Diplomarbeit im Studiengang Informatik

Automatenbasierte Detektion von Composite Events gemäß Snoop in XML-Umgebungen

vorgelegt von
Sebastian Herbert Spautz

eingereicht bei
Prof. Dr. Wolfgang May
Institut für Informatik
Universität Göttingen

17. März 2006

Institut für Informatik
Technische Universität Clausthal

Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Sarstedt, 17. März 2006 Sebastian Herbert Spautz

Danksagung

Diese Arbeit konnte nur durch die Menschen in meinem Umfeld entstehen:
Danken muss ich daher zu allererst meiner Ehefrau, die meine geistige Abwesenheit ertragen und meine Vorträge über sich ergehen lassen hat. Darüber hinaus spornte Sie mich an, wann immer mir die Ideen ausgingen oder der Ehrgeiz nachließ. Mit Erfolg, wie man sieht.
Auch Herrn Prof. Dr. May und seinem Team von der Universität Göttingen gebührt mein Dank. Sie haben mich an ihrer Arbeit teilhaben lassen und sind mir bei allen fachlichen Fragen und Problemen sehr hilfreich gewesen.
Meiner Kommilitonin Elke von Lienen danke ich, die immer für einen Meinungs- und Wissensaustausch zu haben war; und das, obwohl sie selbst genug zu tun hatte.
Ebenfalls danken muss ich Prof. Dr. Ecker, Prof. Dr. Joubert, Prof. Dr. Kupka, Dr. Kemnitz und Dr. Reuter sowie den Assistenten und Tutoren an der TU-Claushtal, weil sie mir die Fundamente und Grundmauern meines Wissens errichtet haben.
Zu letzt aber nicht an letzter Stelle danke ich meinen Eltern, die mir die Gaben in die Wiege legten, alles zu erreichen, was ich erreichen möchte.

Danke!

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das vorliegende Buch ist die schriftliche Aufbereitung einer Diplomarbeit zum Thema „Automatenbasierte Detektion von Composite Events gemäß Snoop in XML-Umgebungen“. Es entstand als Abschluss eines Informatik-Studiums. Vorangegangen sind dabei einige theoretische Überlegungen und praktische Umsetzungen zu obigem Thema. Die Ergebnisse, die daraus resultierten, werden im vorliegenden Buch vorgestellt und erläutert.

Die Arbeit gliedert sich in die Forschungen der Projektgruppe REWERSE[1] ein. Diese Forschungsgruppe befasst sich mit dem sogenannten Semantic Web. In Kapitel 2 wird genauer auf diese Integration eingegangen.

1.1 Voraussetzungen

Um die Ausführungen dieses Buches zu verstehen, ist es nötig, dass der Leser mit den grundlegenden Strukturen der „Extended Markup Language“ (XML) [W3C04] vertraut ist. Ebenfalls nötig ist das Verständnis von XSLT [W3C99b] und XPath [W3C99a] (jeweils in der Version 1.0), da mittels dieser Technologien der Großteil der praktischen Umsetzung vorgenommen wird. Für die Realisierung der Netzwerkkommunikation wird die Programmiersprache Java [Sunb] verwendet. Daher ist es für Kapitel 7 hilfreich, wenn man diese Sprache, insbesondere die Java-Servlets [Suna], kennt.

Mit oben genannten Technologien wird die praktische Realisierung umgesetzt. In den eher theoretischen Bereich fällt das Verständnis von Endlichen Automaten. Basierend auf dieser, in der theoretischen Informatik häufig verwendeten, Konstrukte wird die Datenstruktur für den Algorithmus aus dieser Arbeit spezifiziert. Dabei geht diese Arbeit nicht über die recht einfachen „Deterministischen Endlichen Automaten“ (DEA) hinaus. Wann immer im Folgenden von Auto-

[1] Internet: http://www.rewerse.net/

maten gesprochen wird, sind solche DEA's gemeint.

1.2 Zielsetzung dieser Arbeit

Diese Arbeit hat zum Ziel einen Algorithmus zu formulieren, der Events erkennen kann. Dabei ist mit einem Event nicht das einfache Eintreten eines Ereignisses gemeint, sondern die Kombination mehrerer solcher Vorkommnisse zu einem zusammengesetzten Event. Die zu detektierenden Events werden durch gewisse Strukturen spezifiziert, die als Eventausdrücke bezeichnet werden. Diese Ausdrücke setzen sich aus Operatoren und sogenannten atomaren Events zusammen. Zur Verdeutlichung hier ein kleines Beispiel:

Beispiel 1.1 *„(A und B) oder C“*
Diese Zeile bringt zum Ausdruck, dass entweder das atomare Event C auftreten muss oder alternativ die atomaren Events A und B. Die Reihenfolge, in der A und B auftreten, ist dabei irrelevant.

In diesem Beispiel sind A, B und C die atomaren Events. Die Operatoren dagegen sind „und“ und „oder“. Es gibt verschiedene Algebren, die solche Operatoren definieren. In dieser Arbeit wird die Algebra Snoop [Mis91, CKAK94] verwendet. Ein Algorithmus, der solche Events erkennen kann, wird hier Detektionsalgorithmus genannt.

Um die Eventausdrücke verarbeiten zu können, müssen sie natürlich in einer maschinenlesbaren Form vorliegen. Gemäß dem Thema der Arbeit werden zur Darstellung der Ausdrücke Endliche Automaten verwendet. Wie diese Darstellung genau aussieht, muss durch diese Arbeit geklärt werden, bevor die Implementierung des Detektionsalgorithmus vorgenommen werden kann.

Nach der abstrakten Entwicklung des Algorithmus soll dieser auch noch implementiert werden. Auf diese Weise wird auch gleich die Realisierbarkeit der erarbeiteten Ergebnisse geprüft. Ein Punkt, der bei der Programmierung beachtet werden muss, ist die Integration in das ECA-Framework der REWERSE-Gruppe. Da diese Gruppe auf den Einsatz von XML setzt, um sich im „Semantic Web“ zu bewegen, soll auch dieses Programm XML und darauf aufsetzende Technologien wie XSLT verwenden.

1.3 Struktur dieser Arbeit

In den folgenden Kapiteln wird einiges gesagt, das zum Verständnis der Problemlösung nötig ist. Aber der Großteil beschäftigt sich mit der Lösung selbst. Der erste Teil der Arbeit führt in die Grundlagen ein, auf denen die vorliegende Arbeit beruht. Der zweite und umfangreichere Teil umfasst dann die gesamte Realisierung.

In Kapitel 2 dreht es sich um das Framework, in dessen Rahmen diese Arbeit angesiedelt ist. Betrachtet Kapitel 2 die externen Bedingungen, so umfasst Kapitel 3 die grundlegenden Interna.

Dabei geht es um die Eventalgebra Snoop. Besonders deren Operatoren und ihre Semantik werden vorgestellt. Darüber hinaus wird auf die zeitlichen Anforderungen und Bedingungen beim Detektieren von Events eingegangen, da diese eine wesentliche Rolle beim Entwurf des Algorithmus spielten. Danach werden in Kapitel 4 zwei XML-Sprachen vorgestellt. Diese stellen Eventausdrücke in deklarativer sowie in Automatenschreibweise dar. Die deklarative Schreibweise dient dem Benutzer als Eingabeformat. Die automatenbasierte Repräsentation der Daten ist hingegen für die Verwendung durch den Algorithmus entworfen worden. Die Kapitel 5 und 6 behandeln den Algorithmus und seine Implementierung in XSLT. Dabei ist Kapitel 5 dem Entwurf gewidmet und Kapitel 6 der Realisierung. Kapitel 7 stellt eine in Java geschriebene Plattform für den Algorithmus vor. Diese Plattform kapselt die Eventdetektion und stellt ihre Funktionen über Schnittstellen in einem Netz zur Verfügung. Damit wird es möglich diese Arbeit in einer Umgebung, wie sie in Kapitel 3 vorgestellt wird, einzubetten.

Kapitel 2

ECA-Framework

Diese Arbeit entsteht im Rahmen der Arbeitsgruppe „I5 – Evolution and Reactivity“ des Projekts REWERSE. REWERSE beschäftigt sich mit „Reasoning on the Web“. Dabei wird das bestehende Internet zu einem sogenannten „Semantic Web“ erweitert, indem die Inhalte mit Metainformationen versehen werden. Mittels dieser Metainformationen wird es möglich vielfältige Beziehungen der Inhalte automatisch zu erschließen. Die Gruppe I5 befasst sich in diesem Zusammenhang mit dem Reaktionsvermögen im Semantic Web. Es wird untersucht, ob und wie Änderungen an den Inhalten und Daten in einem heterogenen Netzwerk erkannt werden können und wie auf solche Änderungen reagiert werden kann.

Eine Möglichkeit das Reaktionsvermögen eines Systems zu modellieren sind sogenannte ECA-Regeln. ECA steht dabei für Event, Condition und Action, also Ereignis, Bedingung und Aktion. Eine solche ECA-Regel spezifiziert somit das Verhalten eines Systems, wenn ein bestimmtes Ereignis eingetreten ist und zusätzlich eine Bedingung erfüllt ist. Durch die Kombination solcher Regeln lassen sich, je nach Komplexität der drei Komponenten, mächtige Systeme realisieren, die mit ihrer Umgebung auf vielfältige Weise interagieren. Am Beginn einer Reaktion steht immer ein Ereignis. Diese Ereignisse zu erkennen ist eine wesentliche Aufgabe eines ECA-Systems.

In einem Netzwerk sind der Verursacher eines Events und derjenige, der darauf reagieren möchte, üblicherweise verschieden. Daher muss eine Kommunikation zwischen den einzelnen Netzknoten erfolgen. Für eine solche Kommunikation sind zwei Voraussetzungen nötig: eine gemeinsame Sprache und ein Transportsystem für Texte in dieser Sprache.

Für die gemeinschaftliche Sprache werden in REWERSE sämtliche Daten- und Austauschformate in XML verfasst. Das hat mehrere Vorteile. XML-Formate lassen sich unter anderem leicht ineinander überführen. Ebenfalls wichtig ist, dass sich einzelne XML-Dialekte kombinieren lassen. Damit ist es möglich speziell angepasste Sprachen für jede Art von Ereignissen, Bedingungen und Aktionen zu verwenden. Diese Sprachen lassen sich dann durch die Verwendung von

Namespaces und einem verbindenden Markup zu einer ECA-Regel kombinieren. Darüber hinaus existieren erprobte Methoden um XML-Daten zu verarbeiten. Ein XML-Markup zur Darstellung von ECA-Regeln wird in [MAA05] angerissen.

```
<eca:rule>                                                    1
  <eca:event>                                                 2
    <!-- Event-Spezifikation-Language -->                     3
  </eca:event>                                                4
  <eca:condition>                                             5
    <!-- Condition-Spezifikation-Language -->                 6
  </eca:condition>                                            7
  <eca:action>                                                8
    <!-- Action-Spezifikation-Language -->                    9
  </eca:action>                                               10
</eca:rule>                                                   11
```

Listing 2.1: ECA-Rule

Anstelle der Kommentare wird das Markup der jeweiligen Sprache verwendet. Wenn im Folgenden von diesem Markup die Rede ist, wird immer der Namespace „eca“ für dessen Elemente verwendet.

Für die Kommunikation kommt SOAP zur Anwendung. SOAP ist eine XML-basierte Technik und passt damit zum gewählten Datenformat. Die Daten werden in SOAP in einem speziellen XML-Markup verpackt und übermittelt, indem sie mit bestehenden Protokollen wie dem Hypertext Transfer Protokoll (HTTP) kombiniert werden.

Nach der allgemeinen Betrachtung des Framework wird jetzt darauf eingegangen welche Auswirkungen dieses auf die Entwicklung eines Algorithmus zur Detektion von Composite Events hat. Natürlich müssen die im Framework eingesetzten Technologien mit denen kompatibel sein, mit denen der Algorithmus umgesetzt wird. Das betrifft vor allem den Datenaustausch und die Kommunikation. Die folgenden Abschnitte gehen auf diese Punkte ein, um aufzuzeigen welche Art von Daten der Algorithmus verarbeiten muss und wie er sich in das Framework integriert.

2.1 Events

Da diese Arbeit sich mit dem Erkennen von Events beschäftigt, soll erst einmal geklärt werden, was ein Event alles sein kann. Die einfachste Art sind sogenannte „Atomare Events“. Diese Ereignisse treten in einzelnen Knoten des Netzes auf. Je nach Art des Knoten sind dabei andere Events möglich. Welche atomaren Events insgesamt möglich sind, hängt dabei von der Anwendungsdomäne ab, in der das ECA-System betrieben wird.

Beispiel 2.1 (Atomare Events)
In einem System zu Flugreisen sind atomare Events beispielsweise das Auftreten einer Buchung oder das Absagen eines Fluges. Ein solches Event hat jedesmal wenn es auftritt bestimmte Werte.

Wenn ein Flug gestrichen wird, wäre ein solcher Wert die Flugnummer des betroffenen Fluges, beim Buchen einer Reise die Kundennummer und der Name der Reise.

Zur Darstellung der atomaren Events wird XML verwendet. Dabei wird jedes Event mit einem Element beschrieben. Diese Elemente können, wie in XML üblich, Attribute und Kind-Elemente enthalten. Das verwendete Markup ist dabei beliebig, sollte aber für jedes Event eindeutig und damit unterscheidbar sein.

Beispiel 2.2 (Darstellung atomarer Events)
Ein Event zum Buchen eines Fluges könnte so aussehen:
<travel:booking customernr="187589" travelnr="935639"/>.
Im Fall der Streichung eines Fluges ist das Markup vielleicht das folgende:

```
<travel:cancel-flight>
    <travel:code>A-3476</travel:code>
</travel:cancel-flight>
```

Die Detektion von atomaren Events kann grundsätzlich auf zwei Arten erfolgen. Entweder kann jeder Netzwerkknoten bei Änderungen seines internen Zustands entsprechende Events verschicken oder die interessierten Knoten fragen diesen Zustand regelmäßig ab und generieren aus den Änderungen die gewünschten Events. Mittels eines Event-Brokers ist eine Kombination beider Vorgehensweisen möglich. Der Broker ermittelt die Events und leitet sie an andere Knoten im Netzwerk weiter.

Eine weitere Form von Events sind die „Composite Events“. Solche zusammengesetzten Events entstehen durch die Kombination von atomaren Events. Mit diesen Events lassen sich auch komplexere Auslöser für Aktionen in einer ECA-Umgebung realisieren.

Beispiel 2.3 (Zusammengesetzte Events)
Ein Composite Event würde beispielsweise durch „Ein Kunde hat einen Flug gebucht und danach wurde dieser Flug abgesagt.“ beschrieben. In diesem Event werden zwei atomare Events verknüpft. Zuerst muss das Ereignis „eine Buchung wurde vorgenommen“ eintreten. Wenn dies geschehen ist soll auch noch das Event „ein Flug wurde abgesagt“ eintreten. Wenn zwei solcher Events den selben Wert für die Flugnummer haben, bilden sie ein zusammengesetzte Event zu dem oben angegebenen Ausdruck.

Die Art der Kombinationsmöglichkeiten hängt davon ab, welche Eventalgebra verwendet wird. Eine Algebra ist nötig, damit sich auch zusammengesetzte Events kombinieren lassen. Um zusammengesetzte Events in einer ECA-Regel zu verwenden, wird statt eines primitiven Events ein Eventausdruck gemäß der gewählten Algebra angegeben. Dieser Ausdruck enthält die atomaren Events und ihre Verknüpfungsoperatoren, sogenannte Eventoperatoren. Durch den Einsatz von

Eventausdrücken wird es möglich komplexeste Zusammenhänge zwischen atomaren Events darzustellen und damit auch darauf zu reagieren.

Zum Erkennen zusammengesetzter Events bedarf es mehr als einzelne Netzwerkknoten zu beobachten, denn es können auch atomare Events aus mehreren Knoten kombiniert werden. Wie genau eine Detektion von kombinierten Events aussehen kann, ist – bei Verwendung der Eventalgebra Snoop (Kap. 3) – Hauptbestandteil dieser Arbeit. Dabei wird davon ausgegangen, dass die Snoop-Engine automatisch alle atomaren Events erhält. Wenn die Entwicklung des Frameworks weiter fortgeschritten ist, kann aber leicht dazu übergegangen werden eine explizitere Kommunikation z.B. mit Event-Brokern zu realisieren (siehe Abschnitt 2.3).

2.2 Variablen

Die drei Komponenten einer ECA-Regel sollen nicht völlig unabhängig voneinander operieren, sondern auf die Ergebnisse der vorangegangen Komponenten zugreifen können. Damit ist es möglich z.B. in der Aktion die Werte der Events zu verwenden. Da die Ereignisse, Bedingungen und Aktionen einer ECA-Regel getrennt behandelt werden, stellt das Framework Variablen bereit. An solche Variablen können alle Bestandteile innerhalb der ECA-Regeln gebunden werden. Einmal gebunden, kann später darauf zurückgegriffen werden.

Beispiel 2.4 (Verwendung von Variablen)
Um die Bedeutung von Variablen zu verdeutlichen, wird in diesem Beispiel eine ECA-Regel vorgestellt. Anhand dieser Regel wird gezeigt wie sich Variablen aus dem Eventteil auf die Bedingungs- und Aktionskomponente auswirken.

Eine Fluggesellschaft bietet ihren Premium-Kunden an sie über Flugausfälle per E-Mail und SMS zu informieren. Um diesen Dienst zu ermöglichen, registriert die Gesellschaft die folgende ECA-Regel:

<eca:rule>
<eca:event>
Ein Kunde bucht einen Flug. (Variable Kunde=
Kundennummer; Variable Flug=Flugnummer)
Und
Der Flug mit der Nummer aus der Variablen
'Flug' wird gestrichen.
</eca:event>
<eca:condition>
Der Kunde mit der Kundennummer aus der Variablen
'Kunde' ist Premium−Kunde.
</eca:condition>
<eca:action>
Benachrichtige den Kunden mit der Kundennummer aus

der Variablen 'Kunde' per SMS und E−Mail.
</eca:action>
</eca:rule>

Im Eventteil werden gleich im ersten Event die beiden Variablen „Kunde“ und „Flug“ an Werte gebunden. Die Variable „Flug“ wird noch im Eventteil verwendet um festzulegen, auf welche Flugstreichung reagiert werden soll. Wenn das zusammengesetzte Event detektiert wurde, werden die Variablenbindungen über die ECA-Engine an die Bedingungskomponente weitergegeben. Dort wird für die in „Kunde“ angegebene Kundennummer überprüft, ob es sich um einen Premium-Kunden handelt. Wenn das der Fall ist, gehen die Variablen an die Aktionskomponente weiter. Diese sendet eine E-Mail und SMS an die wiederum durch „Kunde“ festgelegte Person.

Für die Detektion von Events sind vor allem diejenigen Variablen von Interesse, die innerhalb eines Eventausdrucks auftreten. Solche Variablen können an einzelne Werte des Events gebunden werden. Wie im obigen Beispiel 2.4 zu sehen ist, kann eine Variable auch mehrmals innerhalb eines Eventausdrucks vorkommen. Dabei wird sie nur beim ersten Auftreten gebunden. Bei jedem weiteren Vorkommnis im Ausdruck wird dann der Wert der Variable mit dem an der entsprechenden Stelle im atomaren Event vorliegenden Wert verglichen. Nur wenn beide Werte übereinstimmen, kommt das atomare Event für die Detektion in Frage.

Von dieser Arbeit werden zwei Möglichkeiten Variablen anzugeben unterstützt. Einerseits kann ein Element <eca:variable> aus dem ECA-Markup verwendet werden. Damit wird das dem Inhalt eines solchen Elements entsprechende Fragment eines Events an die Variable gebunden. Andererseits kann eine Variable an den Wert eines Attributes gebunden werden, indem dessen Wert ein $-Zeichen vorangestellt wird. Der Wert eines solchen Attributes wird als Variablenname aufgefasst.

Beispiel 2.5 (Variablen in Events)
Die Kundennummer kann zum Beispiel verwendet werden, um dem Kunden, der einen Flug gebucht hat, Veranstaltungshinweise am Zielort seiner Reise zukommen zu lassen. Mit <travel:booking customer=”$customer” travelnr=”935639”/> als Event führt jede Buchung für die Reise mit der Nummer 935639 zum Auslösen der Regel. Dabei wird die Variable „customer“ mit dem jeweiligen Wert des Attributes Customer verknüpft.

Eine andere Schreibweise des Events demonstriert die Verwendung des <eca:variable>-Elements:

<travel:booking>
<eca:variable name="customer">
<travel:customer/>
</eca:variable>
<travel:travel>935639</travel:travel>
</travel:booking>

Wenn eine Buchung vorgenommen wurde, wird im entsprechenden Event ein Element<customer> gesucht und mit seinem gesamten Inhalt an die Variable „Customer“ gebunden.

2.3 Struktur & Schnittstellen

Innerhalb von der I5-Gruppe wird, um Reaktivität im Semantic Web zu realisieren, ein Framework auf Basis von ECA-Regeln erstellt. Teil dieses Frameworks ist eine Struktur, mit der die Aufgaben eines ECA-Systems wahrgenommen werden können. Diese Struktur trennt die Komponenten Event, Condition und Action voneinander. Damit die so entstehenden Teile zusammenarbeiten können, werden auch Datenformate und Kommunikationsprotokolle erarbeitet. In einer speziellen Einheit, der ECA-Engine, werden Regeln angelegt. Diese sorgt dann dafür, dass die Events detektiert werden, die Bedingungen überprüft und die Aktionen durchgeführt werden, indem sie diese Aufgaben über das Netzwerk an andere Einheiten verteilt. Diese greifen auf die benötigten Datenquellen zu und melden die Ergebnisse an die ECA-Engine zurück. Anhand von Abbildung 2.1 bekommt man eine gewisse Vorstellung wie die ECA-Regeln im REWERSE-Framework verarbeitet werden.

In Abbildung 2.1 liegt das Hauptaugenmerk auf der Verarbeitung der Event-Komponente. Die Condition- und Action-Komponente werden nur durch die beiden Kästen rechts oben angedeutet. Die senkrechte Anordnung von weißen Kästen stellt den Ablauf zur Behandlung einer Event-Komponente dar. Die grau hinterlegten Kästen symbolisieren alternative Einheiten. Die drei „CED“-Einheiten könnten zum Beispiel verschiedene Event-Algebren unterstützen.

Links oben in der Grafik beginnend, wird eine ECA-Regel bei der ECA-Engine registriert. Diese zerlegt die Regel in ihre drei Bestandteile: Event, Condition und Action. Zur weiteren Verarbeitung wählt die ECA-Engine für jeden der Teile eine passende Engine aus. Der Eventausdruck wird bei einer Composite-Event-Detection-Unit (CED) registriert. Diese wiederum entnimmt dem Ausdruck die atomaren Events und registriert sie bei einem Atomic-Event-Matcher (AEM). Dort wird für diese Muster analysiert, aus welcher Anwendungsdomäne die gesuchten Events sind. Der AEM registriert sich dann bei den Event-Brokern (EB) der jeweiligen Domänen. Nach diesem Schritt sind alle Vorbereitungen zur Detektion von Events getroffen.

Die Detektion beginnt bei einem Domain-Knoten. Dieser erzeugt ein atomares Event und schickt es dem EB seiner Anwendungsdomäne. Dort wird es an alle registrierten AEM's weitergeleitet. Diese versuchen eine Übereinstimmung mit den registrierten Events zu finden. Wenn dies gelingt, wird eine entsprechende Nachricht an die CED gesandt, die das atomare Event registriert hat. Darin sind außer dem Event selbst auch die sich daraus ergebenden Variablenbindungen enthalten. Wenn eine solche Nachricht bei einer CED ankommt, setzt diese die Detektion von zusammengesetzten Events fort. Wird ein solches Event detektiert, wird es an die ECA-Engine gesandt, die daraufhin die Condition-Komponente der entsprechenden Regel aufruft. Wenn deren Aufruf ein positives Ergebnis liefert, wird die Aktion ausgeführt.

