_FRAMEWORK_INSERT_NEW_EVENT" -d @"/mnt/quel
len/pi/test/event.xml" -X POST http://eventserver:8890/EventFrameworkProcedures
```

Abbildung 12: Versenden einer SOAP-Nachricht unter Linux mit 'curl'

4. Zum Einfügen von dynamischen Row-Vektor-Werten muss diese Nachricht entsprechend angepasst werden.

Dieses Beispiel soll verdeutlichen, wie einfach es ist, mit externen Events, Rohdaten in Form von atomaren Events dem Event-Framework zur Verfügung zu stellen. Das beschriebene Kommando lässt sich leicht in Script-Dateien unterbringen, welche zum automatischen Auslösen eines atomaren Events verwendet werden können. Ähnliche Funktionen stehen unter Windows-Bash zur Verfügung.

7.4. Bekannte Probleme und nächste Schritte

Das vorgestellte Framework wurde hier in einer ersten Version beschrieben. Neben dem bereits Erreichten gibt es daher noch an vielen Stellen Verbesserungsbedarf. Eine kurze Aufzählung soll die wichtigsten Punkte zusammenfassen:

1. Cycling Events: Dabei handelt es sich nicht um Sportveranstaltungen, sondern um komplexe Events, die sich selber auslösen können. Das heißt, diese Events verursachen die Erfüllung einer anderen Eventinstanz. Daraufhin wird eine neue Eventinstanz des gleiches komplexes Event angelegt usw. Dieser Prozess kann auch zeitversetzt auftreten und daher nicht sofort erkennbar sein. Um diese sich selbst erfüllende Kette zu unterbrechen, muss ein Mechanismus implementiert werden, der mögliche automatische Eventzyklen erkennt und diese stoppt. Ein solche Mechanismus ist bisher nicht im Event-Framework implementiert.
2. Definition von atomaren Events für Laien: Zur Zeit ist es nötig, den WHERE-Teil einer SQL, bzw. SPARQL-Anfrage zu ergänzen, um ein atomares Event auf einer Satellitendatenbank zu erstellen. Dieser Prozess sollte in Zukunft auch für Laien, mit Hilfe einer entsprechenden Benutzeroberfläche im Event-Framework-Control, möglich sein.
3. Auswertung von EventSets: Die Boole'sche Struktur von EventSets wird zur Zeit nicht parallel bei dessen Erstellung durch einen Benutzer auf logische Fehler überwacht. Tautologien, Kontradiktionen und andere logische Fehlerquellen werden nicht als solche erkannt.
4. Wie bereits in Kapitel 7.1. erwähnt, sollte verstärkt auf eine sichere Übertragung von SOAP-Nachrichten eingegangen werden. Das Rechtesystem des Event-Framework ist zu erweitern.

7.5. Ausblick

Im Verlauf der Entwicklung dieses Frameworks sind verschiedene neue Konzepte zur Verarbeitung komplexer Events oder Verbesserungsmöglichkeiten der bisherigen Ansätze deutlich geworden. Eine Überarbeitung des Event-Frameworks bezüglich der folgenden Punkte ist bei zukünftigen Weiterentwicklungen zu berücksichtigen.

1. Das vorliegende Framework war am Anfang der Entwicklung als reine Erweiterung einer Virtuoso-Datenbank angedacht. Es stellte sich aber heraus, dass eine Evaluierung von komplexen Events allein durch die Zentraldatenbank verschiedene Probleme birgt (siehe Kapitel 6.3.4.). Dazu kommt, dass der Vorgang des Debuggings von Datenbankprozeduren mühsam und fehleranfällig ist. In zukünftigen Entwicklungsschritten sollte daher die Evaluation in den Event-Service ausgelagert werden.
2. Wie jedes CEP-System wertet auch das Event-Framework einen Strom von Events aus, um daraus komplexe Events zu extrahieren. Anders als Event-Stream-Processing-Systeme wird dabei auf keine Abfragesprache (vgl. Kapitel 2.1.) zurückgegriffen, sondern eine fest implementierte Heuristik eingesetzt, welche die Kontrolle der verwendeten ECA-Regeln implementiert. Um eine größere Flexibilität bei Definition und Auswertung eines Eventstroms zu erreichen, ist von diesem Konzept in Zukunft abzuweichen und wenn möglich, eine Abfragesprache zur Extraktion komplexer Events, zu verwenden. Dies ist besonders im Hinblick auf den ersten Punkt ein nachvollziehbarer Schritt. Denkbar wäre in diesem Zusammenhang eine Kombination aus Event-Stream-Processing (ESP) und ECA-Regeln. Dabei müssen EventStages als Teievents durch ein ESP-Verfahren aus einem Eventstrom extrahiert werden. Diese können dann durch ECA-Regeln, der gewohnten stufenweisen Abarbeitung von komplexen Events zugeführt werden.