Abweichend von dieser Struktur wird in dieser Arbeit der AEM als integrierter Bestandteil der

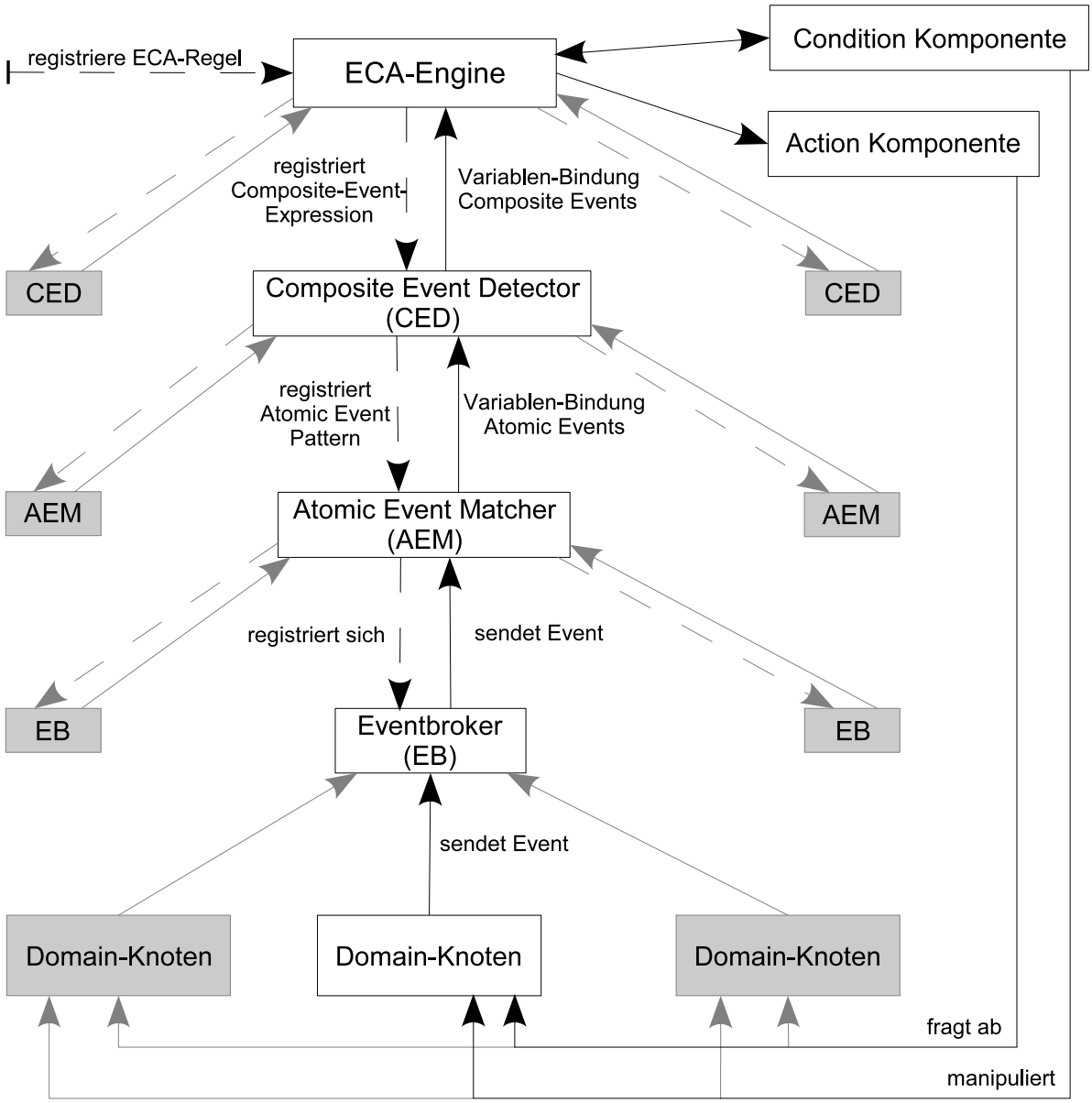


Abbildung 2.1: Aufbau des ECA-Frameworks

CED's betrachtet. Das führt dazu, dass die Snoop-CED atomare Events direkt von einem Event-Broker entgegennimmt und diese durch eigene Routinen auf Übereinstimmungen überprüft. Die dafür vorgesehenen Komponenten der Implementierung werden in Abschnitt 6.5.2 vorgestellt. Eine Trennung der Implementierung von CED und AEM ist nicht so einfach möglich, da in der vorliegenden Implementierung die bestehenden Variablenbindungen vom CED an den AEM übergeben werden, der sie dann in die Überprüfung mit einfließen lässt. Im oben beschriebenen Vorgehen wird hingegen das Pattern-Matching ohne Einfluss durch Variablen bestimmt und erst im CED ein Abgleich der bereits bestehenden mit den durch das atomare Event gebundenen Variablenwerten

durchgeführt.

Für die einzelnen Bestandteile von ECA-Regeln gibt es jeweils verschiedene Arten der Realisierung. Zum Beispiel könnten die ausgelösten Aktionen in verschiedenen Programmiersprachen formuliert werden. Durch die gewählte Struktur ist es möglich alle Arten gleichzeitig zu unterstützen, indem für jede eine Implementierung erzeugt wird, die sich an die festgelegten Schnittstellen hält. Beim Anlegen der ECA-Regeln muss dann nur noch jeweils angegeben werden, welche Event-, Bedingungs- und Aktionssprache verwendet wird. Die ECA-Engine sorgt dann für die Auswahl der richtigen Verarbeitungseinheit.

Um die Snoop-Einheit als CED in das ECA-Framework zu integrieren, müssen verschiedene Schnittstellen implementiert werden. Zur Detektion von Events benötigt man einerseits Eventausdrücke und andererseits atomare Events. Beide Schnittstellen sind im Framework zur Zeit noch nicht näher spezifiziert. Die hier verwendeten Realisierungen sind daher als Vorschläge zu verstehen und könnten nach Abschluss dieser Arbeit noch geändert werden.

Um atomare Events entgegenzunehmen wird eine URL bereitgestellt, auf der SOAP-Nachrichten mit einem beliebigen XML-Element als Inhalt erwartet werden. Dieses Element wird als atomares Event interpretiert.

Für die Übermittlung von Snoop-Event-Ausdrücken wird in Abschnitt 4.1 ein eigenes Format vorgestellt. Es wird davon ausgegangen, dass die Ausdrücke von der ECA-Engine in diesem Format an eine SOAP-Schnittstelle gesandt werden, wo sie die Snoop-Einheit entgegennimmt. Zusätzlich zu dem Ausdruck muss jede SOAP-Nachricht das Ziel für die ermittelten Events und die ID der ECA-Regel übergeben werden. Diese Daten werden im Header der Nachricht übermittelt. Mit <answer−url> wird das Ziel für die detektierten Events übergeben und mit <rule−id> die ID der Regel, für die das Event ist. Zusätzlich kann eine beliebige Anzahl von Variablensätzen angegeben werden. Jede dieser Sätze enthält beliebige Variablen und ihre Werte. Diese Angabe erfolgt wieder im Kopf der Nachricht und zwar mit dem Element <eca:variable−bindings>. Die Struktur dieses Elements wird weiter unten in diesem Abschnitt vorgestellt. Als Antwort auf einen eingegangenen Eventausdruck wird eine SOAP-Nachricht mit der ID des Automaten zurückgesandt, um den Empfang zu bestätigen. Die Realisierung der beiden Schnittstellen wird in Kapitel 7 vorgestellt.

Das Format zur Übermittlung von erkannten Events an die ECA-Engine hat schon etwas konkretere Formen angenommen. In [BFMS05] wird ein Prototyp für eine ECA-Engine vorgestellt. Dieser erweitert das ECA-Markup um Elemente für Antworten an sich selbst. Dieses Format soll sinnvollerweise auch für die Übermittlung der erkannten Composite Events verwendet werden. Jede Nachricht an die ECA-Engine beginnt dabei mit einem <answer>-Element. Diese wiederum enthält die Folge der atomaren Events, die zu dem zusammengesetzten Event beigetragen haben und die daraus resultierenden Variablenbindungen. Darüber hinaus gibt es zwei Attribute. Das eine enthält die ID der Regel, auf die sich die Antwort bezieht (ref), und das Andere,

von welcher der drei ECA-Komponenten die Antwort stammt (component). In dieser Arbeit ist der Wert immer „event“, da eine Event-Engine erstellt wird. <result> enthält als erstes Element der Antwort die primitiven Events in ihrer XML-Darstellung. Die Variablenbindungen hingegen sind in <variable−bindings> abgelegt. Für jeden Satz von Variablenbindungen wird ein <tupel>-Element angelegt. Ein Event kann höchstens einen solchen Satz haben. In einem Tupel wird jede Variable durch ein gleichnamiges Element repräsentiert. Ein Attribut „name“ enthält den Namen der Variablen. Der Wert wird als Inhalt des Elements angegeben. Alle hier vorgestellten Elemente sind im Namespace „http://www.eca.org/eca-ml“ enthalten.

Beispiel 2.6 (Meldung an die ECA-Engine)

Dieses Beispiel enthält die Übermittlung eines zusammengesetzten Events an eine ECA-Engine. Das Event zu der ECA-Regel ‚ECARule1‘ wird durch die atomaren Events ‚booking‘ und ‚cancel-flight‘ bestimmt. Darüber hinaus wurden durch diese Events die Variablen ‚customer‘ und ‚flight‘ gebunden.

```
<eca:answer component="event" ref="ECARule1" xmlns:eca="http://www.eca.org/eca−ml">
    <eca:result>
        <travel:booking customernr="187589" flightcode="935639" />
        <travel:cancel−flight code="935639" />
    </eca:result>
    <eca:variable−bindings>
        <eca:tuple>
            <eca:variable name="customer">187589</eca:variable>
            <eca:variable name="flight">935639</eca:variable>
        </eca:tuple>
    </eca:variable−bindings>
</eca:answer>
```

Weitere Informationen zum ECA-Framework des Projekts REWERSE sind unter anderem in [AAM05], [AAB+05], [MAA05] und der Webseite des Projekts[1] enthalten.

[1] URL: http://rewerse.net/

Kapitel 3

Snoop

Dieses Kapitel beschäftigt sich mit der Basis für die Detektion von kombinierten Events: der Eventalgebra. Als solche wird, bedingt durch die Aufgabenstellung, Snoop verwendet.

Snoop wird in der Masterthesis von Deepak Mishra [Mis91] als „Event Specification Language“ vorgestellt. Seine Arbeit bewegt sich im Rahmen von aktiven Datenbank Management Systemen (DBMS). Solche Datenbanksysteme sollen nicht nur auf eine Anfrage hin tätig werden, sondern auch selbstständig auf Änderungen der aktuellen Situationen reagieren und neue Aktionen auslösen können. Mögliche Auslöser für Aktivitäten des DBMS werden als Events bezeichnet.

D. Mishra unterscheidet dabei zwischen „Primitive Events“ und „Composite Events“. Die primitiven Events werden in [Mis91] weiter klassifiziert. Dabei entstehen entsprechend dem datenbankorientierten Ansatz Datenbankevents, transaktionsbasierte Events, temporale Events und explizite Events. Damit lassen sich Events zu Datenbankoperationen wie dem Einfügen oder Auslesen von Daten, dem Beginn und dem Ende einer Transaktion, dem Eintreten beliebiger Zeitpunkte und dem Ende eines Intervalls erzeugen. Die expliziten Events sind anwendungsspezifisch vom Software-Entwickler zu definieren und auszulösen. Diese expliziten Events entsprechen den atomaren Events in der Terminologie von REWERSE (siehe Kapitel 2).

Die expliziten Events sind in dieser Arbeit die einzigen von Interesse. Da hier mit Netzwerken gearbeitet wird und nicht mit Datenbanken, fallen die Datenbank- und Transaktionsevents bereits weg. Eigentlich fallen sie nicht weg, sondern werden von den einzelnen DBMS intern ausgewertet und eventuell in Form von expliziten Events ins Netz publiziert. Temporale Events könnten zwar von jedem Knoten im Netz für seine eigenen Belange erzeugt werden, aber dazu wäre eine Synchronisation der Zeiten im Netz nötig. Davon kann man aber im Allgemeinen nicht ausgehen. Daher sind einzelne Knoten im Netz für das Erzeugen von temporalen Events zuständig. Diese Events werden dann wiederum in expliziter Form im Netz zur Verfügung gestellt.

Bisher kann auf einzelne atomare Events reagiert werden. Das ist aber nicht immer ausrei-

chend. Zum Beispiel soll, um im Datenbankkontext zu bleiben, das Ändern von Daten meist nur berücksichtigt werden, wenn es nicht mehr rückgängig gemacht werden kann. Dazu ist es notwendig drei Ereignisse zu beobachten: den Start einer Transaktion, eine Änderung der Daten innerhalb dieser Transaktion und das Ende dieser Transaktion. An dieser Stelle setzt D. Mishra mit seiner Arbeit an. Er spezifiziert einige Operatoren, so dass sich die atomaren Events untereinander zu sogenannten „Composite Events“ verknüpfen lassen. Damit ist es dann z.B. möglich das Transaktionsbeispiel zu formulieren. Der Abschnitt 3.1 in diesem Kapitel geht näher auf diese Operatoren ein.

Aufbauend auf der Arbeit von Mishra wurde in [CKAK94] Snoop nochmals aufgegriffen und im Hinblick auf das Erkennen der kombinierten Events näher spezifiziert. Damit erhält Snoop eine sehr viel detailliertere Beschreibung. Einige Punkte, die bei der Entwicklung dieser Arbeit zu beachten sind, werden in Abschnitt 3.2 erläutert.

3.1 Eventausdrücke und -Operatoren

Einige Operatoren mit denen Events in Snoop verknüpft werden sind in [Mis91] enthalten. Allerdings geht [CKAK94] darüber hinaus. Dort werden weitere Operatoren eingeführt und die bestehenden formeller spezifiziert als das bei D. Mishra der Fall ist.

Mittels der Operatoren und sogenannten Eventausdrücken lassen sich weitere solcher Ausdrücke spezifizieren. Die Algebra, die damit gebildet wird, ermöglicht es auch auf kompliziertesteSituationen zu reagieren, denn mit jedem Ausdruck können Events erkannt werden, die zu ihm passen. Die einfachsten Eventausdrücke sind dabei die atomaren Events. Eigentlich muss man auch bei diesen Events zwischen dem Ausdruck und dem eigentlichen Event unterscheiden. Diese Unterscheidung wird vor allem dann ersichtlich, wenn man die atomaren Eventausdrücke parametrisiert. Dann enthalten die Ausdrücke nämlich variable Bestandteile, während die Events anstelle dieser, konkrete Werte enthalten. Man kann sagen, jedes atomare Event ist die Instanz eines Ausdrucks.

Beispiel 3.1 (Unterschied zwischen Event und Eventausdruck)
Anhand der Repräsentation der Events als XML-Markup kann man diesen Unterschied verdeutlichen. <cancel−flight code="$flight"/> ist das Eventmuster für alle Events, die besagen, dass ein Flug gestrichen wurde. Zum Beispiel ist <cancel−flight code="A−76543"/> ein Event, das auf das Muster zutrifft, aber auch <cancel−flight code="G−8967"/>.

Man kann die atomaren Eventausdrücke quasi als Muster für ein Event betrachten. Daher werden sie in dieser Arbeit als Eventmuster bezeichnet. Aus diesen Mustern und den Operatoren können nun alle weiteren Ausdrücke zusammengesetzt werden. Damit zerfallen die Events in zwei Gruppen: die atomaren und die zusammengesetzten Events, in Snoop „Composite Events“ genannt. Erstere

erfüllen einen atomaren Eventausdruck, also ein Eventmuster. Letztere hingegen passen zu einem Eventausdruck, der nicht atomar ist.

Da Snoop eine Algebra spezifiziert, deren Elemente die Eventausdrücke sind, fehlen hier noch die Operatoren. Um die Semantik der einzelnen Operatoren beschreiben zu können, wird jeder Eventausdruck E als gleichnamige Funktion aufgefasst, die jedem Zeitpunkt einem booleschen Wert zuweist.

$$E : T \longrightarrow \{True, False\} \tag{3.1}$$

Diese Funktion ist wie folgt definiert:

$$E(t) = \begin{cases} True & \text{falls ein Event gemäß } E \text{ zum Zeitpunkt } t \\ & \text{aufgetreten ist} \\ False & \text{anderenfalls} \end{cases} \tag{3.2}$$

$E(t) = True$ ist gleichbedeutend dazu, dass ein Event zum Ausdruck E zum Zeitpunkt t aufgetreten ist. $\sim E$ ist die Negation der Funktion E.

In [CKAK94] werden die folgenden Eventoperatoren definiert: OR, AND, ANY, ANY*, SEQ, A, A*, P, P* und NOT.
Was sich hinter diesen Kürzeln verbirgt, wird im Folgenden dargelegt. Dabei sind die Formeln [CKAK94] entnommen. Mit E_n werden die Eventausdrücke bezeichnet, die miteinander verknüpft werden sollen. Diese E's sind also die Operanden des jeweiligen Operators.

OR – Die Disjunktion

Mit diesem zweistelligen Operator wird ein Eventausdruck erzeugt, gemäß dem das zusammengesetzte Event auftritt, wann immer ein Event zu einem der beiden Operanden auftritt. Die Schreibweise dieses Operators ist $E_1 \nabla E_2$

$$(E_1 \nabla E_2)(t) = E_1(t) \vee E_2(t) \tag{3.3}$$

AND – Die Konjunktion

Auch dieser Operator, geschrieben als $E_1 \Delta E_2$ ist zweistellig. Dabei müssen Events zu beiden Operanden, ungeachtet ihrer Reihenfolge, auftreten.

$$(E_1 \Delta E_2)(t) = (\exists t_1)(((E_1(t_1) \wedge E_2(t)) \vee (E_2(t_1) \wedge E_1(t))) \wedge t_1 \leq t) \tag{3.4}$$

ANY – Die n-fache Konjunktion

Wenn man eine beliebige Kombination von n verschiedenen Events einer Menge detektieren will, so kann man dieses mit dem Operator ANY tun. Er besitzt einen Parameter, mit dem man die

Anzahl der nötigen Events angibt, und eine Liste von Operanden. Diese Liste enthält diejenigen Ausdrücke, die vom Operator berücksichtigt werden müssen. Es müssen also n verschiedene Events zu m verschiedenen Ausdrücken auftreten, um das kombinierte Event eintreten zu lassen. Natürlich muss n mindestens 1 und kleiner oder gleich der Anzahl an Operanden sein.

$$\begin{aligned} ANY(n, E_1, E_2, \ldots, E_{m-1}, E_m)(t) \quad &= \\ (\exists t_1)(\exists t_2)\cdots(\exists t_{n-1}) & \\ (E_i(t_1) \wedge E_j(t_2) \wedge \cdots \wedge E_k(t_{n-1}) \wedge E_l(t) \quad &\wedge \\ (t_1 \leq t_2 \leq \cdots \leq t_{n-1} \leq t) \quad &\wedge \\ (1 \leq i, j, \ldots, k, l \leq n) \quad &\wedge \\ (i \neq j \neq \cdots \neq k \neq l)) & \end{aligned} \tag{3.5}$$

ANY* – Die Konjunktion

Der Operator ANY(n,E*) oder, ohne die Parameter und Operanden, ANY* ist eine Variante von ANY. Wie man sieht, hat er nur einen Operanden, der allerdings mit einem Stern gegenzeichnet ist. Ein Event, das diesem Ausdruck entspricht, tritt auf wenn, n verschiedene Events zu dem Ausdruck E aufgetreten sind.

$$\begin{aligned} ANY(n, E^*)(t) \quad &= \\ (\exists t_1)(\exists t_1)\cdots(\exists t_{n-1}) & \\ (E(t_1) \wedge E(t_2) \wedge \cdots \wedge E(t_{n-1}) \wedge E(t)) \quad &\wedge \\ (t_1 < t_2 < \cdots < t_{n-1} < t) & \end{aligned} \tag{3.6}$$

SEQ – Die Sequenz

Ein Operator, der Ähnlichkeit mit dem AND-Operator hat, ist „SEQ“. Allerdings ist in diesem Fall die Reihenfolge, in der die Events auftreten, nicht egal. Ihre Abfolge muss genau der entsprechen, in der die Operanden angeordnet sind. Der Operator wird durch ein Semikolon dargestellt.

$$(E_1; E_2)(t) = (\exists t_1)((E_1(t_1) \wedge E_2(t)) \wedge t_1 < t) \tag{3.7}$$

A – Der Aperiodic-Event-Operator

Dieser Operator ermöglicht es ein Intervall anzugeben. In diesem Intervall führt jedes Auftreten eines bestimmten Events zum Auftreten des zusammengesetzten Events. Die Schreibweise $A(E_2)[E_1, E_3]$ symbolisiert, dass E_2 nach E_1 und vor dem ersten folgenden E_3 eintreten soll, um das Aperiodic-Event zu detektieren. Die Detektion dieser Events wird fortgesetzt bis ein E_3 das

Intervall beendet.

$$\begin{aligned}(A(E_2)[E_1, E_3])(t) &= (\exists t_1)(\forall t_2)(E_1(t_1) \wedge E_2(t) \\ &\wedge \quad (t_1 \leq t) \\ &\wedge \quad ((t_1 \leq t_2 < t) \longrightarrow\sim E_3(t_2)))\end{aligned} \tag{3.8}$$

A* – Der Kumulative Aperiodic-Event-Operator

Diese Variation von A gibt nicht jedesmal wenn E_2 auftritt ein Event aus, sondern sammelt die aufgetretenen E_2 und gibt sie dem Event mit, das am Ende des Intervalls durch E_3 ausgelöst wird.

$$(A^*(E_2)[E_1, E_3])(t) = (\exists t_1)(E_1(t_1) \wedge E_3(t) \wedge (t_1 < t)) \tag{3.9}$$

P – Der Periodic-Event-Operator

Dieser Operator weist fast die gleiche Semantik wie der aperiodische Operator auf. Der Unterschied liegt in der Art des Eventausdrucks, der das Composite Event auslöst. Hier ist nämlich zwingend ein temporales Event anzugeben, das periodisch auftritt, etwa „alle 20 Minuten“. Diese Form von Event wird in der Formel mit TI bezeichnet.

$$\begin{aligned}(P(TI)[E_1, E_3])(t) &= (\exists t_1)(\forall t_2)(E_1(t_1) \\ &\wedge \quad ((t_1 \leq t_2 < t) \longrightarrow\sim E_3(t_2)) \\ &\wedge \quad (t = t_1 + i * TI : i \geq 1))\end{aligned} \tag{3.10}$$

P* – Der Kumulative Periodic-Event-Operator

Für diesen Parameter gilt in Bezug auf A* das selbe wie für P in Bezug auf A. Die aufzusammelnden Events treten periodisch auf.

$$(P^*(TI)[E_1, E_3])(t) = (\exists t_1)(E_1(t_1) \wedge E_3(t) \wedge (t \geq t_1 + TI)) \tag{3.11}$$

NOT – Die Negation

Einer der Operatoren, die nicht in [Mis91] enthalten sind, ist der Negationsoperator. Mittels diesem Operator lässt sich ausdrücken, dass ein Event nicht auftreten soll. Da man die Detektion in einem unendlichen Zeitraum betreibt, könnte dieses Event nie detektiert werden. Darum wird zu jedem NOT-Operator ein Intervall angegeben, in dessen Inneren das Event nicht auftreten soll. $\neg(E_2)[E_1, E_3]$ bedeutet also, dass das kombinierte Event auftritt, wenn E_3 nach E_1 auftritt und dazwischen kein E_2 vorkommt.

$$\begin{aligned}(\neg(E_2)[E_1, E_3])(t) &= (\exists t_1)(\forall t_2)(E_1(t_1) \wedge \sim E_2(t) \wedge E_3(t) \\ &\wedge \quad ((t_1 \leq t_2 < t) \longrightarrow\sim (E_2(t_2) \vee E_3(t_2))))\end{aligned} \tag{3.12}$$

Die in Großbuchstaben geschriebenen Bezeichner werden in diesem Buch die Operatoren identifizieren.

Bei den einzelnen Operatoren ist besonders darauf zu achten wie das zeitliche Verhalten angegeben ist. Das heißt, in welcher Relation die einzelnen Zeitpunkte in den Formeln zueinander stehen. Beim AND-Operator zum Beispiel ist die Relation mit $t_1 \leq t$ angegeben. Solche Kleiner-Gleich-Beziehungen bedeuten, dass die Events, die zu den Operanden passen, hintereinander oder gleichzeitig eintreten können. Ein Event zu $(A \Delta A)$ beispielsweise würde immer ausgegeben, wenn *ein* A auftritt, hingegen bei (A;A) müssten *zwei verschiedene* Events zum Ausdruck A auftreten, um das kombinierte Event auszulösen. Das liegt daran, dass in der Sequenz t_1 kleiner als t sein muss ($t_1 < t$). Operatoren mit einer Kleiner-Beziehung sind SEQ und ANY*, die bedingt durch ihre Semantik die einzelnen Events hintereinander erwarten. Dieser Unterschied ist später bei der Entwicklung des Detektionsalgorithmus wichtig.