Ein Einsatz der Reactive-Extensions-Bibliothek [44] von Microsoft ist dabei eine mögliche Option. Durch sie sollen asynchrone Datenströme (Eventströme) mit der Abfragesprache LINQ auswertbar gemacht werden. Bei einem einheitlichen Format von Events in einem Eventstrom (wie durch das Event-Framework gegeben), kann so auf das Implementieren einer eigenen Abfragesprache verzichtet werden. Darüber hinaus steht der großen Umfang an Funktionen der Reactive-Extensions-Bibliothek zur Verfügung.
3. Es bietet sich an, den Event-Service zu modularisieren und so, vergleichbar zum Framework für RDF-basierte Events des REWERSE I5 Projekts [19], verschiedene Evaluationsmodelle einsetzen zu können.
4. Die Gestaltung des Event-Framework-Controls soll komplett überarbeitet werden. Eine einfache verständliche Oberfläche, die Vereinfachungen von Definitionsvorgängen und der Einsatz auf verschiedenen Betriebssystemen ist wichtig, für einen einfachen Einstieg in das Problem 'komplexe Events'.

Das vorliegende Event-Framework unterliegt einer aktiven Entwicklung, welche diese aufgeführten Punkte einbeziehen wird. Da eine grundlegende Veränderung dieses CEP-Systems von einem reinen regelbasierten System hin zu einem ESP-System in Erwägung gezogen wird, ist ein klares Entwicklungsziel noch nicht festzumachen. Durch die Verwendung von anderen Datenbanken und Triple-Stores soll der Anspruch eines systemunabhängigen Framework für komplexe Events in Zukunft unterstrichen werden.

Literatur

- [1] Artur P. Schmidt. *Endo-Management : Entrepreneurship im Interface des World Wide Web*. Bern [u.a.] : Haupt, 2., überarb. und erw. aufl. edition, 1999.
- [2] Julian Dziki. Das semantic web: Die zukunft des internets. <http://www.netzpiloten.de/das-semantic-web-die-zukunft-des-internets/>.
- [3] Virtuoso universal server. <http://virtuoso.openlinksw.com/>.
- [4] Tim Berners-Lee, Roy Thomas Fielding, and Larry Masinter. Uniform resource identifier (uri): Generic syntax. *Network Working Group*, 66(3986):1–61, 2005.
- [5] Wikipedia. Literal — Wikipedia, the free encyclopedia. <http://de.wikipedia.org/wiki/Literal>. [Online; accessed 11-January-2014].
- [6] Wikipedia. Nosql — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/wiki/NoSQL#Taxonomy>. [Online; accessed 6-January-2014].
- [7] Mikael Berndtsson. Reactive object-oriented databases and cim. In *IN PROC. 5TH INT'L CONF. ON DATABASE AND EXPERT SYSTEM APPLICATIONS*, pages 769–778. Springer-Verlag, 1994.
- [8] David C. Luckham. A short history of complex event processing. Online Reference, 2007.
- [9] F.Kötter F. Burger, P. Debicki. Vergleich von complex event processing-ansätzen für business activity monitoring, 2010.
- [10] Umeshwar Dayal, Barbara T. Blaustein, Alejandro P. Buchmann, Upen S. Chakravarthy, Meichun Hsu, R. Ledin, Dennis R. McCarthy, Arnon Rosenthal, Sunil K. Sarin, Michael J. Carey, Miron Livny, and Rajiv Jauhari. The hipac project: Combining active databases and timing constraints. *SIGMOD Record*, 17(1):51–70, 1988.
- [11] Eric N. Hanson. An initial report on the design of ariel: A dbms with an integrated production rule system. *SIGMOD Record*, 18(3):12–19, 1989.
- [12] Jennifer Widom and Sheldon J. Finkelstein. Set-oriented production rules in relational database systems, 1990.
- [13] S. Chakravarthy, E. Anwar, and L. Maugis. Design and implementation of active capability for an object-oriented database, 1993.
- [14] Mikael Berndtsson and Brian Lings. On developing reactive object-oriented databases, 1992.
- [15] Oscar Díaz, Norman W. Paton, and Peter M. D. Gray. Rule management in object oriented databases: A uniform approach. In Guy M. Lohman, Amílcar Sernadas, and Rafael Camps, editors, *VLDB*, pages 317–326. Morgan Kaufmann, 1991.