3.2 Detektionssemantik

Die Semantik der Operatoren gibt bereits Aufschluss über die Art, in der zusammengesetzte Events detektiert werden. Allerdings bedarf der Zeitpunkt t, an dem die Eventfunktion E zu „Wahr“ ausgewertet wird, noch einer Interpretation. Bei atomaren Events ist es einfach: Das Event tritt zu einem festen Zeitpunkt ein. Bei zusammengesetzten Events ist dieser Zeitpunkt schwieriger zu bestimmen. Da sich der Vorgang der Detektion über einen längeren Zeitraum erstreckt, kommen mehrere Zeitpunkte in Frage. Sowohl der Zeitpunkt, in dem das erste oder das letzte Event eintreten, können herangezogen werden, um den Zeitpunkt des Auftretens festzulegen. In Snoop tritt ein zusammengesetztes Event zum selben Zeitpunkt ein, zu dem das die Detektion abschließende Event eintritt.

Beispiel 3.2 *Um einen Kunden über die Streichung eines von ihm gebuchten Fluges zu benachrichtigen, wird folgendes Event spezifiziert:*
<booking flight="$flight">;<cancel−flight flight="$flight">
Damit wird registriert, wenn ein Kunde einen Flug bucht. Sollte dann dieser Flug ausfallen, tritt das zusammengesetzte Event auf. Der Zeitpunkt, in dem der Flug gestrichen wird, ist somit der Zeitpunkt, in dem der „Flug eines Kunden“ gecancelt wird.

Abweichend von Snoop wird in dieser Arbeit bei atomaren Events nicht der Zeitpunkt verwendet zu dem das Event aufgetreten ist, sondern der, an dem die Detektionseinheit das Event empfangen hat. Dies wird durch die Verwendung der Automaten nötig. Da diese ein Eingabeband besitzen, von dem sequentiell gelesen wird, werden die Events in der Reihenfolge bearbeitet, in der die Events durch den Automaten wahrgenommen werden. Diese Reihenfolge ist aber verschieden

vom Auftreten der Events, da diese unterschiedlich lange durchs Netzwerk unterwegs sind. Eine Annäherung an die ursprüngliche Semantik von Snoop kann nur durch eine Optimierung des Netzes erreicht werden. Der Einfachheit halber wird allerdings immer vom Zeitpunkt des Eintretens gesprochen.

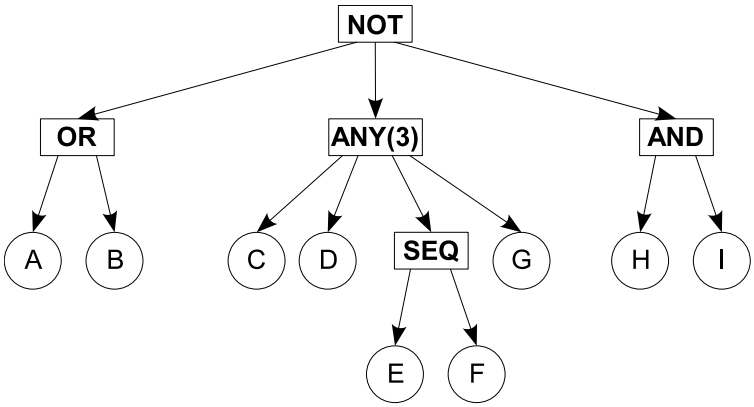
Aus den Formeln der einzelnen Eventfunktionen lässt sich ablesen, dass der Zeitpunkt, zu dem ein kombiniertes Event eintritt, dem Zeitpunkt entspricht, an dem das letzte Event eingetreten ist. Der Zeitpunkt t von der linken Seite der Gleichungen ist immer auch der Zeitpunkt, in dem einer der Operanden eintritt. Ein Event zu diesem Operand wird als auslösendes Event bezeichnet.

In [CKAK94] wird eine mengenbasierte Form der Detektion verwendet. Dort werden aus einer Menge mit allen bisher eingetretenen atomaren Events die Vorkommnisse der kombinierten Events ermittelt. Entscheidend für die Detektion ist, dass die einzelnen Teilausdrücke auf gleicher Ebene jeweils unabhängig voneinander bearbeitet werden können. Anhand eines Operatorbaums lässt sich das Vorgehen gut darstellen. Dabei stellt jeder Knoten des Baums einen Teilausdruck dar. Ausgehend von der Wurzel werden die einzelnen Teilausdrücke unabhängig voneinander ausgewertet. Auch die Teilausdrücke werden wieder so behandelt. Dadurch wird jeder (Teil-)Ausdruck genau dann detektiert, wenn alle seine Teilbäume positiv ausgewertet wurden. Dabei können die einzelnen Teilbäume getrennt behandelt werden.

Beispiel 3.3 (Operatorbaum) *Zum Eventausdruck*

$$\neg((A \nabla B), (ANY(3, C, D, (E; F), G)), (H \Delta I))$$

würde ein Operatorbaum wie in untenstehender Abbildung aussehen. Die Buchstaben symbolisieren die atomaren Events.



Um den NOT-Operator zu detektieren, werden drei Prozesse gestartet, die den OR-, ANY- und AND-Operator detektieren. Jeder dieser Prozesse erzeugt wiederum Kindprozesse, um die Events in den Blättern zu erkennen. Im Fall des ANY-Operators wird außerdem ein Prozess erzeugt, der den Sequenz-Operator behandelt.

Diese Art der Detektion unterscheidet sich allerdings derart von der automatenbasierten, dass nicht weiter darauf eingegangen werden soll. Wichtig ist nur, dass die Ergebnisse in beiden Arten des Detektierens identisch sind.

Kapitel 4

XML-Markup's

Bevor die Verarbeitung der Daten erfolgen kann, muss deren Struktur erst einmal festgelegt werden. Dieses Kapitel beschäftigt sich mit den Markup-Sprachen, in denen die benötigten Daten, die Eventspezifikationen, formuliert sind. Im Rahmen dieser Arbeit wurden zwei solcher Sprachen entwickelt. Eine, um die Eventausdrücke in deklarativer Form abzufassen, und eine andere für eine Darstellung auf Basis von Automaten.

4.1 Eventausdrücke

Dieser Abschnitt ist dem XML-Markup gewidmet mit dem man Snoop-Events ausdrücken kann. Das Ziel war es, eine Sprache zu formulieren, die es dem Benutzer möglich macht, schnell und einfach Eventausdrücke gemäß Snoop zu beschreiben. Diese können dann automatisch in das komplexere Automatenformat umgewandelt werden, das für die Verarbeitung durch den Algorithmus verwendet wird (siehe Abschnitt 4.2). Als Ergebnis ist ein XML-Format entstanden, das sich durch eine kleine Menge von zulässigen Elementen und eine einfache Anordnung dieser Elemente auszeichnet.

Alle im Folgenden beschriebenen Elemente sind im Namespace „http://rewerse.net/snoop-expression-ml“ enthalten. Für jeden Operator, der in Snoop spezifiziert wird, gibt es ein Element. Im einzelnen sind dies:

- <And> für den Operator AND
- <Or> für den Operator OR
- <Sequence> für den Operator SEQUENZ
- <Any> für den Operator ANY
- <Multi_Occurrences> für den Operator ANY*

- <Aperiodic> für den Aperiodic-Event Operator
- <Cumulative_Aperiodic> für den kumulativen Aperiodic Operator
- <Periodic> für den Periodic-Event Operator
- <Cumulative_Periodic> für den kumulativen Periodic Operator
- <Not> für den Operator NOT

Diese Elemente werden im Folgenden als „Operator-Tags" bezeichnet. Über diese Tags hinaus gibt es noch zwei weitere Elemente. Zum einen <Eventdeclaration> als Wurzel für eine Eventdeklaration; und zum Zweiten <Atomic−Event>. Es symbolisiert, dass sein Inhalt als Vorlage für atomare Events zu interpretieren ist. Dieser Inhalt kann aus beliebigem Markup bestehen.

Innerhalb der Operator-Tags existieren eine Reihe von Kindern. Deren Anzahl ist abhängig von der Zahl der Operanden des repräsentierten Operators. Deren Spezifikationen wurden im Kapitel über SNOOP (Kap. 3) vorgestellt. Darüber hinaus benötigen zwei der Operatoren eine zusätzliche Angabe. Dies sind die Operatoren ANY und ANY*. Die benötigte Nummer ist mittels eines Attributs namens „number-of-occurrences" anzugeben.

Die Struktur, in der die Elemente angeordnet werden, ist die eines Operatorbaumes. Dieser steht innerhalb des Tags „Eventdeclaration". Als Wurzel des Operatorbaumes dient ein beliebiger Operator. Dessen Inhalt sind seine Operanden. Diese wiederum können atomare Events oder weitere Operatoren sein. Ein Beispiel für einen Operatorbaum findet man auf Seite 21. Die Darstellung dieses Baums im XML-Format ist in Beispiel 4.2 auf Seite 26 abgedruckt.

Die Variablen eines Ausdrucks können vorbelegt sein. Hierzu muss im <Eventdeclaration>-Element ein XML-Fragment mit den Variablenbindungen angelegt werden. Dieses Fragment muss im ECA-Markup verfasst werden, welches bereits in Abschnitt 2.3 vorgestellt wurde. Von Interesse aus diesem Markup ist hier <eca:variable−bindings>. Damit können mehrere Sätze an Variablen angegeben werden. Für jeden dieser Sätze muss dann eine eigene Detektion vorgenommen werden, in der die jeweiligen Variablen Gültigkeit haben.

Beispiel 4.1 (Variablenbindungen)
Die Bindung zweier Variablen mit jeweils zwei verschiedenen Werten zeigt das folgende XML-Fragment:

```
<eca:variable−bindings>
    <eca:tuple>
        <eca:variable name="Variable1">Erster Wert der ersten Variable</eca:variable>
        <eca:variable name="Variable2">Erster Wert der zweiten Variable</eca:variable>
    </eca:tuple>
    <eca:tuple>
        <eca:variable name="Variable1">Zweiter Wert der ersten Variable</eca:variable>
```

```
        <eca:variable name="Variable2">Zweiter Wert der zweiten Variable</eca:variable>
    </eca:tuple>
</eca:variable-bindings>
```

Das folgende Listing beinhaltet eine „Document Type Definition“ (DTD). Diese modelliert die Eventausdrücke nicht vollständig. Sie enthält keine Einschränkungen für das Attribut des ANY- bzw. ANY*-Operators. Weder ist eine positive Zahl vorgeschrieben noch eine obere Grenze durch die Anzahl der Operanden realisiert. Die Beschränkungen der Attribute könnten zwar mittels XML-Schema dargestellt werden, allerdings würde das immer noch nicht für eine Validierung ausreichen. Wegen des beliebigen Inhalts der <Atomic−Event>-Elemente wird es nie gelingen eine vollständige Definition zu schreiben. Aus diesem Grund dient diese DTD nur der Veranschaulichung des Markups.

```
<!ENTITY % operand "(And|Or|Sequence|Any|Multi_Occurrences|Aperiodic|Cumulative_Aperiodic|
    Periodic|Cumulative_Periodic|Not|Atomic-Event)" >
<!ELEMENT Eventdeclaration (%operand; | eca:variable-bindings )>
<!ATTLIST Eventdeclaration
    version CDATA #FIXED "0.4"
    xmlns:snoop CDATA #REQUIRED >
<!ELEMENT And (%operand; , %operand;) >
<!ELEMENT Or (%operand; , %operand;) >
<!ELEMENT Sequence (%operand;, %operand;) >
<!ELEMENT Any (%operand;)* >
<!ATTLIST Any
    number-of-occurrences CDATA #REQUIRED >
<!ELEMENT Multi_Occurrences %operand; >
<!ATTLIST Multi_Occurrences
    number-of-occurrences CDATA #REQUIRED >
<!ELEMENT Aperiodic (%operand;, %operand;, %operand;) >
<!ELEMENT Cumulative_Aperiodic (%operand;, %operand;, %operand;) >
<!ELEMENT Periodic (%operand;, %operand;, %operand;) >
<!ELEMENT Cumulative_Periodic (%operand;, %operand;, %operand;) >
<!ELEMENT Not (%operand;, %operand;, %operand;) >
<!ELEMENT Atomic-Event ANY >
<!ELEMENT eca:variable-bindings (eca:tuple)* >
<!ELEMENT eca:tuple (eca:variable)* >
<!ELEMENT eca:variable (PCDATA) >
<!ATTLIST eca:variable
    name CDATA #REQUIRED >
```

Listing 4.1: DTD für Snoop Ausdrücke

Um entgegen dem oben gesagten eine validierbare Version der DTD zu erhalten, gibt es zwei Möglichkeiten. Einerseits könnte man aus dem Eventausdruck alle atomaren Events entfernen,

bevor man die Validierung vornimmt. Die Events können beliebiges Markup sein, so dass bei ihnen kein Fehler auftreten kann. Somit ist ihr Entfernen ohne Verlust für die Validierung möglich. Die zweite Möglichkeit besteht darin, die DTD dynamisch an den Ausdruck anzupassen. Dazu ist es nötig, für jedes Element, das in den atomaren Events vorkommt, eine möglichst allgemeine Deklaration in die DTD aufzunehmen. Das gleiche müsste für alle Attribute geschehen, die in den atomaren Events vorkommen. Keine der beiden Möglichkeiten wird im Rahmen dieser Arbeit umgesetzt, da zeitliche Beschränkungen dieses verhindern.

Als Abschluss dieses Abschnittes soll folgendes Beispiel stehen.

Beispiel 4.2 (Snoopausdruck im XML-Format)
Dieses Beispiel enthält die XML-Darstellung von $\neg((A \nabla B), (ANY(3, C, D, (E; F), G)), (H \Delta I))$. *Der Operatorbaum zu diesem Eventausdruck wurde bereits in Beispiel 3.3 angegeben. Beim Vergleich der Strukturen zwischen dem Baum und dem XML-Format fällt eine weitgehende Übereinstimmung auf. Die Elemente des Namespace foo stehen für die Muster der atomaren Events. Diese werden hier nur durch leere Elemente angedeutet.*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Eventdeclaration SYSTEM "./snoop.dtd" >
<snoop:Eventdeclaration
   xmlns:snoop= "http://rewerse.net/snoop-expression-ml"
   xmlns:foo= "http://rewerse.net/event-ml">
<snoop:Not>
   <snoop:Or>
      <snoop:Atomic-Event >
         <foo:EventA />
      </snoop:Atomic-Event>
      <snoop:Atomic-Event>
         <foo:EventB />
      </snoop:Atomic-Event>
   </snoop:Or>
   <snoop:Any number-of-occurrences= "3">
      <snoop:Atomic-Event>
         <foo:EventC />
      </snoop:Atomic-Event>
      <snoop:Atomic-Event>
         <foo:EventD />
      </snoop:Atomic-Event>
      <snoop:Sequence>
         <snoop:Atomic-Event>
            <foo:EventE />
         </snoop:Atomic-Event>
         <snoop:Atomic-Event>
            <foo:EventF />
         </snoop:Atomic-Event>
      </snoop:Sequence>
```

```
        <snoop:Atomic-Event>
            <foo:EventG />
        </snoop:Atomic-Event>
    </snoop:Any>
    <snoop:And>
        <snoop:Atomic-Event>
            <foo:EventH />
        </snoop:Atomic-Event>
        <snoop:Atomic-Event>
            <foo:EventI />
        </snoop:Atomic-Event>
    </snoop:And>
</snoop:Not>
</snoop:Eventdeclaration>
```

An diesem Beispiel ist gut zu sehen, wie die Operatoren ineinander geschachtelt werden. Der äußere Not-Operator enthält als Operanden den Or-, Any- und And-Operator. Diese enthalten wiederum weitere Operatoren und atomare Events.

4.2 Eventautomaten

Im Gegensatz zum ersten Markup in diesem Kapitel, ist das aus diesem Abschnitt nicht darauf ausgelegt einfach zu erstellen und zu lesen zu sein. Vielmehr ist es auf die algorithmische Bearbeitung ausgerichtet. Dazu wird eine automatenbasierte Darstellung gewählt. Als Ausgangsbasis werden endliche Automaten verwendet. Diese werden in einigen Punkten erweitert, um den Anforderungen genüge zu tun.

4.2.1 Automatendarstellung

Bevor die XML-Syntax vorgestellt wird, soll erst einmal das Konstrukt erläutert werden, das dadurch abgebildet wird: die Automaten.

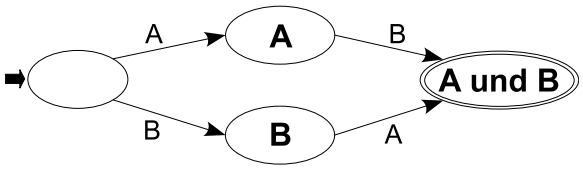
Ein jeder Automat stellt dabei einen SNOOP-Ausdruck dar. In einer Variante, die später besprochen wird, entspricht er dagegen einem Operator innerhalb eines Ausdrucks. Die Idee ist, jeden Zustand eines Automaten als eine Menge von bereits eingetretenen atomaren Events zu betrachten. Die Zustandsübergänge des Automaten symbolisieren dann das Auftreten eines weiteren Events.

Da die Menge der möglichen Events unbeschränkt ist, kann nicht zu jedem Zustand und jedem Event eine Transition angegeben werde. Für alle Events, zu denen keine Transition existiert, wird angenommen, dass sie den Automaten nicht verändern. Somit legen die Transitionen die Menge der Events fest, die etwas zu dem zusammengesetzten Event beitragen.

In einigen Punkten sind Automaten unzureichend. Wenn man die Automaten, die aus Eventdeklarationen entstehen können, näher betrachtet, kann man zum Beispiel feststellen, dass die Endzustände des Automaten nicht immer die Zustände sind, an denen ein Event detektiert wird. Dieser Fall tritt immer dann auf, wenn ein atomares Event innerhalb eines Intervalls auftreten soll (Periodic- und Aperiodic-Operatoren). Aus diesem Grund wird eine „Detect"-Markierung in die Automaten aufgenommen, die die Zustände markiert, an denen das Event detektiert wird.

Eine weitere Markierung der Zustände ergibt sich aus der Semantik von Snoop. Wie in Kapitel 3 dargestellt wurde, können atomare Events in verschiedenen Kombinationen zu ein und demselben Eventausdruck beitragen.

Beispiel 4.3 *Geht man von dem Ausdruck „$A\Delta B$ " aus, ergibt sich folgende Darstellung:*



Das bedeutet, jede Kombination von A und B die auftritt, soll als Event detektiert werden. Nun trete A zweimal auf. Danach soll B auftreten. Daraus ergeben sich zwei neue Events die „A und B" genügen: (A_1, B_1) und (A_2, B_1). Wenn ein weiteres B auftritt, ergeben sich zwei weitere Events: (A_1, B_2) und (A_2, B_2). Wie man sieht, kann für die Detektion eines Events jedes bereits eingetretene atomare Event herangezogen werden und das unabhängig davon, ob es bereits einmal verwendet wurde.

Damit dieses Verhalten modelliert werden kann existieren zu jedem Automaten üblicherweise mehrere Instanzen. Jede dieser Instanzen hat ihren eigenen Status und damit auch einen eigenen aktiven Zustand. Eine neue Instanz entsteht immer dann, wenn nach einem Zustandsübergang der Automat zusätzlich im alten Zustand verbleiben soll.

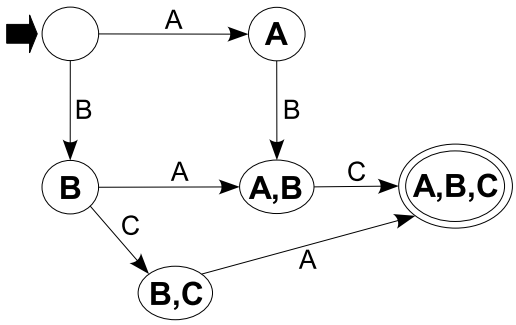
Beispiel 4.4 *Erneut wird der Ausdruck aus Beispiel 4.3 betrachtet. Wenn es zu diesem Automat immer nur eine Instanz gibt, würde dieser beim Auftreten von A_1 den Zustand wechseln. Alle weiteren A's würden nicht mehr bearbeitet, da in unserem Fall nur noch B's für den Automaten in Frage kommen. Das selbe passiert bei allen B's nach B_1. Wenn hingegen mehrere Instanzen existieren könnten, verbleibt eine vor dem ersten A. Eine weitere führt den Zustandsübergang aus. Am Ende unseres Beispiels gibt es damit sieben Instanzen von „A und B": die Ursprüngliche, jeweils eine, die bei A_1 bzw. A_2 in den nächsten Zustand überging, und vier, die aus der Kombination der A- und B-Events entstehen.*

Allerdings ist eine Trennung in mehrere Instanzen nicht immer erforderlich. In welchen Fällen die Aufspaltung nötig ist und wann nicht, wird später bei der Besprechung der Umsetzung der einzelnen Operatoren erläutert.

Eine weitere, wichtige Information leitet sich aus dem zeitlichen Verhalten von Snoop ab. Es geht darum, ob die einzelnen Operanden eines Events zum gleichen Zeitpunkt auftreten dürfen, oder ob diese Zeitpunkte unterschiedlich sein müssen. Die Information über dieses Verhalten sollte in den Automaten mit einfließen, wenn der verarbeitende Algorithmus sie nicht aufwändig aus der Struktur des Automaten rekonstruieren soll.

Wie weiter oben bereits erwähnt, können die Automaten in zwei Varianten aus den deklarativen Ausdrücken erzeugt werden. Einerseits kann der gesamte Ausdruck in einen einzigen Automaten umgesetzt werden. Anderenfalls gibt es die Möglichkeit, einen jeden Operator als eigenen Automaten umzusetzen und diese untereinander zu verknüpfen. Erstere Lösung scheint nahe liegend. Sie fasst alle nötigen Informationen zum Erkennen eines Events in einem Automat zusammen. Ein Nachteil dieser Methode ist, dass die Automaten, sehr kompliziert werden. Bedingt dadurch, dass in Snoop der Beginn der Detektion eines Teilausdrucks nicht daran gebunden ist, dass alle in der Deklaration vorangehenden Teilausdrücke bereits abgeschlossen sind (siehe Kapitel 3), kann jedes atomare Event, das nicht zum Abschluss eines Subausdrucks führt, zu jedem beliebigen Zeitpunkt auftreten. Diese Eigenschaft soll an einem kleinen Beispiel verdeutlicht werden.

Beispiel 4.5 *Der Ausdruck „A und (B seq C)“ führt zu folgendem Automaten.*



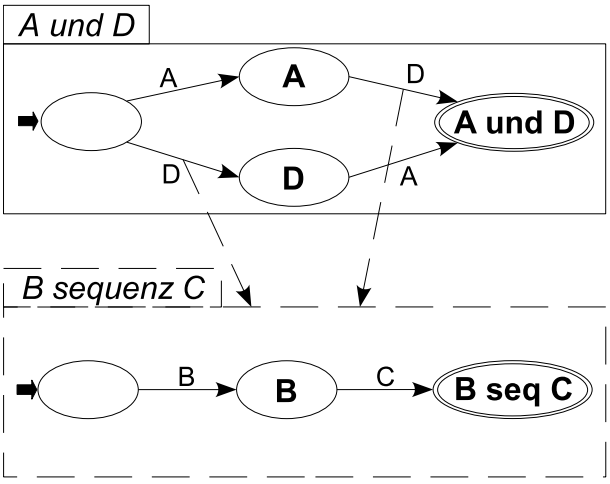
Da Snoop ein zusammengesetztes Event zu dem Zeitpunkt detektiert, zu dem das letzte atomare Event eingetreten ist und der Beginn der Detektion eines Subausdruck nicht daran gebunden ist, dass alle in der Deklaration vorangehenden Subausdrücke bereits abgeschlossen sind, kann B zu einem beliebigen Zeitpunkt vor C auftreten – also auch vor A.

Da im Allgemeinen viele atomare Events in einem Ausdruck vorhanden sind, die einen Subausdruck nicht abschließen, man denke nur an den Operator ANY mit seiner unbegrenzten Anzahl von Operanden, entstehen entsprechend viele Zustände für alle möglichen Kombinationen von solchen ‚freien‘ Events aus dem ganzen Ausdruck. Im schlimmsten Fall ist die Anzahl der Zustände exponentiell zu der der atomaren Events. Dies ist dann der Fall wenn alle 2^n Kombinationen von n unterschiedlichen Events im Automat aufgenommen werden müssen. Dazu kommt, dass diese Zustände auf vielfältige Weise miteinander verknüpft sind. Dadurch bedingt, kann der Ausdruck

bei der Umwandlung in einen Automaten nur als eine Einheit bearbeitet werden. Ein Programm, das die Umwandlung durchführt, würde entsprechend komplex.

Glücklicherweise gibt es eine Alternative. Wenn man dazu übergeht einen Automaten pro Operator statt pro Ausdruck zu erstellen, treten die Probleme mit der Umwandlung der Ausdrücke nicht mehr auf. Im Folgenden wird die Menge aller Automaten, die zusammen einen Ausdruck nachbilden, als Automaten-Gruppe bezeichnet.

Beispiel 4.6 *Der Ausdruck aus Beispiel 4.5 sieht als Automatengruppe wie folgt aus.*



Er resultiert in zwei Automaten. Ein Automat für „A und D" und ein Automat für „B seq C". Dabei ist D ein Synonym für „B seq C". Die gestrichelten Linien zwischen den mit D beschrifteten Linien und dem unteren Automaten symbolisieren, dass der Auslöser für die Transition am Anfang des Pfeils der Automat am Ende des Pfeils ist.

Damit der Zusammenhang der einzelnen Operatoren nicht verloren geht, bedarf es einer Verknüpfung der ihnen zugeordneten Automaten. Dazu ist für diese eine eindeutige Adressierung nötig. Dies wird erreicht, indem jedem Automaten ein Name zugewiesen wird. Um über diesen Namen auf einen Automaten zuzugreifen, wird eine neue Art von Transitionen eingefügt. Bisher wurde ein Zustandsübergang ausgelöst, wenn das atomare Event der Transition aufgetreten ist. Zusätzlich muss jetzt auf das Eintreten eines Composite Events reagiert werden können. Die neue Art der Transition erhält dazu, statt des atomaren Events, eine Referenz auf einen Automaten. Wenn der referenzierte Automat nun in einen Zustand übergeht, der die Detect-Markierung enthält, soll der referenzierende Automat schalten.

Der Vorteil bei diesem Vorgehen liegt darin, dass jeder Teilausdruck auch als Automat geschlossen bleibt. Das heißt, die Teilausdrücke beeinflussen sich bei der Erzeugung der Automaten nicht. Das war bei der ersten Lösung nicht der Fall. Dort ist diese gegenseitige Beeinflussung der Grund für den hohen Aufwand beim Erstellen des Automaten. Diese Unabhängigkeit erleichtert die Erzeugung der Automaten natürlich erheblich. Jeder Teilausdruck kann einzeln und unabhängig von

den anderen transformiert werden. Der Aufwand für die Erzeugung verschiedener Transitionen für atomare und kombinierte Events und das Verknüpfen der Automaten hält sich in Grenzen. Ein weiterer Vorteil der Lösung mit Automatengruppen ist, dass alle Automaten zum selben Operator eine identische Grundstruktur aufweisen. Der einzige Unterschied zwischen solchen Automaten liegt in den Events, die auftreten. Die Menge und Anordnung der Zustände sind immer gleich. Das bedeutet, dass zu jedem Operator einmalig ein sogenanntes Automaten-Muster erstellt werden kann, dass dann nur noch mit den richtigen atomaren Events bestückt werden muss, um eingesetzt werden zu können. Diese Muster werden im folgenden Abschnitt 4.2.2 näher vorgestellt.

Wenn man die beiden möglichen Modellierungen gegenüberstellt, fällt auf, dass die Lösung mit Automatengruppen eine einfache Umwandlung der Ausdrücke in Automaten ermöglicht. Demgegenüber steht ein hoher Aufwand und exponentielles Wachstum beim Modellieren der Ausdrücke mit nur einem Automat. Bei der Detektion von Events hingegen ist diese Methode im Vorteil. Der Detektionsalgorithmus für Automatengruppen benötigt voraussichtlich eine Fallunterscheidung mehr. Diese Fallunterscheidung realisiert die unterschiedliche Behandlung der beiden Typen von Transitionen. Die atomaren Events benötigen ein anderes Vorgehen als die Transitionen, die von einem anderen Automaten abhängen.

Im Rahmen dieser Arbeit war nicht die Zeit beide Möglichkeiten ausführlich zu untersuchen, deshalb wurde für das weitere Vorgehen eine ausgewählt. Die Wahl fiel auf die Automatengruppen, da sie leicht zu erstellende Automaten mit nur geringfügig höherem Aufwand bei der Implementierung eines Detektionsalgorithmus in sich vereint. Außerdem ist die Abbildung näher an der Semantik von Snoop. Dort werden die einzelnen Teilausdrücke auch unabhängig voneinander detektiert, um erst dann kombiniert zu werden. Im Folgenden ist also immer, wenn von Automaten gesprochen wird, die Interpretation als Snoop-Operatoren gemeint.

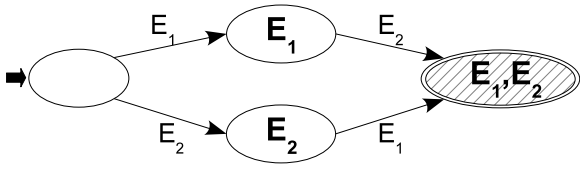
4.2.2 Automatenmuster

Um Automaten aus einem Eventausdruck zu generieren, können dessen Operatoren rekursiv verarbeitet werden. Dazu wird der Operatorbaum zu dem Ausdruck von der Wurzel an verarbeitet. Für jeden Operator entsteht dabei ein eigener Automat, der mit seinen Nachfolgern im Baum verknüpft ist.

Beispiel 4.7 *Zum Ausdruck aus Bsp. 4.2 würde jeweils ein Automat für jeden Knoten erzeugt werden, der kein atomares Event darstellt. Das heißt, es würden Automaten für $A \nabla B$, $E;F$ und $H \Delta I$ erzeugt. Dann könnten die Automaten zu den Operatoren NOT und ANY erzeugt werden, die auf die anderen Automaten verweisen. Wie diese Automaten genau aussehen wird im Beispiel 4.8 gezeigt.*

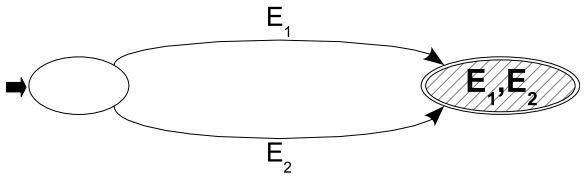
Im Folgenden werden die Operatoren, die durch Snoop definiert werden, in ihrer Automatenschreibweise vorgestellt. Dabei stellen Kreise und Ellipsen die Zustände dar. Ihre Beschriftung enthält die Events, die aufgetreten sind, wenn der Zustand erreicht wird. Dabei ist die Reihenfolge, in der die Events aufgeführt werden, nach dem Index sortiert und entspricht nicht der Reihenfolge, in der die Events aufgetreten sind. Ein breiter Pfeil markiert den Zustand, der zu Beginn der Detektion aktiv ist. Doppelte Linien kennzeichnen einen Endzustand. Zustände, in denen ein Event erkannt wird, sind schraffiert dargestellt. Die Pfeile zwischen den Zuständen symbolisieren die Transitionen. Ihre Beschriftung ist der jeweilige Auslöser. Wie bisher auch, werden keine Transitionen für Events angegeben, die in der Spezifikation nicht vorkommen, da sie den Automaten nicht beeinflussen. Die hier als Operanden verwendeten Events E_n können sowohl atomare Events als auch Composite Events sein. Diese Unterscheidung wird erst relevant, wenn die Vorlagen auf einen konkreten Ausdruck angewandt werden.

Muster für Automaten zum AND-Operator



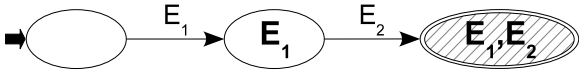
Dieser Automat findet bei (Teil-)Ausdrücken Verwendung, die aus dem AND-Operator bestehen. Im Startzustand kann entweder der erste oder zweite Operand (hier E_1 und E_2) auftreten, was zu entsprechenden Zuständen führt. In diesen Zuständen kann außerdem noch das jeweils andere Event hinzukommen. Damit wird der Endzustand erreicht. In diesem Fall führt dieser auch zur Detektion des kombinierten Events. Dabei kann die Abfolge E_1 – E_2 über den oberen Pfad des Automaten und E_2 – E_1 über den unteren das Event auslösen. Damit erfüllt der Automat die Anforderungen des Operators.

Muster für Automaten zum OR-Operator



Noch einfacher ist der Automat im Falle des OR-Operators. Er besteht nur aus zwei Zuständen und zwei Transitionen. Vom Startzustand aus führt jeder der beiden Operanden zum Endzustand. Dieser löst auch gleich das kombinierte Event aus. Wie man sieht, führt sowohl E_1 als auch E_2 zum Auslösen des Composite Events. Das entspricht einem Oder.

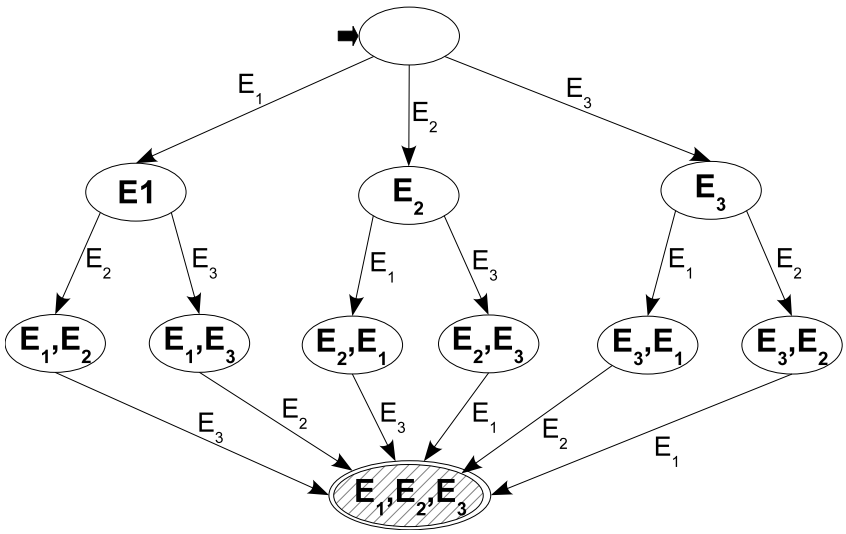
Muster für Automaten zum Sequenz-Operator



Anders als beim Oder sind in der Sequenz die Zustände nicht nebeneinander, sondern hintereinander angeordnet. Vom Startzustand aus wird nur auf den ersten Operanden reagiert. Tritt dieser ein, geht der Automat in den zweiten Zustand über. Dort ist nur noch der zweite Operant E_2 von Interesse. Wenn E_2 eintritt, wechselt der Automat in den End- und Detect-Zustand. Das heißt, nur die Abfolge E_1 – E_2 ist gültig für das Event. Die strenge Abfolge der Zustände und damit der Events ist das, was den sequentiellen Operator auszeichnet.

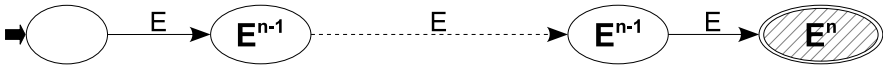
Muster für Automaten zum ANY-Operator

Der Automat zum ANY-Operator ist im Gegensatz zu den drei bisherigen Automaten nicht so statisch konstruiert. Dadurch, dass sowohl die Anzahl der Operanden beliebig ist, als auch eine variable Anzahl zum Erkennen des Events vorliegt, muss der Automat dynamisch erzeugt werden.



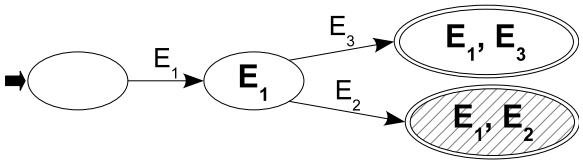
Das Bild stellt einen Automaten für drei Operanden und einer Anzahl von ebenfalls drei nötigen Events dar. Die Baumstruktur ist gut zu sehen. Die Blätter des Baums werden in einem zusätzlichen Zustand, dem Endzustand, zusammengeführt. Dieser führt auch zur Detektion des zusammengesetzten Events. Die Höhe des Baums entspricht immer der Anzahl der benötigten Events. Jeder der Zustände aus dem Baum kann auf alle Events reagieren, die auf dem Pfad, der zu diesem führt, noch nicht aufgetreten sind. Damit gibt es für jede Abfolge von Events, die das Composite Event auslösen eine Abfolge von Zuständen durch den Algorithmus.

Muster für Automaten zum ANY*-Operator



Ähnlich seinem Verwandten, dem ANY-Operator, besitzt ANY* ebenfalls einen variablen Wert. Allerdings hat dieser Operator immer nur einen Operanden. Der Automat, der aus ANY* entsteht, hat Ähnlichkeit mit dem Sequenzoperator. Der Unterschied liegt in der Anzahl der Zustände. Hier sind es immer so viele Zustände, wie der Parameter vorgibt, plus dem Initialen. Von jedem Zustand, außer dem letzten, führt genau eine Transition zu einem weiteren Zustand. Alle Transitionen haben als Auslöser das Event aus dem einzigen Operanden. Durch die sequentielle Abfolge der Zustände steht am Ende das n-malige Auftreten des gewünschten Events.

Muster für Automaten zum Aperiodic-Event-Operator

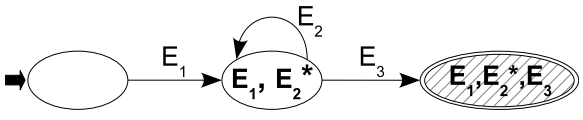


Das Aperiodic-Event zeigt in Snoop an, dass ein Event E_2 nach einem E_1 eingetreten ist. Zwischen diesen beiden Events darf kein E_3 eingetreten sein. Der oben abgebildete Automat sieht etwas merkwürdig aus, da er zwei Endzustände hat. Wenn das Detektionsintervall durch Eintreten des Events E_1 beginnt, schaltet der Automat in den Zustand E_1. Wenn im Folgenden ein E_3 auftritt, geht der Automat in den Endzustand E_1,E_3 über und erzeugt kein Event. Sollte aber E_2 anstelle von E_3 auftreten, wird der zweite Endzustand erreicht und das kombinierte Event ausgelöst.

Da innerhalb des Intervalls aus E_1 und E_3 mehrere E_2 auftreten können, die alle zu einem neuen Event führen sollen, ist es nötig den Zustand (E_1) bei einem Zustandsübergang zu (E_1,E_2) zu erhalten. Beim zweiten möglichen Übergang ist eine Aufspaltung in mehrere Instanzen nicht erlaubt. Dadurch wird erreicht, dass alle Instanzen des Automaten im Endzustand sind, nachdem E_3 aufgetreten ist. Der initiale Zustand muss auch dupliziert werden, damit neue Intervallstarts bemerkt werden können.

An diesem Automaten sieht man, auch, dass die Endzustände nicht automatisch ein Event auslösen.

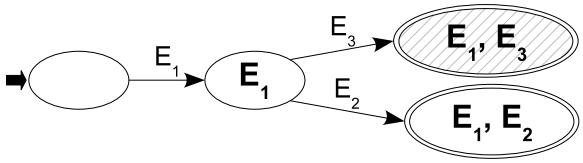
Muster für Automaten zum Aperiodic*-Event-Operator



Bei der kumulativen Variante des Aperiodic-Event-Operators ist es nicht nötig zwei Endzustände zu erzeugen. Nachdem das Intervall mit E_1 begonnen hat, muss jedes Auftreten von E_2 protokolliert werden bis ein E_3 das Composite Event abschließt. Im Automat wird dieses geregelt, indem eine Transition den (E_1,E_2*)-Zustand auf sich selbst abbildet. Dabei bedeutet der Stern, dass das entsprechende Event mehrfach auftreten kann.

Außer beim Verlassen des initialen Zustands, muss bei keinem der Zustandsübergängen eine neue Instanz des Automaten erzeugt werden. Damit wird sichergestellt, dass alle Events E_1 zu einem neuen Intervall führen und alle Automateninstanzen nach einem Event E_3 in einem Endzustand sind.

Muster für Automaten zum Negationsoperator



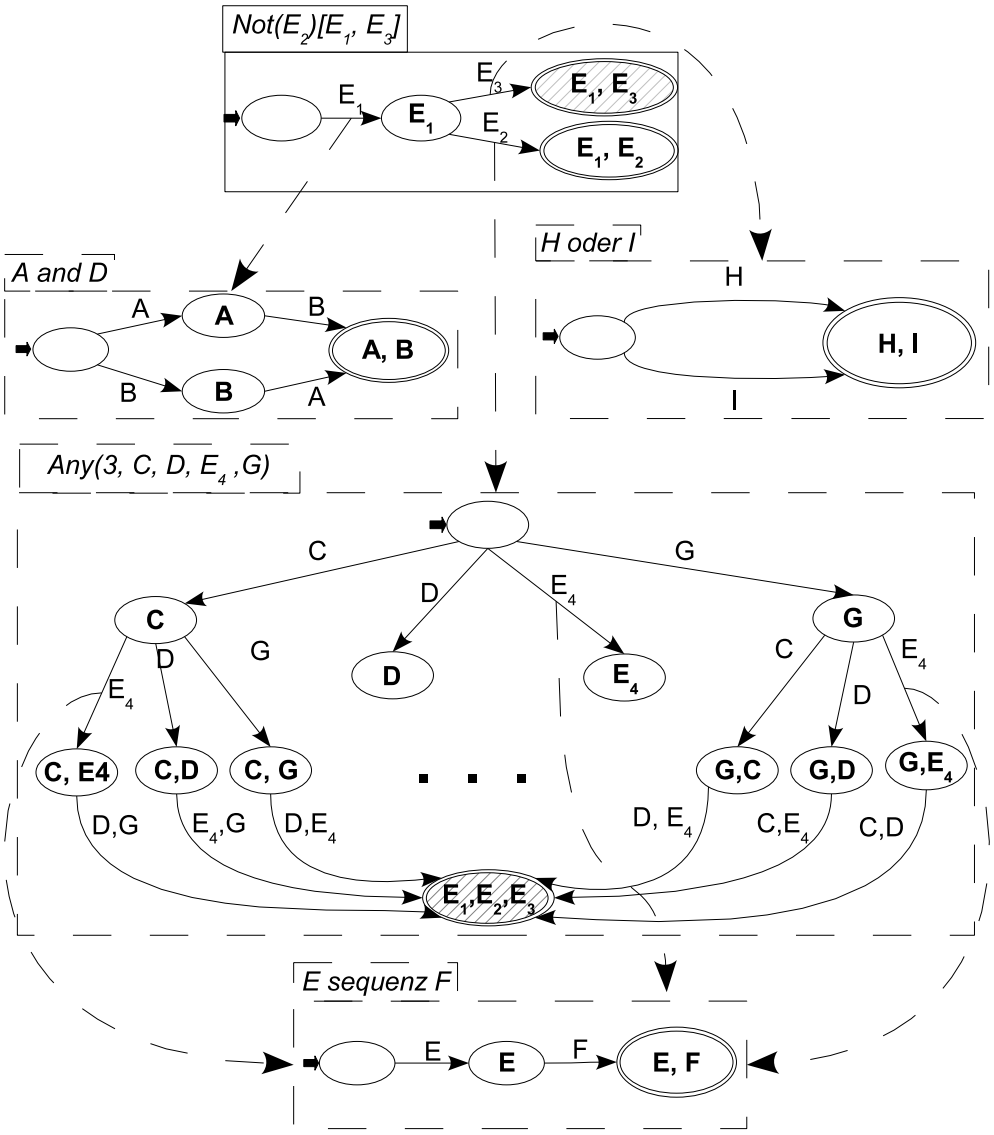
Auch bei diesem Operator wird ein Intervall angegeben. Im Unterschied zu den vorherigen Operatoren darf innerhalb des Intervalls aber ein Event *nicht* auftreten. Nachdem das einleitende Element eingetreten ist, wird gewartet, ob ein Event E_2 oder E_3 eintritt. Ist ersteres der Fall, wird der Automat in den Endzustand (E_1,E_2) überführt. Sollte statt dessen E_3 eintreten, geht der Automat zu (E_1,E_3) über. Zusätzlich wird dann das Not-Event ausgegeben.

Auch bei diesem Automaten wird nur dann eine neue Instanz des Automaten abgespalten, wenn der initiale Zustand verlassen wird. Damit wird wieder erreicht, dass jedes E_1 zu einem weiteren Intervall führt. Bei allen anderen Zustandsübergängen ist das Aufrechterhalten des alten Status eines Automaten nicht nötig. Dies ist so, weil nur das erste Auftreten eines E_2 oder E_1 relevant ist. Daher müssen auch keine neuen Instanzen erzeugt werden.

Automaten für Periodic- und Periodic*-Event Operatoren

Diese Operatoren unterscheiden sich von ihren aperiodischen Varianten in der Art des auslösenden Events. Bei den Periodic-Event-Operatoren ist dieses ein periodisch auftretendes, temporales Event wie z.B. „Alle 20 Minuten“. Da in dieser Arbeit nicht zwischen temporalen und anderen Events unterschieden wird, entstehen dieselben Automaten, wie im Falle eines Aperiodic-Event Operators.

Beispiel 4.8 *Wie in Beispiel 4.7 angekündigt wird, nachdem allgemein gezeigt wurde wie Automaten aussehen, hier der Eventausdruck aus Beispiel 4.2 graphisch als Automatengruppe dargestellt. Die Pfeile zwischen den Automaten stellen dabei deren Abhängigkeiten dar. Der Automat zum Any-Operator ist sehr umfangreich und wird daher nicht vollständig abgebildet.*



4.2.3 XML-Darstellung

Um die Automaten weiterverarbeiten zu können, wird eine XML-Sprache formuliert, die die Automaten repräsentieren kann. Dokumente in dieser Sprache dienen dem später formulierten Algorithmus zur Detektion von zusammengesetzten Events. Für diese Sprache wird der Namespace „http://rewerse.net/event-automaton-ml“ verwendet.

Als Wurzelelement dient „ceda“. Dabei steht ceda für „Composite Event Detection Automatons“. In diesem Element werden dann die Automaten aufgeführt. Da ein Automat in mehreren Instanzen vorliegen kann, wird das Markup getrennt. Die Spezifikation der Automaten wird in <automaton> abgelegt. Die Instanzen mit den aktuellen Zuständen, in denen sie sich befinden, werden durch jeweils ein <instance>-Element dargestellt.

<automaton> enthält eine ID. Diese ermöglicht eine eindeutige Adressierung jedes Automaten.

Ein zweites Attribut namens "time-constraint" nimmt die Information auf, ob die Events gleichzeitig auftreten dürfen oder nicht. Die Kinder eines Eventautomaton-Elements sind die Zustände. Sie werden durch <state> repräsentiert. Zusätzlich kann mit <eventspecification> eine für Menschen lesbare Form des durch den Automaten repräsentierten Eventausdrucks angegeben werden.

Die Zustände enthalten ebenfalls eine ID. Darüber hinaus existieren die Attribute „typ", „detect" und „duplicate". Im Typ wird angegeben, ob es sich um einen initialen Zustand, einen End- oder Abbruchzustand oder keines davon handelt. Die Abbruchzustände sind diejenigen, die einen Automaten abschließen ohne ein Event auszulösen. In „detect" wird angegeben, ob bei Erreichen dieses Zustands ein Event erkannt wurde. „duplicate" steuert das Erzeugen neuer Instanzen des Automaten. Als Inhalt der <state>-Tags kommen nur <transition>-Elemente in Frage.

<transition>-Elemente gibt es in zwei Varianten. Eine davon hat ein atomares Event als Auslöser. Dieses wird innerhalb des Tags angegeben. Der Auslöser der anderen ist ein Composite Event. In diesem Fall gibt es ein Attribut „subautomaton", das die ID des Automaten enthält, der das entsprechende Event nachbildet. In jedem Fall enthält das Attribut „target" die ID des Zustands, in den der Automat übergehen soll.

Andere Kindelemente von <ceda> sind die Instanz-Tags. In diesen wird jeweils der Status einer Instanz eines Automaten abgelegt. Das heißt, sie enthalten den aktuellen Zustand, die bereits eingetretenen Events und die Werte der gebundenen Variablen. In einem solchen Element wird mittels des Attributes „automaton" eine Referenz auf den Automaten angegeben. Ein Status kann die folgenden drei Kinder umfassen: <current−state>, <parameterstore> und <variables>.

<current−state> enthält nur ein Attribut „ref". Damit wird die ID des aktuellen Zustands dieser Automateninstanz angegeben. In <parameterstore> werden alle Events abgelegt, so wie sie aufgetreten sind. <variables> hat als Inhalt einzelne Variablen in Form von <variable>-Tags. Diese Elemente haben ein Attribut „name" mit dem Namen der Variablen. Der Inhalt des Elements entspricht dem Wert der Variablen.

Die weiteren Einzelheiten des Markups können dem folgenden Listing mit einer DTD entnommen werden. Diese DTD ist wieder nicht zum Validieren der Dateien geeignet, da die unbeschränkte Anzahl an möglichen atomaren Events nicht modelliert werden kann.