- [16] Stella Gatzui and Klaus R. Dittrich. Events in an active object-oriented database system, 1993.
- [17] Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.

- [18] Norman W. Paton. *Active rules in database systems*. Monographs in computer science. Springer, New York, 1999.
- [19] José Júlio Alferes, Michael Eckert, and Wolfgang May. Evolution and reactivity in the semantic web, 2009.
- [20] François Bry and Paula Lavinia P. Átrânjan. Reactivity on the web: Paradigms and applications of the language xchange. *J. of Web Engineering*, 5:2006, 2005.
- [21] OwlIm-SE notifications. <http://owlim.ontotext.com/display/OWLIMv43/OWLIM-SE+Notifications>.
- [22] Soap version 1.2. <http://www.w3.org/TR/soap/>.
- [23] Natanael Arndt. Xodx - konzeption und implementierung eines distributed semantic social network knotens, 2013.
- [24] Wikipedia. Complex event processing — Wikipedia, the free encyclopedia. http://de.wikipedia.org/wiki/Complex_Event_Processing. [Online; accessed 7-January-2014].
- [25] Universität Oldenburg Abteilung Informationssysteme. Odysseus - ein flexibles framework zur erstellung von datenstrommanagementsystemen. <http://www-is.informatik.uni-oldenburg.de/odysseus/>.
- [26] Marco Grawunder Timo Michelsen Daniela Nicklas H.-Jürgen Appelrath, Dennis Geesen. Poster: Odysseus – a highly customizable framework for creating efficient event stream management systems, 2012.
- [27] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.
- [28] Cep in the large. http://www.ipvs.uni-stuttgart.de/abteilungen/vs/forschung/projekte/CEP-in-the-Large?__locale=de.
- [29] David C. Luckham. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. In *Princeton University*, 1996.
- [30] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. Stream: The stanford data stream management system. Springer, 2004.
- [31] George Papamarkos, Alexandra Poulouvasilis, Ra Poulouvasilis, and Peter T. Wood. RdfTL: An event-condition-action language for rdf. In *In Proc. 3rd Int. Workshop on Web Dynamics (in conjunction with WWW2004)*, pages 223–248, 2004.
- [32] Reverse i5 - evolution and reactivity. <http://reverse.net/I5/>.
- [33] Wolfgang May, José Júlio Alferes, and Ricardo Amador. Active rules in the semantic web: Dealing with language heterogeneity. In *In Proc. Int. Conf. on Rules and Rule Markup Languages for the Semantic Web*, pages 30–44. Springer, 2005.
- [34] Mapping sql data to linked data views. <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSSQL2RDF>.

- [35] Wikipedia. Open database connectivity — Wikipedia, the free encyclopedia. <http://de.wikipedia.org/wiki/ODBC>. [Online; accessed 11-January-2014].
- [36] Wikipedia. Interprozesskommunikation — Wikipedia, the free encyclopedia. <http://de.wikipedia.org/wiki/Interprozesskommunikation>. [Online; accessed 11-January-2014].
- [37] Pipeoperationen in .net framework. [http://msdn.microsoft.com/de-de/library/bb762927\(v=vs.110\).aspx](http://msdn.microsoft.com/de-de/library/bb762927(v=vs.110).aspx).
- [38] Orri Erling. Implementing a sparql compliant rdf triple store using a sql-ordbms. <http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSRDFWP/>.
- [39] Orri Erling. Mono - cross platform, open source .net development framework. http://www.mono-project.com/Main_Page/.
- [40] Wikipedia. Web services description language — Wikipedia, the free encyclopedia. http://de.wikipedia.org/wiki/Web_Services_Description_Language. [Online; accessed 11-January-2014].
- [41] Wcf samples. <http://archive.msdn.microsoft.com/netfxsamples/Wiki/Print.aspx?title=WCF%20samples>.
- [42] Methodbase.invoke-methode (object, object[]). [http://msdn.microsoft.com/de-de/library/a89hcwhh\(v=vs.110\).aspx](http://msdn.microsoft.com/de-de/library/a89hcwhh(v=vs.110).aspx).
- [43] Wikipedia. Raspberry pi — Wikipedia, the free encyclopedia. http://de.wikipedia.org/wiki/Raspberry_Pi. [Online; accessed 11-January-2014].