```
<!ELEMENT ceda (automaton | instance)* >                                    1
<!ATTLIST ceda                                                              2
   version CDATA #FIXED "0.2" >                                             3
<!ELEMENT automaton (eventspecification?, state*) >                         4
<!ATTLIST automaton                                                         5
   id ID #REQUIRED                                                          6
   time−constraint (less|equalorless|non) "non" >                           7
<!ELEMENT eventspecification (#PCDATA) >                                    8
<!ELEMENT state (transition*) >                                             9
<!ATTLIST state                                                             10
```

```
   id ID #REQUIRED                                                          11
   typ (normal|initial|abort|ende) #REQUIRED                                12
   detect (yes|no) #REQUIRED                                                13
   duplicate (yes|no) "no" >                                                14
<!ELEMENT transition ANY >                                                  15
<!ATTLIST transition                                                        16
   typ (composite|primitiv) #REQUIRED                                       17
   target IDREF #REQUIRED                                                   18
   subautomaton IDREF #IMPLIED >                                            19
<!ELEMENT instance (current-state,parameterstore,variables) >               20
<!ATTLIST instance                                                          21
   automaton IDREF #REQUIRED >                                              22
<!ELEMENT current-state EMPTY >                                             23
<!ATTLIST current-state                                                     24
   ref IDREF #REQUIRED >                                                    25
<!ELEMENT parameterstore ANY >                                              26
<!ELEMENT variables (Variable*) >                                           27
<!ELEMENT Variable ANY >                                                    28
<!ATTLIST Variable                                                          29
   name CDATA #REQUIRED >                                                   30
```

Listing 4.3: DTD für Snoop-Automaten

Auch dieser Abschnitt wird wieder mit einem Beispiel beendet. Dazu wird Beispiel 4.2 aufgegriffen.

Beispiel 4.9 *Der Ausdruck aus Beispiel 4.2 ist als Automatenmarkup sehr lang. Aus diesem Grund wird der Ausdruck zu $\neg(A, (E; F), H)$ abgespeckt. Das Ergebnis ist:*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE ceda SYSTEM "./ceda.dtd">
<ceda xmlns="http://rewerse.net/event-automaton-ml">
   <automaton id="AutomatonX" time-constraint="equalorless">
      <eventspecification>Not(foo:EventE; foo:EventF)[foo:EventA, foo:EventH]</eventspecification>
      <state typ="initial" detect="no" duplicate="yes" id="AutomatonX_zustand1">
         <transition target="AutomatonX_zustand2" typ="primitiv">
            <foo:EventA xmlns:foo="/rewerse/event"/>
         </transition>
      </state>
      <state typ="normal" detect="no" duplicate="no" id="AutomatonX_zustand2">
         <transition target="AutomatonX_zustand3" typ="composite" subautomaton="
              AutomatonX_sub-not"/>
         <transition target="AutomatonX_zustand4" typ="primitiv">
            <foo:EventH xmlns:foo="/rewerse/event"/>
         </transition>
      </state>
      <state typ="abort" detect="no" id="AutomatonX_zustand3"/>
      <state typ="ende" detect="yes" id="AutomatonX_zustand4"/>
   </automaton>
```

```
<automaton time-constraint="less" id="AutomatonX_sub-not">
  <eventspecification>foo:EventE; foo:EventF</eventspecification>
  <state typ="initial" detect="no" duplicate="yes" id="AutomatonX_sub-not_zustand1">
    <transition target="AutomatonX_sub-not_zustand2" typ="primitiv">
      <foo:EventE xmlns:foo="/rewerse/event"/>
    </transition>
  </state>
  <state typ="normal" detect="no" duplicate="yes" id="AutomatonX_sub-not_zustand2">
    <transition target="AutomatonX_sub-not_zustand3" typ="primitiv">
      <foo:EventF xmlns:foo="/rewerse/event"/>
    </transition>
  </state>
  <state typ="ende" detect="yes" id="AutomatonX_sub-not_zustand3"/>
</automaton>
<instance automaton="AutomatonX_sub-not">
  <current-state ref="AutomatonX_sub-not_zustand1"/>
  <parameterstore/>
  <variables/>
</instance>
<instance automaton="AutomatonX">
  <current-state ref="AutomatonX_zustand1"/>
  <parameterstore/>
  <variables/>
</instance>
</ceda>
```

Das <automaton>- und <instance>-Elements mit der ID ‚AutomatonX' bilden den NOT-Operator ab. Die Elemente zum Sequenz-Operator haben die ID ‚AutomatonX_sub-not'.

4.3 Transformationen

Um die Vorteile beider, des deklarativen und des automatenbasierten Markups zu kombinieren, ist eine automatische Transformation nötig. Damit ist es möglich die Events auf einfache Art und Weise zu definieren und trotzdem in einer geeigneten aber komplexeren Form zur Weiterverarbeitung vorliegen zu haben. Für die Transformation bietet sich XSLT an.

Durch die in Abschnitt 4.2 festgestellten Eigenschaften der Automaten liegt es nahe, innerhalb des XSLT-Stylesheets für jeden Operator ein Template zu erstellen. Diese erzeugen aus den Automatenvorlagen und dem zu transformierenden Ausdruck die Automaten. Die Verarbeitung folgt dabei der Struktur des Snoop-Ausdrucks. Diese Baumstruktur wird dabei in Preorder-Traversierung durchlaufen. Zuerst wird die Wurzel verarbeitet, danach ihre Kinder. Diese werden wieder in gleicher Weise behandelt.

Die Umwandlung eines Operators erfolgt immer auf die selbe Art. Aus der Vorlage wird ein Automat erzeugt. In diesem werden die Events aus der Vorlage durch die Operanden des aktuell

bearbeiteten Operators ersetzt. Für die Operanden, die wiederum Operatoren sind, werden Namen erzeugt. Die Transitionen, mit den betroffenen Operanden als Auslöser, erhalten eine Referenz auf einen Automaten mit dem jeweils erzeugten Namen. Zwar existieren diese Automaten noch nicht, da die zugehörigen Operatoren tiefer im Operatorbaum stehen, aber sie werden noch mit den erzeugten Namen angelegt. Die Operanden, die nur aus atomaren Events bestehen, werden als Inhalt in die Transition mit aufgenommen. Nachdem der Automat so angepasst wurde, werden die Templates für die Kinder im Operatorbaum aufgerufen, zumindestens wenn diese keine Blätter, also atomare Events, sind. Diese bedürfen keiner weiteren Bearbeitung. Nachdem das Stylesheet die Kinder des aktuellen Operatorbaums bearbeitet hat, instantiiert es den erzeugten Automaten. Das bedeutet, es wird ein <instance>-Element erzeugt. Falls Variablenbindungen zu dem Eventausdruck gegeben sind, werden mehrere solcher Instanzen erstellt. Jede enthält einen anderen Satz an Variablen. Sofern keine Variablen vorbelegt sind, wird eine einzelne Instanz erzeugt, die keine Variablen enthält.

Das Verhalten wird jetzt mit dem Template des AND-Operators beispielhaft noch einmal aufgezeigt. Dazu hier erst einmal das Template:

```
<xsl:template match= "//snoop:And" >                                                    52
  <xsl:param name= "AUTOMATON_ID" />                                                    53
  <!-- Erzeugen des Automaten zu einem AND-Operator. -->                                54
  <ceda:automaton time-constraint= "equalorless">                                       55
    <xsl:attribute name= "id" select= "$AUTOMATON_ID" />                                56
    <ceda:eventspecification> AND </ceda:eventspecification>                            57
    <ceda:state typ= "initial" detect= "no" duplicate= "yes" >                          58
      <xsl:attribute name= "id">                                                        59
        <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</xsl:text>          60
      </xsl:attribute>                                                                  61
      <ceda:transition>                                                                 62
        <xsl:attribute name= "target">                                                  63
          <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</xsl:text>        64
        </xsl:attribute>                                                                65
        <xsl:choose>                                                                    66
        <xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">                67
          <xsl:attribute name= "typ">primitiv</xsl:attribute>                           68
          <xsl:copy-of select= "./child::*[1]/child::*[1]" />                           69
        </xsl:when>                                                                     70
        <xsl:otherwise>                                                                 71
          <xsl:attribute name= "typ">composite</xsl:attribute>                          72
          <xsl:attribute name= "subautomaton">                                          73
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub1</xsl:text>          74
          </xsl:attribute>                                                              75
        </xsl:otherwise>                                                                76
        </xsl:choose>                                                                   77
      </ceda:transition>                                                                78
      <ceda:transition>                                                                 79
```

```
<xsl:attribute name= "target">                                                    80
  <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text>          81
</xsl:attribute>                                                                  82
<xsl:choose>                                                                      83
<xsl:when test= "./child::*[2][name(.) = 'snoop:Atomic-Event']">                  84
  <xsl:attribute name= "typ">primitiv</xsl:attribute>                             85
  <xsl:copy-of select= "./child::*[2]/child::*[1]" />                             86
</xsl:when>                                                                       87
<xsl:otherwise>                                                                   88
  <xsl:attribute name= "typ">composite</xsl:attribute>                            89
  <xsl:attribute name= "subautomaton">                                            90
    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub2</xsl:text>            91
  </xsl:attribute>                                                                92
</xsl:otherwise>                                                                  93
</xsl:choose>                                                                     94
</ceda:transition>                                                                95
</ceda:state>                                                                     96
<ceda:state typ= "normal" detect= "no" duplicate= "yes" >                         97
  <xsl:attribute name= "id">                                                      98
    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</xsl:text>        99
  </xsl:attribute>                                                                100
  <ceda:transition>                                                               101
    <xsl:attribute name= "target">                                                102
      <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>      103
    </xsl:attribute>                                                              104
    <xsl:choose>                                                                  105
    <xsl:when test= "./child::*[2][name(.) = 'snoop:Atomic-Event']">              106
      <xsl:attribute name= "typ">primitiv</xsl:attribute>                         107
      <xsl:copy-of select= "./child::*[2]/child::*[1]" />                         108
    </xsl:when>                                                                   109
    <xsl:otherwise>                                                               110
      <xsl:attribute name= "typ">composite</xsl:attribute>                        111
      <xsl:attribute name= "subautomaton">                                        112
        <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub2</xsl:text>        113
      </xsl:attribute>                                                            114
    </xsl:otherwise>                                                              115
    </xsl:choose>                                                                 116
  </ceda:transition>                                                              117
</ceda:state>                                                                     118
<ceda:state typ= "normal" detect= "no" duplicate= "yes" >                         119
  <xsl:attribute name= "id">                                                      120
    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text>        121
  </xsl:attribute>                                                                122
  <ceda:transition>                                                               123
    <xsl:attribute name= "target">                                                124
      <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>      125
    </xsl:attribute>                                                              126
    <xsl:choose>                                                                  127
    <xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">              128
```

```
                <xsl:attribute name= "typ">primitiv</xsl:attribute>                                    129
                <xsl:copy-of select= "./child::*[1]/child::*[1]" />                                    130
            </xsl:when>                                                                                131
            <xsl:otherwise>                                                                            132
                <xsl:attribute name= "typ">composite</xsl:attribute>                                   133
                <xsl:attribute name= "subautomaton">                                                   134
                    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub1</xsl:text>                 135
                </xsl:attribute>                                                                       136
            </xsl:otherwise>                                                                           137
            </xsl:choose>                                                                              138
        </ceda:transition>                                                                             139
    </ceda:state>                                                                                      140
    <ceda:state typ= "ende" detect= "yes" >                                                            141
        <xsl:attribute name= "id">                                                                     142
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>                     143
        </xsl:attribute>                                                                               144
    </ceda:state>                                                                                      145
</ceda:automaton>                                                                                      146
<!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->               147
<xsl:apply-templates select= "./child::*[1][not(name(.) = 'snoop:Atomic-Event')]">                      148
    <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$                                       149
        AUTOMATON_ID" /><xsl:text>_sub1</xsl:text></xsl:with-param>
</xsl:apply-templates>                                                                                 150
<xsl:apply-templates select= "./child::*[2][not(name(.) = 'snoop:Atomic-Event')]">                      151
    <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$                                       152
        AUTOMATON_ID" /><xsl:text>_sub2</xsl:text></xsl:with-param>
</xsl:apply-templates>                                                                                 153
<!-- Erzeugen der initialen Instanzen. -->                                                             154
<xsl:call-template name="createInitialInstance">                                                       155
    <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />                                     156
</xsl:call-template>                                                                                   157
</xsl:template>                                                                                        158
```

Listing 4.5: Template für die Erzeugung eines AND-Operators

Das Template wird aufgerufen, wann immer ein <snoop:And> durch das Stylesheet aufgerufen wird (Zeile 52). Der Name des zu erzeugenden Automaten wird im Parameter „AUTOMATON_ID“ übergeben (Zeile 53). Der Rest des Templates teilt sich in drei Blöcke. Zuerst wird der Automat erzeugt (Zeilen 55 – 146), danach die Verarbeitung für die im Operatorbaum folgenden Operatoren angestoßen (Zeilen 148 – 153) und zuletzt wird der erzeugte Automat initialisiert (Zeilen 155 – 157).

Die Erzeugung des Automaten bildet die Vorlage für den And-Automaten ab, indem die Zustände und Transitionen in ihrer jeweiligen XML-Representation erzeugt und mit ihren Attributen versehen werden. Bei jeder Transition wird geprüft, ob der entsprechende Operand des Ausdrucks aus einem atomaren Event besteht. In diesem Fall wird dieses Event in das <transition>-Element eingeführt ohne weiter bearbeitet zu werden. Anderenfalls ist das Event vom Typ Composite. Um

dieses Event zu bearbeiten, wird ein Name erzeugt. Dieser Name wird in der Referenz der Transition auf einen anderen Automaten verwendet. Nachdem alle Zustände und Transitionen auf diese Art erzeugt wurden, ist der Automat fertig.

Im zweiten Schritt werden für die Operanden, die keine atomaren Events sind, die Templates aufgerufen, die die entsprechenden Teilausdrücke in Automaten überführen. Als Namen für diese Automaten werden diejenigen verwendet, die bereits beim Erstellen der Transitionen erzeugt wurden.

Für den letzten Schritt wird ein anderes Template aufgerufen:

```
<xsl:template name= "createInitialInstance" >                                    581
  <xsl:param name= "AUTOMATON_ID" />                                             582
  <xsl:param name= "INITIAL_STATE" select= "'_zustand1'"/>                       583
                                                                                 584
  <xsl:choose>                                                                   585
  <!-- Wenn Variablen vorbelegt sind, dann soll für jedes Variablen-Tuple eine Instanz erzeugt   586
      werden. -->
  <xsl:when test= "/snoop:Eventdeclaration/eca:variable-bindings" xmlns:eca= "http://www.eca.   587
      org/eca-ml" >
    <xsl:for-each select="/snoop:Eventdeclaration/eca:variable-bindings/eca:tuple" >           588
      <ceda:instance>                                                            589
        <xsl:attribute name= "automaton"><xsl:value-of select= "$AUTOMATON_ID" /><   590
            /xsl:attribute>
        <ceda:current-state>                                                     591
          <xsl:attribute name= "ref">                                            592
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:value-of select= "$     593
                INITIAL_STATE" />
          </xsl:attribute>                                                       594
        </ceda:current-state>                                                    595
        <ceda:parameterstore />                                                  596
        <ceda:variables>                                                         597
          <xsl:copy-of select= "./eca:variable" />                               598
        </ceda:variables>                                                        599
      </ceda:instance>                                                           600
    </xsl:for-each>                                                              601
  </xsl:when>                                                                    602
  <!-- Anderenfalls muss nur eine Instanz ohne bestehende Variablenbindungen erzeugt werden. -->   603
  <xsl:otherwise>                                                                604
    <ceda:instance>                                                              605
      <xsl:attribute name= "automaton"><xsl:value-of select= "$AUTOMATON_ID" /></   606
          xsl:attribute>
      <ceda:current-state>                                                       607
        <xsl:attribute name= "ref">                                              608
          <xsl:value-of select= "$AUTOMATON_ID" /><xsl:value-of select= "$       609
              INITIAL_STATE" />
        </xsl:attribute>                                                         610
      </ceda:current-state>                                                      611
      <ceda:parameterstore />                                                    612
```

```
        <ceda:variables />                                    613
      </ceda:instance>                                        614
    </xsl:otherwise>                                          615
    </xsl:choose>                                             616
</xsl:template>                                               617
```

Listing 4.6: Template für die Initialisierung eines Automaten

Dieses Template erzeugt die einzelnen <instance>-Elemente, die auf den Automaten verweisen, der in „AUTOMATON_ID“ angegeben wird (Zeile 582). Dazu wird überprüft, ob Variablenbindungen zum Eventausdruck existieren (Zeile 587). Wenn ja, wird für jeden Satz an Variablen (Zeile 588) eine neue Instanz erzeugt (Zeilen 589 – 600). Wenn keine Variablen vorbelegt sind, wird nur ein <instance>-Element erzeugt, das keine Variablen enthält (Zeilen 65 – 614). Der aktuelle Zustand jeder Instanz ist der initiale. Welcher das ist, kann über den Parameter „INITIAL_STATE“ angegeben werden (Zeile 583). Der jeweilige Parameterstore ist leer. Damit ist die Transformation eines (Teil-)Ausdrucks vollständig und kann verwendet werden.

Die Verarbeitung anderer Operatoren erfolgt auf die gleiche Weise. Wenn man davon absieht, dass andere Automatenvorlagen verwendet werden, unterscheiden sich die Templates nicht. Natürlich gibt es zwei Ausnahmen zu dieser Aussage. Die Automaten zu den Operatoren ANY und ANY* sind abhängig von deren Parametern. Deshalb erfolgt dort eine dynamische Anpassung der Vorlagen an die Parameter. Dabei werden die einzelnen Zustände und Transitionen auf dieselbe Weise wie bisher erstellt. Nur ihre Anzahl ist für verschiedene Eventausdrücke unterschiedlich.

Damit alle Automaten einen eindeutigen Namen erhalten, folgt ihre Benennung einem einfachen Schema. Der Basisname, der auch für den Automaten verwendet wird, der aus der Wurzel des Operatorbaums hervorgeht, wird als Parameter beim Aufruf des Stylesheets mit angegeben. Wenn festgestellt wird, dass einer der Operanden ein Subausdruck ist, wird dem Basisnamen ein Unterstrich ‚_‘ und ein zusätzlicher Bezeichner angehängt. Dieser neue Name ist dann der Basisname des Automaten, der aus dem Teilausdruck hervorgeht. Dadurch, dass der Basisname immer erweitert wird, kann er nicht zweimal vorkommen. Die zusätzlichen Bezeichner, die angehängt werden, sind von Operator zu Operator unterschiedlich. Im Falle eines „Und“ wird für den ersten Operanden „sub1“ und für den zweiten „sub2“ angehängt.

Das gesamte Stylesheet mit allen Templates kann Anhang C.2 entnommen werden. Hinweise zur Verwendung des Stylesheets sind in Anhang A.1 enthalten.

Kapitel 5

Detektionsalgorithmus

Dieses Kapitel beschreibt den Algorithmus zum Erkennen von kombinierten Events in einer implementationsunabhänigen, abstrakten Form. Dabei wird – sehr konkret – das automatenbasierte XML-Markup aus dem vorangegangen Kapitel verwendet. Ein Dokument in dieser Sprache dient zusammen mit einem atomaren Event als Eingabe des Algorithmus. Am Ende soll ein weiteres XML-Dokument stehen, das die Situation beschreibt, in der sich die Automaten nach der Verarbeitung des Events befinden. Dazu soll eine Liste der detektierten Composite Events ausgegeben werden. Damit steht das Ein- und Ausgabeverhalten des Algorithmus fest, es fehlt nur noch das „Wie“: Wie soll die Eingabe zu der gewünschten Ausgabe verarbeitet werden? Am Ende dieses Kapitels soll darüber Klarheit bestehen. Dazu wird zuerst der Begriff der „Situation“ vorgestellt. Danach wird über den Ablauf des Algorithmus gesprochen. Abschließend wird detailliert darauf eingegangen, wann ein atomares Event als Auslöser eines Zustandsübergangs verwendbar ist.

5.1 Situationen

Der Begriff der Situation bezeichnet die Menge aller Informationen, die während einem Zeitpunkt der Detektion vorliegen. Dazu gehören die Eventspezifikationen in ihrer Form als Automaten, deren Instanzen und die Events, die seit dem zuletzt eingetretenen atomaren Event detektiert wurden. Dieses atomare Event ist ebenfalls Bestandteil der Situation.

Die Automaten bilden die Basis der Eventdetektion. Sie legen fest, nach welchen Composite Events überhaupt gesucht werden soll. Zu jedem Automaten gibt es mehrere Instanzen mit jeweils einem aktuellen Zustand und weiteren Informationen. Die Zustände, in denen sich die Automaten befinden, repräsentieren den aktuellen Stand der Detektion. Die Events hingegen sind das Ergebnis dieses Vorgangs. Events, die vollständig verarbeitet wurden, also nichts mehr zur Detektion weiterer Events beitragen können, werden aus der Situation entfernt.

Eine Situation ändert sich grundsätzlich nicht, es sei denn, ein atomares Event tritt ein. In diesem Fall wird ein Algorithmus gestartet, der die bestehende Situation in eine neue überführt. Dabei können zwischenzeitlich mehrere Situationen entstehen, die eine unvollständige Verarbeitung des atomaren Events repräsentieren.

5.2 Ablauf des Algorithmus

Wenn bisher von einem Detektionsalgorithmus gesprochen wurde, war einer gemeint, der aus einer Abfolge von atomaren Events alle möglichen kombinierten Events zu den angegebenen Eventausdrücken „berechnet“. Stattdessen wird zu einem Algorithmus übergegangen, der basierend auf einer Situation und einem atomaren Event inkrementell eine neue Situation bestimmt. Hier spielt eine entscheidende Rolle, dass man die Eventausdrücke als Automaten darstellen kann. Denn damit hat man die Möglichkeit, die benötigten Situationen zu formulieren. Wie das geht wurde bereits in Kapitel 4.2.1 und im ersten Abschnitt dieses Kapitels erläutert. Die Situation enthält die Ergebnisse der vorherigen Berechnungen. Beim Eintreten eines weiteren atomaren Events muss daher nicht mehr die Detektion für die bisherigen und das neue Event durchgeführt werden. Stattdessen werden nur die Einflüsse des hinzugekommenen, atomaren Events auf die bestehende Situation bestimmt.

Der Algorithmus wird für jeweils ein atomares Event angestoßen. Zu Beginn wird die aktuelle Situation eingelesen. Danach wird eine Funktion aufgerufen, die Events auf Automaten anwenden kann. Es bedarf dann mehrerer rekursiver Durchläufe dieser Funktion bis das endgültige Ergebnis feststeht. In den einzelnen Durchläufen wird jeweils eine neue Zwischensituation erzeugt, die im darauf folgenden Aufruf weiterverarbeitet wird. Bei jedem Aufruf der Funktion wird ein Liste der Events angegeben, die auf die Automaten angewandt werden sollen. Im ersten Aufruf enthält diese Liste nur das atomare Event. Jeder weitere Aufruf erhält die zusammengesetzten Events, die im vorangegangenen Schritt detektiert wurden. Die Funktion wird solange ausgeführt bis sich an der Situation nichts mehr ändert. Das ist gleichbedeutend damit, dass seit der letzten Situation keine Zustandsübergänge mehr vorgenommen wurden und daher auch keine zusätzlichen Events detektiert werden konnten. Die Situation am Ende des letzten Schrittes entspricht dann dem gewünschten Ergebnis. Zum einen enthält diese abschließende Situation die neuen Zustände der Automaten mit denen die Detektion weiterer Events später wieder aufgenommen werden kann und zum anderen die eingetretenen Composite Events.