- [44] msdn. The reactive extensions (rx)... <http://msdn.microsoft.com/de-de/data/gg577609.aspx>. [Online; accessed 17-January-2014].

Abbildungsverzeichnis

1.	ECA-Regel-Komponenten und deren korrespondierenden Sprachen (aus [33])	22
2.	Event-Framework-Layer-Model	28
3.	Complex-Event-Ontology	30
4.	Useraccounts und Rechte	57
5.	Verwaltung von Satellitendatenbanken und anderen Datenquellen	58
6.	Definition eines atomaren Events für einen Triple-Store	58
7.	Definition einer Action für einen SOAP-Endpunkt	59
8.	Definition eines komplexen Events	60
9.	Definition eines EventSets	61
10.	Registrierung einer externen Eventquelle	67
11.	Registrierung eines externen Events	67
12.	Versenden einer SOAP-Nachricht unter Linux mit 'curl'	68
13.	Event-Data-Objects	xi

Tabellenverzeichnis

1.	EventFrameworkActions - Actions, Conditions und Queries werden hier gespeichert	vi
2.	EventFrameworkAwaitingEvent - hier werden alle Events hinterlegt, auf die durch eine Eventinstanz gewartet wird - auch als 'Eventwarteschlange' bezeichnet	vi
3.	EventFrameworkComplexEventInstances - alle angelegten Eventinstanzen .	vii
4.	EventFrameworkComplexEvents - Eventdefinitionen komplexer Events . .	vii
5.	EventFrameworkEventToInstanceMapping - Abbildung von Eventinstanzen zu aufgetretenen Events	vii
6.	EventFrameworkConstants - Konstanten des Frameworks werden hier hinterlegt	vii
7.	EventFrameworkDataSources - alle registrierten Datenquellen	viii
8.	EventFrameworkEvents - zentraler Event-Log der Zentraldatenbank - alle aufgetretenen Events sind hier zu finden	viii
9.	EventFrameworkParameterMappings - Abbildung von Parametern auf auszuführende Actions, Conditions oder Queries	ix
10.	EventFrameworkTriggers - alle atomaren Events (Trigger), welche dem Framework bekannt sind	ix
11.	EventFrameworkUsers - alle Benutzer des Frameworks inkl. Rechten	x
12.	EventFrameworkTriggerConditions - Liste aller bekannten Trigger mit ihren Conditions dieser Satellitendatenbank	x
13.	EventFrameworkAtomicEvents - speichert alle Events einer Satellitendatenbank	x

Appendix

A. Tabellen Zentraldatenbank

Hier sind alle Tabellen ausführlich dokumentiert, die der Zentraldatenbank zum Betrieb des Event-Framework hinzugefügt werden.

Tabelle 1: EventFrameworkActions - Actions, Conditions und Queries werden hier gespeichert

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
ActionID	INTEGER	10	No	Yes	PK	ID
Condition	SMALLINT	10	No	No		gibt an ob eine Action (0), Condition (1) oder Query (2) vorliegt
CreatedBy	INTEGER	10	No	No		ID des Users (creator)
WsdAddress	VARCHAR	0	No	No		Wsd-Adresse des SOAP-Endpunktes
EndpointAddress	VARCHAR	0	No	No		SOAP-Endpunkt-Adresse
MethodeName	VARCHAR	0	No	No		Name der Methode oder Funktion
Description	VARCHAR	0	Yes	No		Beschreibung
InternalQuery	VARCHAR	0	Yes	No		veralted
ParamTypes	VARCHAR	0	Yes	No		Parameter Datentyp
ParamDescr	VARCHAR	0	Yes	No		Parameterbeschreibung
ReturnType	VARCHAR	0	Yes	No		Datentyp des Rückgabewertes
ReturnDescr	VARCHAR	0	Yes	No		Beschreibung des Rückgabewertes
Query	VARCHAR	2147483647	Yes	No		Querytext falls eine Query vorliegt
UserName	VARCHAR	0	Yes	No		Auth: Benutzername
Password	VARCHAR	0	Yes	No		Auth: Passwort
X509Certificate	VARCHAR	2147483647	Yes	No		Auth: X509Certificate
X509Password	VARCHAR	0	Yes	No		Auth: X509Password
DSInstance	INTEGER	10	Yes	No		veralted

Tabelle 2: EventFrameworkAwaitingEvent - hier werden alle Events hinterlegt, auf die durch eine Eventinstanz gewartet wird - auch als 'Eventwarteschlange' bezeichnet

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
CEID	INTEGER	10	Yes	No	FK	Die ComplexEventID eines erwarteten komplexen Events
TriggerID	INTEGER	10	Yes	No	FK	Die ID eines erwarteten atomaren Events
SourceCE	INTEGER	10	No	No	FK	Die auslösende Eventinstanz
Recurrences	INTEGER	10	Yes	No		Anzahl der Auftritte dieses Events seit der Initiierung (dem Eintrag in diese Tabelle)
Started	DATETIME	19	No	No		Zeitpunkt der Initiierung
Until	DATETIME	19	Yes	No		unterliegt die aktuelle Stage einer Zeitstriktion, wird hier der exakte Zeitpunkt eingetragen, bis wann dieses Event 'erwartet' wird

Tabelle 3: EventFrameworkComplexEventInstances - alle angelegten Eventinstanzen

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
EventID	INTEGER	10	No	Yes	PK	die ID dieser Instanz
CEID	INTEGER	10	No	No	FK	ComplexEventID der zugehörigen Eventdefinition
EventUri	VARCHAR	0	No	No		die Uri des Events im Kontext des Linked-Date-Views
FirstStageUri	VARCHAR	0	No	No		die Uri der InitialStage dieses Events
CurrentStage	INTEGER	10	No	No		die aktuelle Stage dieser Instanz (Fortschritt des Events)
Started	DATETIME	19	No	No		Zeitpunkt der Initiierung
Finished	DATETIME	19	Yes	No		Zeitpunkt des Auftretens (Abschluss) der Eventinstanz

Tabelle 4: EventFrameworkComplexEvents - Eventdefinitionen komplexer Events

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
CEID	INTEGER	10	No	Yes	PK	die ID - ComplexEventID
Name	VARCHAR	0	No	No		Name des komplexen Events
InitialStage	VARCHAR	0	No	No		die Uri der InitialStage dieses Events im Kontext des Linked-Date-Views
CreatedBy	INTEGER	10	No	No		ID des Benutzers (creator)
Recurrences	INTEGER	10	No	No		Anzahl der vorgesehenen Instanzen dieses Event (future work)
Period	VARCHAR	0	Yes	No		Zeitraum periodischer Instanziierungen (future work)
InitializeAt	DATETIME	19	Yes	No		der erste Instanziierungszeitpunkt
Description	VARCHAR	0	Yes	No		Beschreibung
IsActive	SMALLINT	10	No	No		Eine Instanz dieses Events wird gerade ausgeführt
IsOverlapping	SMALLINT	10	No	No		Event ist als 'überlappend' definiert

Tabelle 5: EventFrameworkEventToInstanceMapping - Abbildung von Eventinstanzen zu aufgetretenen Events

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
EventID	INTEGER	10	No	No	FK	ID des aufgetretenen Events
InstanceID	INTEGER	10	No	No	FK	ID der Eventinstanz
StageNr	INTEGER	10	No	No		Stage-Nummer der Instanz in der das Event aufgetreten ist
CEID	INTEGER	10	No	No	FK	ComplexEventID des komplexen Events der Instanz
Occurence	DATETIME	19	No	No		der Zeitpunkt des Auftretens des Events

Tabelle 6: EventFrameworkConstants - Konstanten des Frameworks werden hier hinterlegt

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
Key	VARCHAR	0	No	No	PK	Schlüssel
Value	ANY	0	No	No		Wert

Tabelle 7: EventFrameworkDataSources - alle registrierten Datenquellen

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
DSInstance	INTEGER	10	No	Yes	PK	ID der Datenquelle
Active	SMALLINT	10	No	No		Datenquelle ist aktiviert
DSName	VARCHAR	0	No	No		Name
DSType	VARCHAR	0	No	No		typ der Datenquelle (z.B. Virtuoso-DB, Triple-Store...)
ControlID	INTEGER	10	No	No		jede Datenquelle erhält eine zufällige ControlID die zur Authentifikation in zwischen Datenquelle und Zentraldatenbank eingesetzt wird
Description	VARCHAR	0	Yes	No		Beschreibung
ProcedureEndpoint	VARCHAR	0	Yes	No		Adresse des SOAP-Endpunktes dieser Datenquelle (falls vorhanden)
SparqlEndpointAddress	VARCHAR	0	Yes	No		Adresse des SPARQL-Endpunktes dieser Datenquelle (falls vorhanden)

Tabelle 8: EventFrameworkEvents - zentraler Event-Log der Zentraldatenbank - alle aufgetretenen Events sind hier zu finden

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
EventID	INTEGER	10	No	Yes	PK	die ID des Events
TriggerID	INTEGER	10	Yes	No		ID des atomaren Events (falls Event atomar ist)
CEID	INTEGER	10	Yes	No		ComplexEVntID (falls Event komplex ist)
CeInstance	INTEGER	10	Yes	No		Die Eventinstanz-ID (falls Event komplex ist)
Occurence	DATETIME	19	No	No		genaue Zeitpunkt des Auftretens
DSInstance	INTEGER	10	Yes	No		die ID der Datenquelle auf der das Event aufgetreten ist
InternalSource	VARCHAR	0	Yes	No		die interne Quelle des Events im Bezug auf die Datenquelle (z.B. Tabelle)
Row	ANY	0	Yes	No		der Row-Vektor des eingetretenen Events

Tabelle 9: EventFrameworkParameterMappings - Abbildung von Parametern auf auszuführende Actions, Conditions oder Queries

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
CEID	INTEGER	10	No	No	FK	ComplexEventID