Der Grund, warum die Verarbeitung der Funktion mehrfach vorgenommen wird, liegt in der Spezifikation von Snoop begründet. Ein Snoop-Event tritt zu genau dem Zeitpunkt ein, in dem das atomare Event, das die Detektion beendet, eingetreten ist.

Beispiel 5.1 *Ein Event gemäß dem Ausdruck $A;B$ tritt, vorausgesetzt A ist bereits einmal eingetreten, genau zu dem Zeitpunkt ein an dem B eintritt.*

Das bedeutet, alle Events, die durch einen Aufruf des Algorithmus bestimmt werden, treten zu exakt dem selben Zeitpunkt ein. Daraus ergibt sich, dass alle Situationen, die im Verlauf des Algorithmus entstehen, die Detektion zu ein und demselben Zeitpunkt beschreiben. Die aktuellen Zustände der Automateninstanzen existieren theoretisch alle gleichzeitig, obwohl sie vom Algorithmus nacheinander erreicht werden.

Beispiel 5.2 *Der Ausdruck $A\Delta(B;A)$ sei gegeben. Zum Zeitpunkt t_1 ist B bereits eingetreten. Zum jetzigen Zeitpunkt t_2 tritt A ein. Da B bereits eingetreten war, wird das Event (B;A) durch den zugehörigen Automaten detektiert. Es tritt zum Zeitpunkt t_2 auf. In demselben Durchlauf wird der Automat für $A\Delta E_{B;A}$ in einen neuen Zustand geschaltet, denn er reagiert ebenfalls auf das A. Erst ein zweiter Durchlauf kann dann feststellen, dass das Event $A\Delta E_{B;A}$ ebenfalls eintritt. Die Events A und (B;A) treten zwar zum gleichen Zeitpunkt auf, aber das ist gemäß dem AND-Operator zulässig.*

Da die sequentiell detektierten Events gleichzeitig eintreten, müssen sie rückwirkend auf die im bisherigen Verlauf bereits geänderten Zustände angewandt werden, um die Semantik von Snoop zu erhalten. Andererseits müssen aber auch Events aus bisherigen Schritten auf neu aufgetretene Zustände angewendet werden. Um festzuhalten, welches Event auf welche Instanz bereits angewendet wurde, wird ihnen jeweils eine Zahl zugeordnet, die angibt in welchem Schritt sie aufgetreten sind. Die Instanzen, die zu Beginn des Algorithmus vorliegen, erhalten die Nummer 0. Wie genau diese verwendet werden kann, wird am Ende des Abschnitts 5.2.1 beschrieben.

5.2.1 Ablauf der Funktion

Die eigentliche Detektion erfolgt in einer rekursiven Funktion. Diese untersucht jede Automateninstanz, ob eines der vorliegenden Events auf sie anwendbar ist. Sofern dieses zutrifft, wird der aktuelle Zustand der Instanz geändert.

Die Verarbeitung erfolgt in drei Schleifen. Die äußere Schleife durchläuft die Instanzen der aktuellen Situation. Es werden allerdings nicht alle Instanzen weiterverarbeitet. Die Auswahl erfolgt über ein Attribut des zugehörigen Automaten. Im Attribut „time-constraint“ wird angegeben, ob die einzelnen Events eines Ausdrucks gleichzeitig oder nacheinander eintreten müssen. Automateninstanzen zu Operatoren, die das gleichzeitige Eintreten erlauben, werden immer weiterverarbeitet. Wenn verlangt wird, dass die Operanden nacheinander eintreten, kommen nur diejenigen Instanzen in Frage, die seit dem Start noch nicht verändert wurden, also im Schritt 0 bereits existierten.

Für jede der ausgewählten Instanzen wird der aktuelle Zustand bestimmt und jede seiner Transitionen mit jedem atomaren und zusammengesetzten Event verglichen. Auf den Vergleich

mit atomaren Events wird in Abschnitt 5.3 detailliert eingegangen. Ein kombiniertes Event ist dann verwendbar, wenn die ID des Automaten, der es detektiert hat, zur entsprechenden Angabe bei der Transition passt. Zusätzlich müssen die gleichnamigen Variablen im Event und in der Automateninstanz in ihren Werten übereinstimmen. Weitere, nur im Event oder der Instanz, vorhandene Variablenbindungen spielen keine Rolle für die Anwendbarkeit eines Events.

Wenn ein Event gefunden wurde, das zu einer Transition passt, wird der Zustandsübergang durchgeführt. Dazu wird eine neue Instanz mit dem neuen Zustand erzeugt. Die alte Instanz bleibt mit unverändertem Zustand erhalten. Allerdings wird sie als ‚instabil' gekennzeichnet. Da die alten Instanzen erhalten bleiben, können auch in einem späteren Schritt erkannte Events noch darauf angewandt werden. Durch die Markierung wird erreicht, dass der Status nicht ins endgültige Ergebnis aufgenommen wird. Ein weiterer Grund, den ursprünglichen Zustand zu erhalten, ergibt sich daraus, dass Events in Snoop nicht verbraucht werden. Das heißt, obwohl ein Event bereits für die Detektion eines zusammengesetzten Events herangezogen wurde, kann es noch zu weiteren Events beitragen. Das führt dazu, dass die Detektion in jedem Zustand fortgesetzt werden kann, der einmal aufgetreten ist.

Beispiel 5.3 *Betrachtet man den Ausdruck $B;A$ und nimmt an, bisher wäre das atomare Event B eingetreten. Aus dem Ausdruck wird ein Automat $\mathcal{A}^{SEQ}$ erzeugt. Nachdem B eingetreten ist, liegen zwei Automateninstanzen vor. Eine im initialen Zustand des Automaten und eine, die das Auftreten von B symbolisiert. Sollte jetzt ein Event A_1 zu A eintreten, wird der Automat in einen neuen Zustand überführt. Wenn allerdings der ursprüngliche Zustand nicht erhalten bleibt, würde ein später eintretendes A_2 nicht erneut zum Auslösen eines Events führen. Dieses Verhalten ist jeder in Snoop vorgesehen.*

Allerdings gibt es bei den Operatoren A, A*, P, P* und NOT ein Intervall, in dem die gewünschten Events auftreten müssen. Sobald das Intervall abgeschlossen wird, dürfen die vorherigen Zustände nicht in die endgültige Situation übernommen werden. Ob die alten Zustände erhalten bleiben, wird für jeden Zustand mittels des Attributs „duplicate" in den Automaten angegeben. Einen Sonderfall im Hinblick auf das Duplizieren von Automateninstanzen sind Zustände vom Typ „Abbruch". Solche Zustände unterbinden das Duplizieren der Instanzen, von denen ein Zustandsübergang ausgeht.

Kombiniert man beide Gründe zum Vervielfältigen der Instanzen, erhält man folgendes Verhalten: Eine neue Instanz mit dem transformierten Zustand wird immer erzeugt. Im Verlauf des Algorithmus bleibt darüber hinaus immer die alte Instanz erhalten. Wenn der aktuelle Zustand dieser Instanz dupliziert werden muss, geht er auch in die endgültige Situation am Ende des Algorithmus ein. Anderenfalls wird sie aus dieser Situation entfernt, bevor sie ausgegeben wird. Sollte allerdings das Ziel einer Transition ein Abbruch-Zustand sein, wird die ursprüngliche Instanz auf

keinen Fall in die abschließende Situation aufgenommen.

Wenn jede Automateninstanz so behandelt wurde, steht das Ergebnis des Funktionsaufrufs fest. Es entspricht damit einer meist unvollständigen Situation. Diese Situation enthält alle bisherigen Instanzen und die Events, die seit Beginn des Algorithmus detektiert wurden. Dieses Ergebnis dient dem nächsten Aufruf der Funktion als Eingabe.

Da die Instanzen und Events aus dem vorangegangenen Schritt auch im nächsten vorliegen, würden sie immer noch einmal bearbeitet und so zu einer Reihe von identischen und damit überflüssigen Instanzen führen. Daher wird zusätzlich zu dem oben beschriebenen Vorgehen eine weitere Auswahl der Events und Instanzen getroffen, die weiterverarbeitet werden. Da jedes dieser Elemente eine Nummer enthält, mit der angegeben wird, in welchem Schritt es in die Situation aufgenommen wurde, ist es möglich diese Mehrfachverarbeitung zu verhindern: In jedem Schritt wird versucht die Events aus dem letzten Schritt auf alle Instanzen anzuwenden und alle Events auf die neuen Instanzen aus dem letzten Schritt. Damit werden nur diejenigen Zustandsübergänge ausgeführt, die wirklich zu neuen Ergebnissen führen.

Ein Teil der so durchgeführten Transformationen führt allerdings zu Instanzen, die sich nur darin unterscheiden, in welcher Reihenfolge die Events angewandt wurden. Solche Instanzen sind ebenfalls ungewollt.

Beispiel 5.4 *Veranschaulichen kann man sich das Problem an dem Eventausdruck* $A\Delta(B;A)$. *Aus diesem Ausdruck entstehen zwei Automaten:*
$\mathcal{A}^1 = A\Delta\mathcal{E}_{SEQ}$
$\mathcal{A}^2 = B;A$
Das atomare Event B sei bereits eingetreten, wenn A die Event-Engine erreicht. In dieser Situation wird A auf $\mathcal{A}^1$ *und auf* $\mathcal{A}^2$ *angewandt. Aus* $\mathcal{A}^2$ *entsteht so ein Event* $\mathcal{E}_{SEQ}$. *Im nächsten Schritt wird* $\mathcal{E}_{SEQ}$ *auf die ursprüngliche und die neue Instanz von* $\mathcal{A}^1$ *angewandt. Daraus entstehen einerseits das Event* $\mathcal{E}_{A\Delta(B;C)}(A, E_{SEQ})$ *und andererseits die Instanz* $\mathcal{A}^1_3$. *Diese Instanz symbolisiert, dass (B;A) eingetreten ist. Damit würde das erwünschte Ergebnis vorliegen, aber es wird noch ein weiterer Schritt durchgeführt. In diesem Schritt wird das atomare Event A auf die Automateninstanz* $\mathcal{A}^1_3$ *angewandt, wodurch wir ein weiteres Event* $\mathcal{E}_{A\Delta(B;C)}$ *erhalten. In ihm wurde* $\mathcal{E}_{SEQ}$ *vor A verarbeitet.*

Dieses Verhalten beruht darauf, dass Events aus länger zurückliegenden Schritten auch auf später erzeugte Instanzen angewandt werden. Daher soll dies näher untersucht werden. Bei der Sequenz und dem Any^*-Operator kommt es gar nicht erst dazu, da dort nur die ursprünglichen Instanzen verarbeitet werden. Die aus ihnen resultierenden, neuen Instanzen werden also gar nicht weiter behandelt. Im Fall des Oder's führt jede Transition zu einem Event. Neue Automateninstanzen entstehen hier also ebenfalls nicht. Diese Operatoren werden daher im Folgenden ignoriert. Das

Problem lässt sich gut anhand der Grafiken zu den Automaten lokalisieren. Der bisherige Ablauf des Automaten führt dazu, dass, wenn ein Event detektiert wird, alle parallelen Pfade verfolgt werden. Daher kommt die unterschiedliche Reihenfolge der Events bei ansonsten Identischen Events.

Bei einer Und-Verknüpfung von Events wird, wie im obigen Beispiel zu sehen, nur eine Vertauschung der Reihenfolge, in der die Events verarbeitet werden, erreicht. Das gleiche gilt für den ähnlich strukturierten ANY-Operator. In den Automaten zu den Operatoren P, A und Not gibt es keine parallelen Pfade über die ein Event ausgelöst werden könnte. Daher gibt es hier keine Auswirkungen. Bei den Operatoren P^* und A^* tritt durch die bisherige Vorgehensweise sogar ein Fehler auf. Dadurch, dass dieser Automat einen Zustand besitzt, der in sich selbst transformiert werden kann, führt das Auftreten eines passenden Events zu einer Endlosschleife, da das gleiche Event immer wieder auf die neu erzeugte Instanz angewandt wird. Ansonsten liegen bei diesen Operatoren ebenfalls keine Parallelitäten vor.

Daraus ergibt sich, dass in jedem Schritt nur diejenigen Events für die Verarbeitung herangezogen werden müssen, die im vorangegangenen Schritt detektiert wurden.

5.3 Vergleich zwischen Atomaren Events und Mustern

Ein wesentlicher Bestandteil des Algorithmus ist der Vergleich zwischen den eingetretenen atomaren Events und den Event-Mustern, die in den Snoop-Events angegeben sind. Dabei ist es zusätzlich nötig auf eventuell vorhandene Variablen zu achten.

5.3.1 Die Muster

Als Muster für ein Event werden alle XML-Fragmente interpretiert, die bei der deklarativen Spezifikation der zusammengesetzten Events innerhalb eines Elements vom Typ „Atomic-Event“ stehen. Erlaubt ist ein beliebiges XML-konformes Markup. Dabei sind alle Elemente, Attribute, und Texte des Musters verpflichtend für die Events. Das heißt, jeder dieser Knoten muss in einem Event vorhanden sein um das Muster zu erfüllen. Im einfachsten Fall ist ein Muster also ein einzelnes Tag.

Beispiel 5.5 (einfachstes Muster)
<travel:cancel−flight/>

In diesem Fall trifft auf das Muster jedes atomare Event zu, das aus dem Tag <travel:cancel−flight> und beliebigem Inhalt innerhalb dieses Tags besteht. Genauer gesagt, muss die URL des Namespace „travel“ und der lokale Name „cancel-flight“ mit den entsprechenden Werten des Wurzelelements des Events übereinstimmen.

Um ein Event genauer zu spezifizieren, ist es möglich dem Muster Attribute hinzuzufügen.

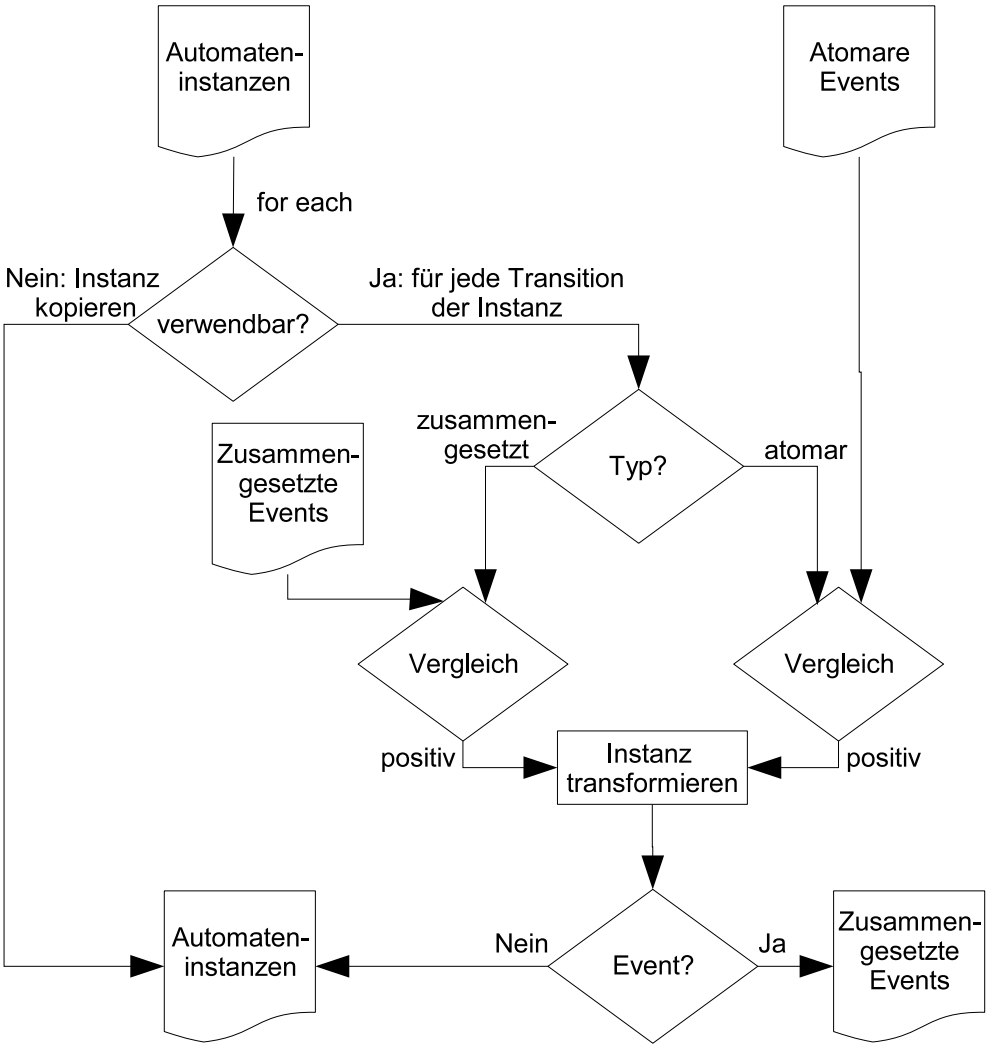


Abbildung 5.1: Ablauf eines Funktionsaufrufs

Beispiel 5.6 (Muster mit Attributangabe)

<travel:cancel−flight code="G7654−12"/>

Damit werden nur noch die Events akzeptiert, die zusätzlich zu obigen Bedingungen in ihrem Wurzelelement ein Attribut mit dem namen „code“, dem gleichen Namespace und dem Wert „G7654-12“ besitzen. Auch Namespaces, die abweichend vom Namespace des umgebenden Elements sind, werden somit für die Erkennung der Events herangezogen.

Dadurch, dass das Muster gültiges XML sein muss, ergibt sich im Bereich der Attribute eine Einschränkung. Wenn man alle Events verarbeiten möchte, die ein gewisses Attribut besitzen, ohne dabei dessen Wert zu berücksichtigen, müsste man diesen im Muster weglassen. Da aber <travel:cancel−flight code/> in XML nicht zulässig ist, kann das gewünschte Muster nicht erzeugt werden. Allerdings gibt es durch die Verwendung einer Variablen doch eine Möglichkeit:

Beispiel 5.7 (Muster mit Variablenbindung eines Attributwertes)

<travel:cancel−flight code="$flight"/>

In diesem Fall wird jeder Inhalt für das Attribut „code“ akzeptiert. Voraussetzung ist natürlich, dass die Variable „flight“ nicht an anderer Stelle bereits gebunden wird. Näheres zu Variablen in den Event-Mustern wird im nächsten Abschnitt beschrieben.

Um die relevanten Events noch weiter zu spezifizieren, besteht die Möglichkeit Kindelemente und Textknoten anzugeben. Die Kinder sind dabei genauso anzugeben wie das Wurzelelement:

Beispiel 5.8 (Event-Muster mit Kindelement)

<travel:cancel−flight>
    <travel:code />
</travel:cancel−flight>

In diesem Beispiel würde jedes Event auf das Muster passen, das wie bisher aus der Wurzel <travel:cancel−flight> und einem direkten Nachfahren <travel:code> besteht. Für die Kinder gelten die gleichen Bedingungen, die auch für das Wurzelelement gelten. Sie müssen in Namespace, Name und den Attributen übereinstimmen. Eine beliebig tiefe Schachtelung von Elementen ist möglich, obwohl die atomaren Events meist einfach strukturiert sein werden.

Ein Bereich im XML-Markup, der noch nicht bedachtet wurde, sind die Textinhalte. Wie folgendes Beispiel zeigt, sollten diese auch in die Muster mit aufgenommen werden können.

Beispiel 5.9 (Muster mit Textinhalt)

<travel:cancel−flight>
    <travel:code>G7654−12</travel:code>
</travel:cancel−flight>

Hier ist der Code des Fluges als Text innerhalb des Elementes <travel:code> angegeben. Damit wird ausgedrückt, dass nur die „cancel-flight“-Events berücksichtigt werden sollen, die in diesem Text übereinstimmen. Zwei Texte, deren Position innerhalb des Musters bzw. Events durch das Muster vorgegeben ist, werden also auf Gleichheit überprüft. Dabei werden Leerräume am Anfang und Ende des Textes nicht berücksichtigt, da sie oftmals nur durch die Formatierung zustande kommen. Daraus folgt, dass Umbrüche und Einrückungen ohne weiteren Inhalt nicht relevant sind.

Bei reinen Textinhalten ist die Semantik noch klar. Aber was bedeutet es, wenn gemischte Inhalte auftreten? Soll der gesamte Textinhalt des Elements, also inklusive der Texte in den Kindern übereinstimmen oder soll für jeden Textknoten im Muster ein Gegenstück im Event gesucht werden? Beides ist nicht möglich, da im Event zusätzliche Elemente und Texte auftreten können, die

im Muster nicht enthalten sind. Dadurch ergeben sich verschiedene Probleme. Die Texte können im Event durch Elemente getrennt sein.

Beispiel 5.10 (Probleme beim Vergleich von Textknoten 1)
Das Muster

<uni:hire−professor>
Prof. Dr. Dipl.−Inf. Hans Heinrich Hagen
</uni:hire−professor>

und das Event

<uni:hire−professor>
Prof. Dr. Dipl.−Inf.<person:age birdthday="1972/05/21" />Hans Heinrich Hagen
</uni:hire−professor>

passen nicht mehr zusammen, da im Event zwei Textknoten vorhanden sind, die beide nicht mit dem einen aus dem Muster übereinstimmen.

Auch können einzelne Textpassagen durch zusätzliche Auszeichnungen in Kindelemente wandern.

Beispiel 5.11 (Probleme beim Vergleich von Textknoten 2)
Das Muster aus Bsp. 5.10 und das Event

<uni:hire−professor>
<uni:title>Prof. Dr. Dipl.−Inf.</uni:title>Hans Heinrich Hagen
</uni:hire−professor>

passen nicht mehr zusammen, obwohl nur der Titel im Event explizit als solcher gekennzeichnet ist.

Möglich ist auch, dass im Event mehr Text vorhanden ist als im Muster verlangt wird. Aus diesen Gründen ist es nötig eine Einschränkung anzugeben, die sowohl die Muster als auch die Events betrifft, die von dem hier beschriebenen System verarbeitet werden sollen. In jedem Element mit gemischtem Inhalt darf nur jeweils ein Textknoten vorkommen!

Beispiel 5.12 (Muster mit gemischten Inhalten eines Elements)
<uni:hire−professor>
<uni:title>Prof. Dr. Dipl.−Inf.</uni:title>Hans Heinrich Hagen
</uni:hire−professor>

Durch diese Einschränkung müssen nur zwei Texte miteinander verglichen werden. Dabei muss der einzige Textknoten des Muster-Elements in dem des Event-Elements enthalten sein. Der Text im Event kann also durch Voranstellen oder Anhängen weiterer Passagen erweitert werden.

Die weiteren Typen von Knoten, nämlich Kommentare und Steueranweisungen, haben keine Wirkung. Sie können zwar sowohl in den Events wie auch in den Mustern vorkommen, werden aber nicht weiter berücksichtigt.

5.3.2 Die Variablen

In diesem Abschnitt wird betrachtet, wie Variablen bei der Spezifikation von kombinierten Events angegeben werden können. Durch Variablen werden die vom Benutzer gewünschten Bestandteile der im Eventausdruck enthaltenen atomaren Events an Namen gebunden.
Mittels Variablen lassen sich verschiedene Funktionen realisieren. Einerseits ermöglichen sie dem Benutzer der Event-Detection-Unit einen direkten Zugriff auf Werte. Andererseits lässt sich mit Variablen eine Art Join-Bedingung zwischen verschiedenen atomaren Events in einem Eventausdruck realisieren. Zusammen mit der Möglichkeit Variablen vorzubelegen, ergibt sich eine effektive Methode Eventausdrücke in verschiedenen Kontexten zu verwenden.

Den Begriff „Join-Bedingung“ verwendet man üblicherweise im Kontext von relationalen Datenbanken. Er bezeichnet eine Bedingung, die einschränkt, welche Datensätze einer Datenbank kombiniert werden sollen. Ähnlich verhält es sich bei Eventausdrücken. Wenn in zwei atomaren Events eines Ausdrucks die gleiche Variable vorkommt, legt das zuerst eintrettende Event den Wert der Variablen fest. Das zweite Event muss dann an entsprechender Stelle ebenfalls diesen Wert aufweisen.

Beispiel 5.13 (Join-Variable)
Damit eine Fluggesellschaft ihre Kunden benachrichtigen kann, wenn deren Flüge gestrichen werden, benötigen sie ein Event, das ausgelöst wird, wenn nach einer Buchung eine Streichung für den selben Flug erfolgt:
<travel:booking flightcode="$flight">;<travel:cancel–flight code="$flight">
Sollte ein Booking-Event eintreten, wird die Variable ‚flight‘ gebunden. Ein Cancel-Flight-Event löst das zusammengesetzte Event jetzt nur noch aus, wenn es sich auf den entsprechenden Flug bezieht.