StageNr	INTEGER	10	No	No		die Nummer der Stage auf der dieser Parameter eingesetzt wird
ActionID	INTEGER	10	No	No	FK	ID der auszuführenden Action (Condition, Query)
ActionNr	INTEGER	10	No	No		Rang der in einer Reihenfolge auszuführenden Actions
ParamNr	INTEGER	10	No	No		Position des Parameters in der Parameterliste
Description	VARCHAR	0	Yes	No		Beschreibung
StaticValue	VARCHAR	0	Yes	No		der statische Parameter (falls statisch)
StaticValueType	VARCHAR	0	Yes	No		der Datentyp des statischen Parameters
ConditionQuery	INTEGER	10	Yes	No		ID (ActionID) der auszuführenden Query
EventValueMap	VARCHAR	0	Yes	No		ein Event-Value-Mapping String

Tabelle 10: EventFrameworkTriggers - alle atomaren Events (Trigger), welche dem Framework bekannt sind

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
TriggerID	INTEGER	10	No	Yes	PK	ID des atomaren Events (Trigger)
TriggerType	VARCHAR	6	No	No		Typ des Triggers (DELETE, INSERT, UPDATE)
TriggerName	VARCHAR	0	No	No		Name (intern vergeben)
AlternativeName	VARCHAR	0	Yes	No		alternativer Name (falls durch Benutzer vergeben)
DSInstance	INTEGER	10	Yes	No		ID der Datenquelle dieses atomaren Events
InternalSource	VARCHAR	0	Yes	No		die interne Quelle des Events im Bezug auf die Datenquelle (z.B. Tabelle)
Values	VARCHAR	0	Yes	No		die Spaltenwerte die als Row-Vektor an dieses Event angeheftet werden
Created	DATETIME	19	No	No		Zeitpunkt der Registrierung
CreatedBy	INTEGER	10	No	No		erstellt durch Benutzer (ID)
Description	VARCHAR	0	Yes	No		Beschreibung
Statement	LONG VARCHAR	2147483647	Yes	No		die Bedingung des Triggers als SQL-Statement

Tabelle 11: EventFrameworkUsers - alle Benutzer des Frameworks inkl. Rechten

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
UserID	INTEGER	10	No	Yes	PK	ID des Benutzers
Name	VARCHAR	0	No	No		Benutzername
Pass	VARCHAR	0	No	No		Passwort
Created	DATETIME	19	No	No		Zeitpunkt der Registrierung
LastLogIn	DATETIME	19	Yes	No		Zeitpunkt des letzten Log-In
SessionNr	INTEGER	10	Yes	No		aktuelle Sessionnummer
Datasources	VARCHAR	0	Yes	No		dieser Nutzer hat Recht für diese Datenquellen
UserAccRight	SMALLINT	10	No	No		dieser Nutzer darf neue Benutzer hinzufügen/ändern
ECADefRight	SMALLINT	10	No	No		dieser Nutzer hat das Recht Events zu erstellen/ändern

B. Tabellen Satellitendatenbank

Tabelle 12: EventFrameworkTriggerConditions - Liste aller bekannten Trigger mit ihren Conditions dieser Satellitendatenbank

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
TriggerName	VARCHAR	0	No	No	PK	Name des atomaren Events (Triggers)
TriggerType	VARCHAR	0	No	No		Triggertyp (DELETE, INSERT, UPDATE)
TableName	VARCHAR	0	No	No		Tabellenname auf der der Trigger operiert
ParamArray	ANY	0	Yes	No		Auflistung von Spaltenwerten, die an Stelle von Parametern in die Condition einzutragen sind
Condition	LONG VARCHAR	2147483647	No	No		die Condition dieses Triggers

Tabelle 13: EventFrameworkAtomicEvents - speichert alle Events einer Satellitendatenbank

Spaltenname	Datentyp	Precision	Nullable	ID	Ind.	Beschreibung
Status	SMALLINT	10	No	No		gibt an ob Event erfolgreich an die Zentraldatenbank übertragen wurde {0,1}
Occurence	DATETIME	19	No	No		gibt den exakten Zeitpunkt des Auftretens dieses Events an
TriggerName	VARCHAR	0	No	No		Name des atomaren Events (Trigger)
InternalSource	VARCHAR	0	Yes	No		interne Quellenangabe (z.B. Tabellenname)
Row	ANY	0	Yes	No		der Row-Vektor dieses Events

C. Event-Data-Objects

Das folgende Klassendiagramm zeigt die verwendeten Datenobjekte die zwischen Event- und Client-Service sowie Event-Framework-Control ausgetauscht werden. Sie spiegeln zu einem großen Teil die Struktur der Konzepte der Complex-Event-Ontology wider.

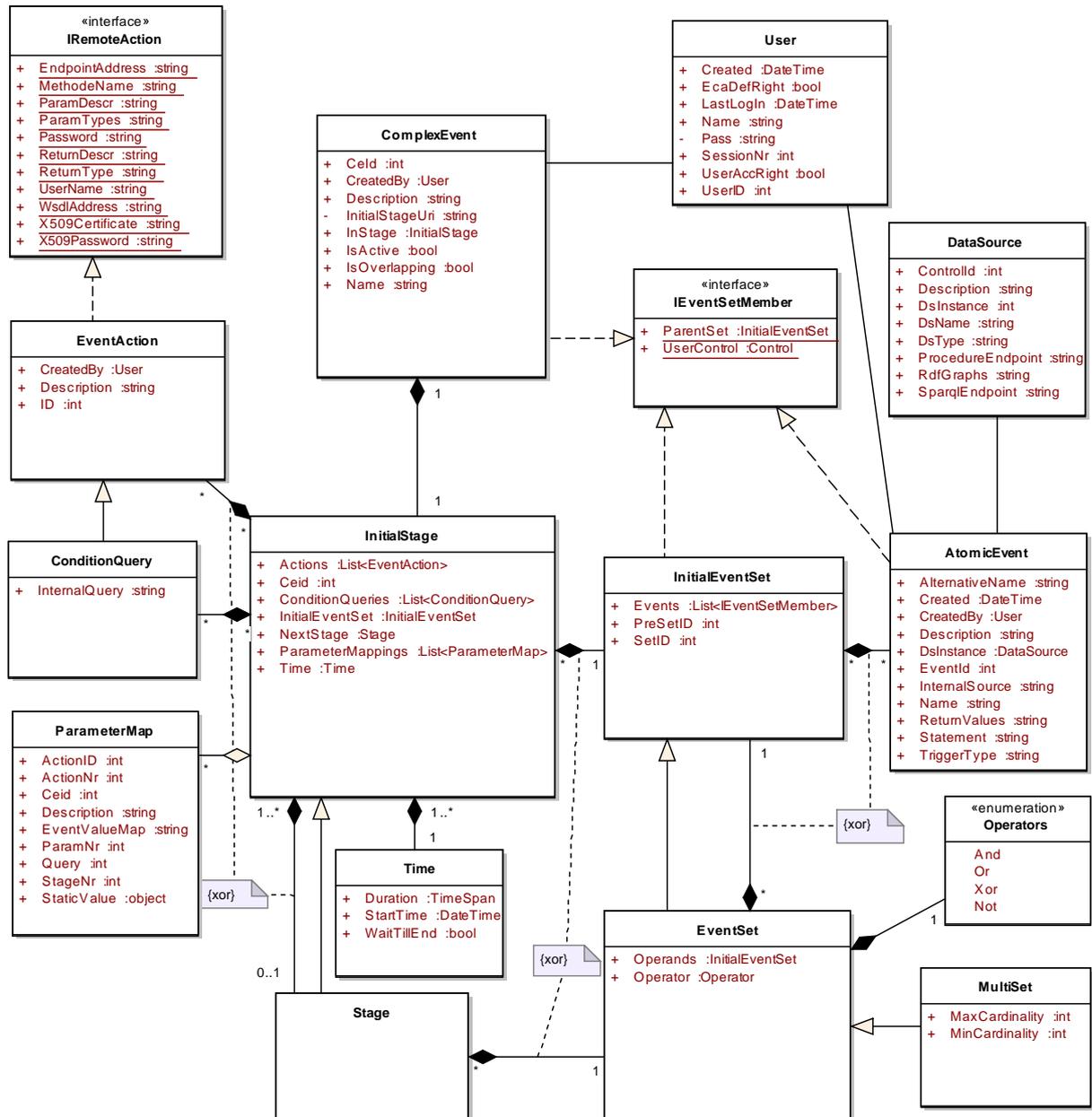


Abbildung 13: Event-Data-Objects

D. Interfaces

In diesem Abschnitt werden alle Schnittstellen für Event- und Client-Service vollständig aufgeführt.

IVirtuosoEventService ist die clientseitige Schnittstelle des Event-Service.

Listing 23: IVirtuosoEventService

```
[ServiceContract]
public interface IVirtuosoEventService
{
    [OperationContract]
    DataTable GetEvents(int minutes);

    [OperationContract]
    DataTable GetEventsBetween(DateTime From, DateTime To);

    [OperationContract]
    DataTable GetUsers(string searchString);

    [OperationContract]
    DataTable GetDatabases(string type);

    [OperationContract]
    DataTable GetAllTriggers(string likeName);

    [OperationContract]
    DataTable GetActionsOrConditions(Activity act);

    [OperationContract]
    string DeleteActionsOrConditions(User admin, int actionID);

    [OperationContract]
    DataTable GetComplexEvents(string likeName = "", int ceid = 0);

    [OperationContract]
    User Login(User currentUser);

    [OperationContract]
    bool CreateNewAccount(User newUser, User admin);

    [OperationContract]
    bool ResetUserpassword(string userName, string newPass, User admin);

    [OperationContract]
    bool UpdateUserAccount(User user, User admin);

    [OperationContract]
    string RegisterTrigger(AtomicEvent trigger, User admin);

    [OperationContract]
    string SetNewSqlTrigger(AtomicEvent trigger, User admin);

    [OperationContract]
    string SetNewRdfTrigger(AtomicEvent trigger, User admin);

    [OperationContract]
    string[] GetSchemaTables(int dbInstance = 1);

    [OperationContract]
    string[] GetTriggersOfTable(string tableName, int dbInstance = 1);

    [OperationContract]
    string GetSupportedDBs();

    [OperationContract]
    string GetRemoteProcedureEndpoint(int dbInstance);

    [OperationContract]
    string RegisterNewRemoteDataSource(DataSource ds, User admin);
}
```

```

[OperationContract]
string [][] ExecuteTestSqlQuery(string queryString, int dbInstance = 1);

[OperationContract]
string [] GetColumnsOfRemoteTable(int dbInstance, string tableName);

[OperationContract]
int InsertNewAction(EventOntology.EventAction action, User admin);

[OperationContract]
int InsertNewCondition(ConditionQuery condition, User admin, Activity act);

[OperationContract]
string [] GetGraphs(int dbInstance);

[OperationContract]
string UpdateAction(EventOntology.EventAction action, User admin);

[OperationContract]
string DropTrigger(AtomicEvent trigger, User admin);

[OperationContract]
string InsertComplexEvent(User admin, ComplexEvent ev);

[OperationContract]
string UpdateComplexEvents(User admin, ComplexEvent newEv, ComplexEvent oldEv);

[OperationContract]
DataTable GetStagesAndSets(string initialStageUri, int stageNr);

[OperationContract]
string InsertCeInstance(User admin, int ceid, string initialStageUri, int startAtStage
    = 0, bool isOverlapping = false);

[OperationContract]
string DeleteComplexEvent(User admin, ComplexEvent ev);

[OperationContract]
string ActivateComplexEvent(User admin, ComplexEvent ev);

[OperationContract]
string DeActivateComplexEvent(User admin, ComplexEvent ev);

[OperationContract]
DataTable GetParamMappings(int ceid);

[OperationContract]
DataTable GetComplexEventInstances(int ceid = 0);

[OperationContract]
string AbortComplexEventInstances(int ceid);

[OperationContract]
string DeleteDataSource(User admin, int instanceID);

[OperationContract]
Time GetTimeOfStage(string initialStageUri, int stageNr);