Wie in Kapitel 4 dargestellt wurde, gibt es zwei Möglichkeiten Variablen zu spezifizieren. Bei der ersten wird die Variable an den Wert eines Parameters gebunden.

Beispiel 5.14 (Muster mit Variablenbindung eines Attributwertes)
<travel:cancel–flight code="$flight"/>

Da es sich bei den Werten einer solchen Variablen immer um Strings handelt, ist ihre Behandlung relativ einfach. Sollte der Algorithmus beim Vergleich eines atomaren Events mit einem Muster auf eine solche Variable stoßen, muss er prüfen, ob die Variable schon gebunden ist, also bereits

einen Wert hat. In diesem Fall muss der Wert der Variablen mit dem Wert des korrespondierenden Attributs im Event übereinstimmen, damit das Event verwendet werden kann. Anderenfalls sollte die Variable gebunden werden. Wenn sonst nichts dagegen spricht, kann das Event nach der Bindung verwendet werden, da bisher keine Bedingungen an den Wert des Attributes gestellt sind.

Die zweite Möglichkeit eine Variable anzugeben, ist ein Fragment des Event-Musters in ein <eca:variable>-Element einzufassen. Dadurch wird das entsprechende Fragment eines atomaren Events an die Variable gebunden.

Beispiel 5.15 (Muster mit Variablenbindung an ein XML-Fragment)

```
<travel:cancel-flight>
    <eca:variable name="flight">
        <travel:code/>
    </eca:variable>
</travel:cancel-flight>
```

Das Vorgehen beim Auftreten einer solchen Variablen ist ähnlich wie im Fall einer Attribut-Variablen. Sollte die Variable bereits gebunden sein, wird ihr Wert mit dem im entsprechenden XML-Fragment des eingetretenen, atomaren Events verglichen. Sollte die Variable noch keinen Wert haben, wird sie gebunden. Allerdings kann der Wert einer solchen Variablen mehr sein als reiner Text. Es muss also ein Vergleich auf Identität zweier XML-Fragmente möglich sein, um solche Variablen korrekt zu behandeln.

Durch die Kombination beider Formen der Variablen stellt sich eine Frage: Wie wird mit einer attribut-basierten Variablen umgegangen, wenn sie bereits an ein XML-Fragment gebunden ist? Der Vergleich eines Attributwertes mit einem XML-Element führt natürlich zu einem negativen Ergebnis. Daher wird in einem solchen Fall der Text-Inhalt des Fragments statt dem gesamten Markup für den Vergleich herangezogen.

Kapitel 6

Implementierung in XSLT

Dieses Kapitel befasst sich mit der praktischen Umsetzung des in Kapitel 5 vorgestellten Konzepts zum Erkennen von Snoop-Events. Zu den Anforderungen dieser Diplomarbeit gehört es, die zu erarbeitende Lösung in XML-Techniken zu realisieren. Da bei Eingang eines Events die Automaten von einem Status in einen anderen übergehen, also das Markup transformiert wird, fiel die Wahl auf „XSL for Transformation"(XSLT). Diese Sprache, die vom W3C spezifiziert wurde, ist speziell für die Transformation von XML-Markup geschaffen worden.

6.1 Interne Darstellung

Wie in Abschnitt 5.1 dargestellt, besteht eine Situation aus Automaten, Instanzen und Events. Diese Komponenten werden im Verlauf des Algorithmus um zusätzliche Informationen erweitert. Um diese Informationen aufnehmen zu können, werden alle Instanzen und Events von einem zusätzlichen Element umhüllt. Bei Instanzen ist dies ein <instancewrapper>-Element, bei Events heißt das Element <eventwrapper>. Die zur Laufzeit auftretenden Informationen werden in Attributen abgelegt. Bei Events sind dies der Typ (typ) und die Angabe des Rekursionsschrittes, in dem das Event eingetreten ist (stepcounter). Bei Instanzen sind zwei Attribute vorgesehen. Das Attribut „stepcounter" nimmt wiederum die Nummer des Rekursionsschritt auf, in dem die Instanz erzeugt wurde. Zusätzlich kann eine Instanz als stabil oder instabil gekennzeichnet werden. Die Wrapper werden beim Start des Algorithmus erzeugt und am Ende wieder entfernt.

Die Automaten werden entgegen der Definition nicht in die Situation aufgenommen. Da sie unverändert bleiben wird so verhindert, dass sie unnötig zwischen den Situationen kopiert werden müssen. Damit die Automaten immer zur Verfügung stehen, werden sie in einer XSLT-Variable abgelegt. In die endgültige Situation zum Abschluss des Algorithmus werden die Automaten natürlich wieder aufgenommen.

6.2 Organisation

Der Quellcode des Algorithmus verteilt sich auf vier Dateien: „ceda-io.xsl“, „ceda-algorithm.xsl“, „ceda-event.xsl“ und „ceda-state.xsl“. Diese Aufteilung wurde so vorgenommen, dass auch eine inhaltliche Trennung gegeben ist. In ceda-io.xsl wurden die Teile des Algorithmus aufgenommen, die die Eingabe- und Ausgabe-Schnittstellen sowie die Datenaufbereitung betreffen. Die Dateien ceda-event.xsl und ceda-instance.xsl beinhalten die Verarbeitung von Events und Automateninstanzen. Der Vergleich zwischen Eventmustern und atomaren Events aus Abschnitt 5.3 befindet sich in ceda-event.xsl. In ceda-algorithm.xsl wird die Transformation der Automaten von einem Zustand in einen anderen aufgrund von Events implementiert. In dieser Datei wird umgesetzt, was in Abschnitt 5.2 besprochen wurde. Im Folgenden werden die jeweiligen Inhalte der Dateien als Module bezeichnet. So ergeben sich die Module IO, Event, Instanz und Algorithmus. Diese Module werden in den vier anschließenden Abschnitten einzeln vorgestellt.

6.3 Modul IO

Dieses Modul bildet das Interface zum Benutzer. Es nimmt das eingetretene, atomare Event aus einer Datei entgegen. Mit den Automaten und diesem Event bildet es eine Ausgangssituation zur weiteren Verarbeitung. Basierend auf der erzeugten Situation wird dann die Transformation der Automaten durchgeführt. Am Ende bereitet das Modul das Ergebnis auf und gibt es danach aus.

Um seiner Ausgabefunktion gerecht zu werden, definiert das Stylesheet zu diesem Modul die XSLT-Ausgabe-Eigenschaften mittels <**xsl:output**>. Das zu erzeugende Dokument erhält eine DTD-Angabe und wird im Format XML 1.0 mit der Kodierung UTF-8 ausgegeben.

Desweiteren werden zwei globale Variablen initialisiert. In „EVENTFILE“ ist der Name der Datei abgelegt aus der das atomare Event gelesen wird. Die zweite Variable, „INPUTFILE“ enthält das gesamte Dokument. Dies wird nötig, da durch die vielfache Verwendung von Variablen und For-Each-Schleifen das Dokument nicht wie üblich mittels XPath-Ausdrücken adressiert werden kann. Durch die Variable wird die Verfügbarkeit des Dokuments sichergestellt.

Ein Template in diesem Modul wird durch die Wurzel des Dokuments aufgerufen. Es überprüft anhand des ersten Elements, ob das Eingangsdokument Eventautomaten enthält. Sollte dem nicht so sein, wird die weitere Verarbeitung mit einer Fehlermeldung abgebrochen.

Das folgende Listing enthält ein zweites Templates aus diesem Modul. Dieses Template wird von obigem aufgerufen und führt sämtliche weiteren Aufgaben des Moduls durch.

```
<xsl:template match= "/ceda:ceda">                                          45
  <!-- Erzeugt die Ausgangssituation mit dem Event und speichert sie in einer Variablen. -->   46
  <!-- Variable mit der Ausgangssituation. -->                              47
  <xsl:variable name= "CURRENT_SITUATION" >                                 48
```

```
<!-- Instanzen aus der Eingabe für die interne Darstellung aufbereiten. -->                    49
<xsl:apply-templates select= "./ceda:instance" /><!--Modul Instanz-->                          50
<!-- Das Primitv-Event aus der Übergabedatei in die aktuelle Situation kopieren. -->           51
<xsl:call-template name= "createPrimitiveEvent" ><!--Modul Event-->                           52
  <xsl:with-param name= "EVENT" select= "document($EVENTFILE)/child::*" />                     53
</xsl:call-template>                                                                          54
</xsl:variable><!-- End Variable CURRENT_SITUATION -->                                        55
<!-- Den Algorithmus zum Detektieren der Composite-Events anstossen. -->                      56
<!-- Variable mit dem Ergebnis des Algorithmus. - Der letzen Situation. -->                   57
<xsl:variable name= "NEW_SITUATION" >                                                         58
  <!-- Aufruf der rekursiven Verarbeitung des Events. -->                                     59
  <xsl:call-template name= "rekursivStep" ><!--Modul Algorithm-->                             60
    <xsl:with-param name= "OLD_SITUATION" select= "$CURRENT_SITUATION" />                     61
    <xsl:with-param name= "COUNTER" select= "'0'" />                                          62
  </xsl:call-template>                                                                        63
</xsl:variable>                                                                               64
<!-- Hier wird die Ausgabe erzeugt. -->                                                       65
<!-- Anlegen des ceda-Elements des Ausgabedokuments. -->                                      66
<ceda:ceda>                                                                                   67
  <!--Kopieren der urspruenglichen Attribute in das neue ceda-Element.-->                     68
  <xsl:copy-of select= "@*" />                                                                69
  <!--Kopieren der Automatenspezifikationen, da sie unveränderlich sind.-->                   70
  <xsl:copy-of select= "ceda:automaton" />                                                    71
  <!-- Kopieren der stabilen Statie aus der neuen Situation in die Ausgabe. -->               72
  <xsl:copy-of select= "$NEW_SITUATION/instancewrapper[./@stable = 'yes']/ceda:instance"      73
    />
  <!-- Kopiert die detektierten Events in die Ausgabe. -->                                    74
  <xsl:copy-of select= "$NEW_SITUATION/eventwrapper[@typ='composite']/ceda:instance" /       75
    >
</ceda:ceda>                                                                                  76
</xsl:template>                                                                               77
```

Listing 6.1: Template für das ceda-Element des Dokuments

Das Template ist dreigeteilt. Im ersten Teil wird die Variable „CURRENT_SITUATION“ erzeugt, deren Inhalt als Ausgangssituation betrachtet wird (Zeile 48 – 55). Sowohl um das atomare Event (Zeile 52) als auch um die einzelnen <instance>-Tags (Zeile 50) wird ein Element als Hülle gelegt, bevor sie in die Variable kopiert werden. Diese Hüllen werden in den beiden Templates erzeugt, deren Aufrufe innerhalb der Variable liegen. Diese Templates sind aus den Modulen Event bzw. Instanz.

Der zweite Teil des Templates erzeugt wiederum eine Variable (Zeilen 58 – 64). In „NEW_SITUATION“ wird am Ende die neue, vollständig bearbeitete Situation stehen. Diese Arbeit wird durch den Aufruf eines Templates aus dem Modul Algorithmus ausgeführt (Zeilen 60 ff). Das Ergebnis dieses Templates enthält allerdings nicht die Automaten, dafür aber noch alle Hüllen, mit denen die Events und Instanzen versehen sind. Daher wird es nicht direkt ausgegeben.

Der letzte Teil in diesem Template bereitet die Daten aus „NEW_SITUATION“ auf (67 – 76). Für das neue Dokument wird ein ceda-Element angelegt (Zeile 67). In dieses werden die Automaten kopiert (Zeile 71). Danach werden die Automateninstanzen aus ihren Hüllen befreit und ebenfalls mit in das ceda-Element aufgenommen (Zeile 73). Dabei werden die instabilen Instanzen ausgelassen, da sie nur für die interne Verarbeitung vorgehalten wurden. Abschließend werden die detektierten Composite Events mit in die Ausgabe aufgenommen (Zeile 75).

6.4 Modul Instanz

Die Automateninstanzen werden im Verlauf des Algorithmus oft manipuliert. Daher sind alle Templates, die solche Manipulationen vornehmen, in einem Modul zusammengefasst. Insgesamt vier Templates sind enthalten.

6.4.1 Erzeugen der internen Darstellung

Ein Template übernimmt die Aufbereitung der Instanzen aus dem Eingabedokument und wird zu Beginn durch das Modul IO aufgerufen. Der aktuelle Knoten in der XSLT-Verarbeitung entspricht dabei der aufzubereitenden Instanz.

```
<xsl:template name= "createAutomatoninstance" match="/ceda:ceda/ceda:instance">                9
  <xsl:element name= "instancewrapper" >                                                       10
    <xsl:attribute name= "stepcounter" >0</xsl:attribute>                                      11
    <xsl:attribute name= "new" >no</xsl:attribute>                                             12
    <xsl:attribute name= "stable" >yes</xsl:attribute>                                         13
    <!-- Die eigentliche Automateninstanz wird hier hinzugefügt. -->                          14
    <xsl:copy-of select= "." />                                                                15
  </xsl:element>                                                                               16
</xsl:template>                                                                                17
```

Listing 6.2: Template zur Kapselung der Instanzen

Die eigentliche Aufgabe des Templates ist die Konstruktion der Hülle (Zeile 10) und ihrer Attribute (Zeilen 11 – 13). Die Attribute „stepcounter“ und „new“ haben den Wert 0 bzw. ‚no‘. Das liegt daran, dass die umhüllten Automateninstanzen aus der Eingabe des Algorithmus stammen und damit nicht im aktuellen Programmaufruf entstanden sind. Darüber hinaus werden die Instanzen mittels „stable“ als stabil markiert, denn alle instabilen Instanzen wären nicht in die Eingangssituation aufgenommen worden, wie man in Zeile 73 von Listing 6.1 sehen kann.

6.4.2 Transformation durch ein Event

Jeweils ein Template dient dem Weiterschalten der Automateninstanzen durch ein atomares oder ein kombiniertes Event. Dabei wird der Zustand geändert, das auslösende Event in den Parameterstore aufgenommen und eventuell neu gebundene Variablen der Instanz hinzugefügt. Die Templates „createNewInstanceWithAtomicEvent“ und „createNewInstanceWithCompositeEvent“ sind sich dabei recht ähnlich, weswegen hier nur ersteres erläutert wird. Anschließend wird noch auf die geringen Unterschiede eingegangen. Beide Templates setzen voraus, dass das Event wirklich den Zustandsübergang auslöst und führen daher keine Prüfung dieser Bedingung durch. Bevor diese Templates angewandt werden, sollte mittels des Templates „equalityTest“ (siehe Seite 65) aus dem Modul Event überprüft werden, ob das gewünschte Event auf die Automateninstanz anwendbar ist.

Beide Templates erwarten eine Reihe von Parametern. „OLD_INSTANCE“ beinhaltet die Hülle der Instanz, die in einen neuen Zustand übergehen soll, „TARGET“ die ID des Zustands in den gewechselt werden soll, „EVENT“ die Hülle des Events, das den Übergang ausgelöst hat und „STEP“ den internen Zähler der angibt, wie oft der Algorithmus sich bereits selbst aufgerufen hat. Der Parameter „NEW_VARIABLES“ wird nur von „createNewInstanceWithAtomicEvent“ benötigt. Er übergibt die Variablenbindungen, die durch das eingetretene Event hinzugekommen sind. Aus diesen Informationen wird eine Hülle mit der geänderten Automateninstanz erstellt, die die alte ersetzt.

```
<xsl:template name= "createNewInstanceWithAtomicEvent" >                                    26
  <xsl:param name= "OLD_INSTANCE" />                                                        27
  <xsl:param name= "EVENT" />                                                               28
  <xsl:param name= "TARGET" />                                                              29
  <xsl:param name= "STEP" />                                                                30
  <xsl:param name= "NEW_VARIABLES" />                                                       31
                                                                                            32
  <!-- Wrapper anlegen und mit dem neuen Kontext des Automaten befüllen.-->                 33
  <xsl:element name= "instancewrapper">                                                     34
    <xsl:attribute name= "stepcounter"><xsl:value-of select= "$STEP" /></xsl:attribute>     35
    <xsl:attribute name= "new">yes</xsl:attribute>                                          36
    <xsl:attribute name= "stable">yes</xsl:attribute>                                       37
    <!-- Element 'instance' anlegen und mit den geänderten Daten bestücken. -->            38
    <ceda:instance>                                                                         39
      <!--Die Referenz auf den Automaten, dessen Kontext durch den vorliegenden Status     40
          ausgetrückt wird, kopieren.-->
      <xsl:attribute name= "automaton">                                                     41
        <xsl:value-of select= "$OLD_INSTANCE/ceda:instance/@automaton" />                   42
      </xsl:attribute>                                                                      43
      <!--Das Element mit der Referenz auf den neuen Zustand anlegen.-->                    44
      <ceda:current-state>                                                                  45
        <!--Das Attribut mit der Referenz auf den neuen Zustand anlegen.-->                 46
```

```
<xsl:attribute name="ref">                                                    47
  <!--Bestimmung des Wertes der Referenz auf den neuen Zustand.-->             48
  <xsl:value-of select= "$TARGET" />                                          49
</xsl:attribute>                                                              50
</ceda:current-state>                                                         51
<!-- Den Parameterstore aktualisieren. -->                                    52
<ceda:parameterstore>                                                         53
  <xsl:copy-of select= "$OLD_INSTANCE/ceda:instance/ceda:parameterstore/*" />  54
  <!--Das Event dem Parameterstore hinzufügen. -->                            55
  <xsl:copy-of select= "$EVENT/child::*" />                                   56
</ceda:parameterstore>                                                        57
<!-- Die Variablen dem Variables-Tags hinzufügen. -->                         58
<ceda:variables>                                                              59
  <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$OLD_INSTANCE/ 60
      ceda:instance/ceda:variables/eca:variable" />
  <xsl:copy-of select= "$NEW_VARIABLES" />                                    61
</ceda:variables>                                                             62
</ceda:instance>                                                              63
</xsl:element>                                                                64
</xsl:template>                                                               65
```

Listing 6.3: Template zum Erzeugen von Instanzen mit geänderten Zuständen

Zu Beginn wird die Hülle (Zeile 34) mit ihren Attributen (Zeilen 35 – 37) erstellt. Das Attribut „stepcounter“ wird auf den Wert des Parameters „STEP“ gesetzt. Damit wird festgelegt, in welchem Rekursionsschritt die Instanz angelegt wurde. Der Wert des Attributs „stable“ ist ‚yes‘, um auszudrücken dass Instanzen, die aus Zustandsübergängen entstehen, immer stabil sind. Innerhalb der Hülle wird die eigentliche Automateninstanz angelegt (Zeile 39 – 63). Dazu wird der aktuelle Zustand der Instanz auf den neuen Wert aus dem Parameter „TARGET“ gesetzt (Zeile 47 ff) und das Event von „EVENT“ in den Parameterstore kopiert (Zeile 56). Die Elemente des Parameterstore der zu transformierenden Instanz werden ebenfalls in den neuen Speicher mit aufgenommen (Zeile 54). In der Behandlung der Variablenbindungen unterscheiden sich die beiden Templates. „createNewInstanceWithAtomicEvent“ fügt den bislang vorhandenen Einträgen jene aus dem Parameter „NEW_VARIABLES“ hinzu. In „createNewInstanceWithCompositeEvent“ hingegen werden die neuen Variablenbindungen aus dem kombinierten Event extrahiert.

```
<ceda:variables>                                                              109
  <!--Die alten Variablen übernehmen.-->                                      110
  <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$OLD_INSTANCE/ 111
      ceda:instance/ceda:variables/eca:variable" />
  <!--Die Variablen aus dem hinzugefügten, Composite Event hinzufügen sofern sie nicht 112
      schon vorhanden sind.-->
  <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$EVENT/ceda:instance 113
      /ceda:variables/eca:variable[not(./@name = $OLD_INSTANCE/ceda:instance/
      ceda:variables/eca:variable/@name)]" />
```

```
</ceda:variables>                                                    114
```

Listing 6.4: Bestimmen der neuen Variablenbindungen

Um zu vermeiden, dass mehrere Einträge mit gleichen Namen und Werten entstehen, werden Variablen, die in der ursprünglichen Situation bereits vorhanden waren, durch den XPath-Ausdruck in Zeile 113 nicht übertragen.

6.4.3 Erzeugen instabiler Kopien

Aus bereits dargelegten Gründen muss die ursprüngliche Automateninstanz nach einer Transformation erhalten bleiben. In bestimmten Fällen muss sie allerdings speziell gekennzeichnet werden. Diese Kennzeichnung führt das folgende Template durch.

```
<xsl:template name= "createInstableCopy">                                    122
  <xsl:param name= "OLD_INSTANCE" />                                         123
                                                                             124
  <xsl:element name= "instancewrapper">                                      125
    <xsl:attribute name= "stepcounter"><xsl:value-of select= "$OLD_INSTANCE/ 126
        @stepcounter" /></xsl:attribute>
    <xsl:attribute name= "new"><xsl:value-of select= "$OLD_INSTANCE/@new" /></ 127
        xsl:attribute>
    <!-- Das Attribute stable auf no setzen. Damit wird angezeigt, dass diese Automateninstanz 128
        nicht in die Ausgabe übergeht. -->
    <xsl:attribute name= "stable">no</xsl:attribute>                         129
    <!-- Die Instanz in den geänderten Wrapper kopieren. -->                 130
    <xsl:copy-of select= "$OLD_INSTANCE/child::ceda:instance" />             131
  </xsl:element>                                                             132
</xsl:template>                                                              133
```

Listing 6.5: Template zum Erzeugen instabiler Kopien

Das Template nimmt im Parameter „OLD_INSTANCE“ die Hülle einer Automateninstanz entgegen. Deren Attribut „stable“ wird auf ‚no‘ gesetzt, um die Instabilität auszudrücken (Zeile 129). Die weiteren Attribute und die eigentliche Instanz werden unverändert übernommen.

6.5 Modul Event

Dieses Modul fasst alle Dinge zusammen, die das Programm mit atomaren und kombinierten Events tun können muss. Im einzelnen sind dies:

- Erzeugen der internen Darstellung eines atomaren Events
- Erzeugen der internen Darstellung eines zusammengesetzten Events
- Vergleich zwischen Event-Mustern und atomaren Events

- Abgleich von Variablenwerten

Die ersten beiden dieser Aufgaben führen die gemäß Abschnitt 6.1 benötigte Darstellung von Events herbei. Sie ähneln daher den Templates aus Abschnitt 6.4.1. Die beiden anderen Punkte ermöglichen es eine Aussage darüber zu treffen, ob ein Event als Auslöser für einen Zustandsübergang geeignet ist.

6.5.1 Erzeugen der internen Darstellung

Der erste Punkt aus obiger Aufzählung findet beim Start des Stylesheets Verwendung. Beim Erzeugen der Ausgangssituation für den Algorithmus wird das atomare Event, mit dem der Algorithmus angestoßen wurde, in die interne Darstellung überführt. Das Template „createPrimitiveEvent“, das diese Aufgabe ausführt, erwartet einen Parameter mit dem Namen ‚EVENT‘. Dessen Wert kann ein beliebiges XML-Fragment sein. Das Ergebnis eines Aufrufs ist ein<eventwrapper>-Element mit dem atomaren Event aus dem Parameter als Inhalt.

```
<xsl:template name= "createPrimitiveEvent" >                                    10
  <xsl:param name= "EVENT" />                                                   11
                                                                                12
  <!-- Anlegen des Umschlags für das eigentliche Event. -->                     13
  <xsl:element name= "eventwrapper">                                            14
    <xsl:attribute name= "typ" >primitiv</xsl:attribute>                        15
    <xsl:attribute name= "stepcounter">0</xsl:attribute>                        16
    <!-- Kopieren des Events in den Umschlag. -->                               17
    <xsl:copy-of select= "$EVENT" />                                            18
  </xsl:element>                                                                19
</xsl:template>                                                                 20
```

Listing 6.6: Template zum Erzeugen eines atomaren Events

Wie man sieht, ist das Template recht einfach. Das im Parameter übergebene Event wird in einen <eventwrapper>-Tag verpackt (Zeile 14) und mit zusätzlichen Informationen versehen. Diese Informationen umfassen die Kennzeichnung als atomares Event (Zeile 15) und den Rekursionsschritt (stepcounter), in dem das Event entstanden ist (Zeile 16). Der letztere Wert ist für atomare Events immer 0, da diese außerhalb des Algorithmus entstehen und zu Beginn der Ausführung bereits vorliegen. Das eigentliche Event wird in dieser Hülle unverändert abgelegt.