[OperationContract]
string GetStageUriFromEvent(string eventUri);

[OperationContract]
string DeActivateDB(User admin, DataSource ds);

[OperationContract]
bool CheckIfDsExists(string endpoint);

[OperationContract]
int GetControlId(int dsInstance);

[OperationContract]
DataTable GetActionById(int actionID);

```

```
}
```

IVirtuosoExtentionService ist die datenbankseitige Schnittstelle des Event-Service.

Listing 24: IVirtuosoExtentionService

```
[ServiceContract]
public interface IVirtuosoExtentionService
{
    //methodes which are called from Virtuoso use RPC as OperationFormatStyle and
    OperationFormatUse
    [OperationContract]
    void CallForAction(int controlID, int actionID, int eventInstance, string[] parameters
        );

    [OperationContract]
    object CheckCondition(int controlID, int actionID, int eventInstance, string[]
        parameters);

    [OperationContract]
    void SendSmtpMail(string smtpHost, int port, string username, string password, string
        from, string to, string subject, string body);
}
```

Die IEventClient-Schnittstelle wird durch den Client-Service implementiert.

Listing 25: IEventClient

```
public interface IEventClient
{
    [OperationContract]
    DataTable GetEvents(int minutes);

    [OperationContract]
    DataTable GetUsers(string searchString);

    [OperationContract]
    User LogIn(User currentUser);

    [OperationContract]
    bool CreateNewAccount(User newUser);

    [OperationContract]
    bool ResetUserpassword(string userName, string newPass);

    [OperationContract]
    bool UpdateUserAccount(User user);

    [OperationContract]
    string RegisterTrigger(AtomicEvent trigger);

    [OperationContract]
    string SetNewSqlTrigger(AtomicEvent trigger);

    [OperationContract]
    string SetNewRdfTrigger(AtomicEvent trigger);

    [OperationContract]
    string[] GetSchemaTables(int dbInstance = 1);

    [OperationContract]
    string[] GetTriggersOfTable(string tableName, int dbInstance = 1);

    [OperationContract]
    DataTable GetDatabases(string type);

    [OperationContract]
}
```

```

string GetSupportedDBs();

[OperationContract]
string GetRemoteProcedureEndpoint(int dbInstance);

[OperationContract]
string RegisterNewRemoteDataSource(DataSource ds);

[OperationContract]
string[][] ExecuteTestSqlQuery(string queryString, int dbInstance = 1);

[OperationContract]
string[] GetColumnsOfRemoteTable(int dbInstance, string tableName);

[OperationContract]
int InsertNewAction(EventAction action);

[OperationContract]
string DeleteActionsOrConditions(int actionID);

[OperationContract]
int InsertNewCondition(ConditionQuery condition, Activity act);

[OperationContract]
DataTable GetAllTriggers(string likeName);

[OperationContract]
string[] GetGraphs(int dbInstance);

[OperationContract]
DataTable GetActionsOrConditions(Activity act);

//[OperationContract]
//object CallActionMethode(int ActionID, object[] Parameters);

[OperationContract]
string UpdateAction(EventAction action);

[OperationContract]
DataTable GetEventsBetween(DateTime From, DateTime To);

[OperationContract]
string DropTrigger(AtomicEvent trigger);

[OperationContract]
[XmlSerializerFormat(Style = OperationFormatStyle.Rpc, Use = OperationFormatUse.
    Encoded)]
DateTime StartCalc(int inInt, string aString);

[OperationContract]
[XmlSerializerFormat(Style = OperationFormatStyle.Rpc, Use = OperationFormatUse.
    Encoded)]
int CheckThis(int inInt);

[OperationContract]
DataTable GetComplexEvents(string likeName = "", int ceid = 0);

[OperationContract]
string InsertComplexEvent(ComplexEvent ev);

[OperationContract]
string UpdateComplexEvents(ComplexEvent newEv, ComplexEvent oldEv);

[OperationContract]
int GetNextUriNr(string IriClass);

[OperationContract]
DataTable GetStagesAndSets(string initialStageUri, int stageNr);

[OperationContract]
string InsertCeInstance(int ceid, string initialStageUri, int startAtStage = 0, bool
    isOverlapping = false);

```

```

[OperationContract]
string DeleteComplexEvent(ComplexEvent ev);

[OperationContract]
string ActivateComplexEvent(ComplexEvent ev);

[OperationContract]
string DeActivateComplexEvent(ComplexEvent ev);

[OperationContract]
DataTable GetParamMappings(int ceid);

[OperationContract]
DataTable GetComplexEventInstances(int ceid = 0);

[OperationContract]
string AbortComplexEventInstances(int ceid);

[OperationContract]
string DeleteDataSource(int instanceID);

[OperationContract]
Time GetTimeOfStage(string initialStageUri, int stageNr);

[OperationContract]
string GetStageUriFromEvent(string eventUri);

[OperationContract]
string DeActivateDB(DataSource ds);

[OperationContract]
bool CheckIfDsExists(string endpoint);

[OperationContract]
int getControlId(int dsInstance);
}

```

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, d. _____
Ort, Datum

Unterschrift