Die interne Darstellung der kombinierten Events wird auf ähnliche Weise erzeugt, wie das auch bei den atomaren Events der Fall ist. Die benötigten Daten werden mittels Parametern an das Template „createCompositeEvent“ übergeben. Daraus wird das Event erzeugt und mit einer Hülle versehen. Der Parameter „INSTANCE“ enthält die Automateninstanz, die sich in einem Zustand befindet, der ein Event auslöst. Ein zweiter Parameter „STEP“ enthält die Nummer des Rekusionsschrittes, in dem dieser Zustand erreicht wurde. Als Ergebnis des Templates wird

wiederum ein <eventwrapper>-Element ausgegeben. Diese Hülle enthält das <instance>-Element, welches das Event repräsentiert.

```
<xsl:template name= "createCompositeEvent" >                                    28
  <xsl:param name= "INSTANCE" />                                                29
  <xsl:param name= "STEP" />                                                    30
  <!-- Anlegen des Umschlags für das eigentliche Event. -->                     31
  <xsl:element name= "eventwrapper" >                                           32
    <xsl:attribute name= "typ" >composite</xsl:attribute>                       33
    <xsl:attribute name= "stepcounter" >                                        34
      <xsl:value-of select= "$STEP" />                                          35
    </xsl:attribute>                                                            36
    <xsl:copy-of select= "$INSTANCE" />                                         37
  </xsl:element>                                                                38
</xsl:template>                                                                 39
```

Listing 6.7: Template zum Erzeugen eines zusammengesetzten Events

Auch hier kann man wieder leicht sehen, welche Struktur durch das Template erzeugt wird. Der Umschlag aus einem <eventwrapper>-Tag (Zeile 32) beinhaltet das mittels „INSTANCE“ übergebenen Event. Da die Automateninstanz aus „INSTANCE“ alle Informationen enthält, die das zusammengesetzt Event auszeichnen, wird sie als Event weiterverwendet. Die Hülle um die Instanz ermöglicht die Unterscheidung von ‚normalen‘ Instanzen. Darüber hinaus existiert in der Hülle ein Attribut, dass das Event als eines vom Typ ‚composite‘, also ein zusammengesetztes Event, beschreibt (Zeile 33) und ein Attribut „stepcounter“, das angibt, wann zur Laufzeit des Algorithmus das Event aufgetreten ist (Zeile 34). Der Wert für „stepcounter“ wird dem Parameter „STEP“ entnommen.

6.5.2 Vergleich zwischen atomaren Events und Eventmustern

Der Vergleich eines Musters mit einem atomaren Event, dessen Semantik und algorithmische Umsetzung wurden bereits bei der allgemeinen Beschreibung des Algorithmus auf Seite 50 dargestellt. An dieser Stelle soll nun die genaue Implementierung in XSLT besprochen werden. Diese verteilt sich auf mehrere Templates. Jeweils ein Template existiert für die Verarbeitung der einzelnen Bestandteile eines Musters. Elemente, Attribute, Textknoten und Variablen werden also getrennt behandelt. Eine Sonderrolle nimmt ein weiteres Template ein. „equalityTest“ dient als Einstiegspunkt in den Vergleich. Dieses Template wird im Folgenden vorgestellt. Die anderen Templates folgen im Anschluss.

Alle Templates, die im Rahmen des Vergleichs angewandt werden, sind im Modus „equalityTest“ deklariert. Eine Ausnahme ist das benannte Template gleichen Namens, da solche Templates keinen Modus haben dürfen.

Bei der gewählten Implementierung sind einige Einschränkungen vorhanden. Variablen, die während dem Vergleich gebunden werden, stehen im weiteren Verlauf des Vergleichs noch nicht zur Verfügung. Dadurch darf innerhalb eines Eventmusters jede Variable nur einmal verwendet werden. Das heißt, innerhalb eines Musters dürfen zwar mehrere Variablen vorkommen, aber keine gleichen Namens. Sollte dies doch der Fall sein, würden mehrere Variablenbindungen zu einem Namen erzeugt. Sofern die entsprechenden Werte im Event verschiedenen sind, würde auch die Variable an mehrere Werte gebunden werden. Eine weitere Einschränkung besteht darin, dass die mehrfache Verwendung gleicher Elemente im Muster nicht dazu führt, dass im Event eine entsprechende Anzahl von Elementen vorkommt. Für jedes Element im Muster wird unabhängig von den anderen eine Übereinstimmung im Event gesucht. Daher wird die Übereinstimmung für gleiche Muster-Elemente immer an der ersten Stelle festgestellt, an der das Muster zum ersten mal im Event auftritt.

Beispiel 6.1 (Beispielhafter Vergleich)

Um die im nächsten Abschnitt besprochene Implementierung des Mustervergleichs besser verfolgen zu können, wird das Vorgehen zuerst anhand eines Beispiels vorgestellt. Dabei werden wissentlich auch einige Fehler in das Beispiel aufgenommen.

Das Muster für dieses Beispiel enthält sowohl Variablen in Attributen als auch solche, die durch <eca:variable>-Elemente deklariert werden. Auch alle relevanten Arten von XML-Knoten sind vorhanden:

```
<travel:booking xmlns:travel="http://travel-organisation.org">
    <travel:customer customerclass="Primary-Customer" customernr="$CUSTOMER" />
    <eca:variable name="TRAVEL">
        <travel:flight>
        <travel:hotel>
        <travel:flight>
    </eca:variable>
</travel:booking>
```

Das Event das auf dieses Muster geprüft werden soll ist das folgende:

```
<travel:booking xmlns:travel="http://travel-organisation.org">
    <travel:customer customernr="JohnDoo" customerclass="Primary-Customer" />
    <travel:flight flightnr="A-H3456">
    <travel:hotel>
        <travel:hotel-name>Paradise Hotel</travel:hotel-name>
    <travel:flight flightnr="C-K7654">
    </eca:variable>
</travel:booking>
```

Der Vergleich beginnt beim „booking“-Element des Musters. Es wird überprüft, ob das Wurzelelement im Event den gleichen Wert und den gleichen Namespace aufweist. Dies ist hier der Fall. Darum wird sowohl für das Element ‚customer‘ als auch für ‚variable‘ nach einer Übereinstimmung

im Event gesucht. Dazu werden alle Elemente aus der Wurzel des Events herangezogen (customer, flight, hotel und nochmals flight). Für ‚customer' wird aus dieser Menge jedes Element ausgewählt, das in Name und Namespace passt. Hier ist es nur eines. Für ‚customer' aus dem Muster und aus dem Event kann jetzt überprüft werden, ob die benötigten Attribute alle vorhanden sind. Die Attribute aus dem Muster werden der Reihe nach abgearbeitet. Begonnen wird mit ‚customerclass'. Es wird in allen Attributen des aktuellen Elements nach einer Übereinstimmung gesucht. Das erste Attribute im Event ‚customernr' kommt nicht in Frage, da sich der Name beider Attribute unterscheidet. Das zweite allerdings stimmt in Name, Namespace und Wert mit dem Muster überein. Das Attribut ‚customernr' des Musters enthält eine Variable. Das heißt, bei der Suche wird auf die Übereinstimmung im Wert verzichtet. Stattdessen wird eine Variablenbindung erzeugt sobald ein passendes Attribut gefunden wurde. In diesem Beispiel würde die Variable CUSTOMER den Wert ‚JohnDoo' erhalten. Das zweite Kindelement aus dem Muster ist ein Spezialfall. Anstatt <eca:variable> mit den Elementen des Musters zu vergleichen, geschieht dies für ihren Inhalt. Das erste <travel:flight> hat sowohl eine Übereinstimmung mit <travel:flight flightnr="A−H3456"> als auch <travel:flight flightnr="C−K7654">. Das gleiche gilt für das zweite Flight-Element des Musters. Auch <travel:hotel> hat ein Gegenstück zum Event. Da alle Elemente im Event vorhanden sind, wird die Variable TRAVEL an diese gebunden und der Vergleich mit einem positiven Ergebnis gebunden. Wenn man an dieser Stelle eines der beiden <flight>-Elemente aus dem Element entfernen würde, wäre das Ergebnis auch noch positiv. Das liegt daran, dass beide Elemente des Musters unabhängig voneinander mit dem verbliebenen Element des Events verglichen werden und daher ein positives Ergebnis liefern.

Template equalityTest

Über dieses Template erfolgt die Interaktion mit dem Rest des Algorithmus. Durch einen Aufruf des Templates wird der Vergleich zwischen Muster und atomaren Event angestoßen. Am Ende dieses Vergleichs wird das Ergebnis durch das Template aufbereitet und ausgegeben.

Die Eingabeparameter umfassen die beiden zu vergleichenden Werte sowie die Menge der zum Zeitpunkt des Aufrufs gebundenen Variablen. „EVENT" ist der Parameter, über den das Event in Form eines XML-Elements angegeben wird. Das Muster wird auf die gleiche Weise mittels „PATTERN" übergeben. Ein dritter Parameter nimmt die Variablen auf. In „VARIABLES" wird die Menge der gebundenen Variablen der im Algorithmus aktuellen Automateninstanz übergeben. Der Parameter enthält also keinen Inhalt oder eine Reihe von <eca:variable>-Tags. Die Ausgabe des Templates umfasst zwei Dinge: erstens einen Boolean-Wert (‚TRUE' oder ‚FALSE') der angibt, ob der Vergleich erfolgreich war oder nicht; Und zweitens eine Reihe von Variablenbindungen in Form von <eca:variable>-Tags. Diese Angaben fehlen natürlich, wenn keine Variable gebunden wurde oder keine Übereinstimmung zwischen Event und Muster vorliegt.

```
<xsl:template name= "equalityTest" >                                                    47
  <xsl:param name= "EVENT" />                                                           48
  <xsl:param name= "PATTERN" />                                                         49
  <xsl:param name= "VARIABLES" />                                                       50
                                                                                        51
  <!-- Variable, die für jede Übereinstimmung die nicht erreicht wird, ein Element enthält. -->  52
  <xsl:variable name= "NOMATCHES" >                                                     53
    <!-- Für die Wurzel, ihre Attribute und Kinder Übereinstimmung des Musters mit dem Event  54
         prüfen. -->
    <xsl:apply-templates select= "$PATTERN" mode= "equalityTest">                       55
      <xsl:with-param name= "CHILDREN" select= "$EVENT" />                              56
      <xsl:with-param name= "VARIABLES" select= "$VARIABLES" />                         57
    </xsl:apply-templates>                                                              58
  </xsl:variable>                                                                       59
                                                                                        60
  <!-- Fallunterscheidung, ob ein Übereinstimmung zwischen Pattern und Event vorliegt. Diese wird  61
       aufgrund der Ergebnisse des Tests der einzelnen Elemente, die in der Variablen NOMATCHES
       gespeichert sind, gefällt. -->
  <xsl:choose>                                                                          62
  <xsl:when test= "count($NOMATCHES/nomatch) = 0" >                                     63
    <xsl:text>TRUE</xsl:text>                                                           64
    <xsl:copy-of select= "$NOMATCHES/eca:variable" />                                   65
  </xsl:when>                                                                           66
  <xsl:otherwise>                                                                       67
    <xsl:text>FALSE</xsl:text>                                                          68
  </xsl:otherwise>                                                                      69
  </xsl:choose>                                                                         70
</xsl:template>                                                                         71
```

Listing 6.8: Template für den Vergleich eines Events mit einem Muster

In den Zeilen 53 bis 59 des oben stehenden Listings wird die eigentliche Arbeit an ein anderes Template weitergereicht und dessen Ergebnis in einer Variablen gespeichert. Das aufgerufene Template ist eines derjenigen, die den Vergleich von Elementen, Attributen, Textknoten oder Variablen durchführen. Die Auswahl hängt davon ab, von welcher Art das Muster ist. Die Variable CHILDREN enthält dabei das Event. Der Knoten, mit dem das Element aufgerufen wird, ist das Muster. Weiter unten in den Zeilen 62 bis 70 wird die Variable ausgewertet und das Ergebnis des Templates erzeugt. Dazu wird überprüft, ob die Variable <nomatch>-Elemente enthält. Ein solches Element würde anzeigen, dass ein Teil des Musters nicht mit dem Event übereinstimmt.

Die Templates, die hier aufgerufen werden, haben einige Gemeinsamkeiten. Sie haben ein einheitliches Ausgabeverhalten und eine ähnliche Struktur. Die Gründe für die gewählte Modellierung des Ausgabeverhaltenes werden anhand des folgenden Templates zur Behandlung von Elementen erläutert. Die Implementierung, die darüber hinausgeht, wird hingegen jeweils einzeln besprochen.

Template zum Element-Test

Dieses Template wird für jedes XML-Element innerhalb des Musters aufgerufen. Seine Aufgabe ist es innerhalb einer Menge von Elementen eines zu finden, das auf das Muster-Element passt. Als Muster wird der aktuelle Knoten zum Zeitpunkt des Aufrufs verwendet. Für die weiteren benötigten Daten existieren Parameter. In „CHILDREN“ werden die Vergleichselemente übergeben. Das heißt, eine Menge beliebiger Elemente ist hier zulässig. In dem zweiten Parameter „VARIABLES“ werden dem Template die bereits gebundenen Variablen mitgegeben. Das Ergebnis dieses Templates ist ein <nomatch>-Element, falls keine Übereinstimmung zu dem Muster gefunden wurde. Anderenfalls können einige Variablenbindungen mittels <eca:variable> ausgegeben werden. Zu diesem Template gibt es noch einen Spezialfall, der sich mit der Behandlung von Variablendeklarationen innerhalb des Musters befasst (siehe Seite 74).

```
<xsl:template match="child::*" mode="equalityTest">                                    79
  <xsl:param name="CHILDREN" />                                                        80
  <xsl:param name="VARIABLES" />                                                       81
  <xsl:variable name="CURRENT_PATTERN" select="." />                                   82
                                                                                       83
  <xsl:variable name="MATCHES">                                                        84
    <xsl:for-each select="$CHILDREN">                                                  85
      <xsl:if test="(local-name(.) = local-name($CURRENT_PATTERN)) and (               86
          namespace-uri(.) = namespace-uri($CURRENT_PATTERN))">
        <!-- Variable, die für jede Übereinstimmung die nicht erreicht wird, ein Element enthält.  87
            -->
        <xsl:variable name="NOMATCHES">                                                88
          <!-- Attribute testen: Für jedes Attribut im Muster eine Übereinstimmung im Event      89
              suchen. -->
          <xsl:apply-templates select="$CURRENT_PATTERN/@*" mode="equalityTest"        90
              >
            <xsl:with-param name="ATTRIBUTES" select="./@*" />                         91
            <xsl:with-param name="VARIABLES" select="$VARIABLES" />                     92
          </xsl:apply-templates>                                                       93
          <!-- Kinder testen: Für jedes Kind im Muster eine Übereinstimmung im Event suchen.  94
              -->
          <xsl:apply-templates select="$CURRENT_PATTERN/child::*" mode="               95
              equalityTest">
            <xsl:with-param name="CHILDREN" select="./child::*" />                     96
            <xsl:with-param name="VARIABLES" select="$VARIABLES" />                    97
          </xsl:apply-templates>                                                       98
          <!-- Texte testen: Für jeden Textknoten im Muster eine Übereinstimmung im Event       99
              suchen. -->
          <xsl:apply-templates select="$CURRENT_PATTERN/text()[1]" mode="              100
              equalityTest">
            <xsl:with-param name="CHILDREN" select="./text()" />                       101
          </xsl:apply-templates>                                                       102
        </xsl:variable>                                                                103
```

```
            <xsl:if test= "count($NOMATCHES/nomatch) = 0" >                                  104
              <match><xsl:copy-of select= "$CURRENT_PATTERN" /></match>                      105
            </xsl:if>                                                                        106
            <xsl:copy-of select= "$NOMATCHES/eca:variable" />                                107
          </xsl:if>                                                                          108
        </xsl:for-each>                                                                      109
      </xsl:variable>                                                                        110
      <xsl:if test= "count($MATCHES/match) = 0" >                                            111
        <nomatch>                                                                            112
          <xsl:copy-of select= "$CURRENT_PATTERN" />                                         113
        </nomatch>                                                                           114
      </xsl:if>                                                                              115
      <xsl:copy-of select= "$MATCHES/eca:variable" />                                        116
</xsl:template>                                                                              117
```

Listing 6.9: Template für den Vergleich eines Elements mit einem Muster

Der Kern des Templates liegt innerhalb der Variable „MATCH“ (Zeile 84). Um eine Übereinstimmung zu finden, wird für jedes Element aus „CHILDREN“ ein Vergleich durchgeführt (Zeile 85). Zu Beginn eines solchen Vergleichs wird eine Übereinstimmung bezüglich Namespace-URI und Name zwischen Muster-Element und Vergleichs-Element gesucht (Zeile 86). Sollte diese Übereinstimmung vorliegen, werden alle Attribute, Kindelemente und Textknoten des Musters geprüft. Dazu wird für jeden Knoten-Typ ein anderes Template verwendet (Zeilen 90, 95 & 100). Kindelemente werden durch einen rekursiven Aufruf diese Templates behandelt, Attribute und Texte mittels ähnlichen Templates. Das Ergebnis dieser Tests wird in die Variable „NOMATCHES“ gespeichert. Sollte diese Variable am Ende der Prüfungen leer sein, gibt es keine Knoten für die *keine* Übereinstimmung gefunden wurde. Das heißt, es ist insgesamt eine Übereinstimmung erreicht. Die Auswertung der letztgenannten Variable erfolgt in den Zeilen 104 bis 107. Um die vorhandene Übereinstimmung anzuzeigen, wird ein <match>-Element in die Variable MATCH geschrieben. Diese wird dann auf gleiche Weise ausgewertet (Zeilen 111 – 116). Sollte kein <match>-Element vorliegen wird als Ausgabe des Templates ein <nomatch>-Element erzeugt. Neue Variablenbindungen, die sich im Verlaufe der Tests ergeben, werden in den Zeilen 107 und 116 durch die XSLT-Variablen bis in die Ausgabe gereicht.

Der Grund für diese merkwürdige Schachtelung von Variablen und die wechselseitige Erzeugung von <match>- und <nomatch>-Elemente ist nicht offensichtlich. Daher wird hier näher darauf eingegangen: Die Templates aus dem Modus „equalityTest“ geben das Fehlen einer Übereinstimmung aus. Stattdessen könnten sie aber auch das Erreichen einer Übereinstimmung ausgeben. Wenn allerdings wie hier die Ergebnisse vieler einzelner Test erst gesammelt werden, müsste anschließend geprüft werden, ob wirklich zu jedem einzelnen Test ein positives Ergebnis vorliegt. Wenn hingegen die negativen Ergebnisse gesammelt werden, reicht das Vorkommen eines einzelnen solchen Ergebnisses um festzustellen, dass es keine Übereinstimmung gibt. Somit erklärt sich

die innere, die „NOMATCH“-Variable. Die äußere Variable ist nötig, da das Muster mit mehreren Elementen verglichen wird. Nur mit einem davon muss allerdings eine Übereinstimmung vorliegen. Daher werden die Ergebnisse jedes Vergleichs gesammelt und am Ende geprüft, ob es eine oder mehrere Übereinstimmungen gab. Sollte dies nicht der Fall sein, wird dieses negative Ergebnis ausgegeben.

Template zum Attribut-Test

Dieses Template testet einzelne Attribute eines Musters. Dabei ist das Vorgehen ähnlich wie im vorangegangenen Template. Es sucht in einer Liste mit Attributen eines, das zu dem Attribut des Musters passt, das überprüft werden soll. Diese Liste ist im Parameter „ATTRIBUTES“ enthalten. Sollte das zu prüfende Attribut mit einer Variable verknüpft sein, wird diese für den Vergleich herangezogen. Dazu werden sämtliche bisherigen Variablenbindungen mittels „VARIABLES“ dem Template mitgeteilt. Die Ausgabe ist, wie bei allen Templates des Modus „equalityTest“, ein <nomatch>-Element, wenn keine Übereinstimmung gefunden wurde und die neu gebundenen Variablen anderenfalls.

```
<xsl:template match= "@*" mode= "equalityTest" >                                    125
  <xsl:param name= "ATTRIBUTES" />                                                  126
  <xsl:param name= "VARIABLES" />                                                   127
  <xsl:variable name= "CURRENT_ATTRIBUTE" select= "." />                            128
                                                                                    129
  <!-- Variable zur aufnahme von Elementen die anzeigen das eine Übereinstimmung gefunden wurde  130
       . -->
  <xsl:variable name= "MATCHES" >                                                   131
    <!-- Fallunterscheidung ob bei diesem Attribute ein Parameter oder ein fixer Wert vorliegt. --  132
         >
    <xsl:choose>                                                                    133
    <!-- Fall 1: Es liegt eine Variable im Template vor. -->                       134
    <xsl:when test= "starts-with(string($CURRENT_ATTRIBUTE), '$')" >                 135
      <!-- Fallunterscheidung, ob der Parameter bereits einen Wert hat. -->         136
      <xsl:choose>                                                                  137
      <!-- Fall 1: Die Variable hat einen Wert. -->                                 138
      <xsl:when test= "$VARIABLES[string(./@name) = substring(string($              139
           CURRENT_ATTRIBUTE), 2)]" >
        <xsl:for-each select= "$ATTRIBUTES" >                                       140
          <xsl:if test= "( local-name(.) = local-name($CURRENT_ATTRIBUTE) ) and (   141
               namespace-uri(.) = namespace-uri($CURRENT_ATTRIBUTE) ) and ( string
               (.) = string($VARIABLES[string(./@name) = substring(string($
               CURRENT_ATTRIBUTE), 2)]) )" >
            <match><xsl:copy-of select= "." /></match>                              142
          </xsl:if>                                                                 143
        </xsl:for-each>                                                             144
      </xsl:when><!-- Ende Fall 1 der Unterscheidung, ob ein Wert gesetzt ist. -->  145
      <!-- Fall 2: Es liegt bisher kein Wert für die Variable vor. -->              146
```

```
      <xsl:otherwise>                                                                    147
        <xsl:for-each select= "$ATTRIBUTES" >                                            148
          <xsl:if test= "(local-name(.)=local-name($CURRENT_ATTRIBUTE))and((              149
              namespace-uri(.)=namespace-uri($CURRENT_ATTRIBUTE))" >
            <match><xsl:copy-of select= "." /></match>                                   150
            <!-- Erzeugen der Variablenbindung. -->                                      151
            <eca:variable>                                                               152
              <xsl:attribute name= "name">                                               153
                <xsl:value-of select= "substring(string($CURRENT_ATTRIBUTE),2)" />        154
              </xsl:attribute>                                                           155
              <xsl:value-of select= "string(.)" />                                       156
            </eca:variable>                                                              157
          </xsl:if>                                                                      158
        </xsl:for-each>                                                                  159
      </xsl:otherwise><!-- Ende Fall 2 der Unterscheidung, ob ein Wert gesetzt ist. -->  160
      </xsl:choose><!-- Ende Fallunterscheidung, ob ein Wert für den Parameter gesetzt ist. --  161
        >
    </xsl:when><!-- Ende Fall 1 der Unterscheidung, ob ein Parameter vorliegt. -->       162
      <!-- Fall 2: Es liegt kein Parameter vor. -->                                      163
    <xsl:otherwise>                                                                      164
      <xsl:for-each select= "$ATTRIBUTES" >                                              165
        <xsl:if test= "(local-name(.)=local-name($CURRENT_ATTRIBUTE))and((               166
            namespace-uri(.)=namespace-uri($CURRENT_ATTRIBUTE))and(.=$
            CURRENT_ATTRIBUTE)" >
          <match><xsl:copy-of select= "." /></match>                                     167
        </xsl:if>                                                                        168
      </xsl:for-each>                                                                    169
    </xsl:otherwise><!-- Ende Fall 2 der Unterscheidung, ob ein Parameter vorliegt oder nicht.  170
        -->
    </xsl:choose><!-- Ende der Fallunterscheidung ob ein Parameter vorliegt oder nicht. -->  171
    </xsl:variable><!-- Ende der Variable "MATCHES". -->                                 172
  <!-- Diese Abfrage überprüft, ob es keine Übereinstimmung gab und erzeugt mit diesem Wissen das  173
      Ergebnis. -->
  <xsl:if test= "count($MATCHES/match)=0" >                                              174
    <nomatch><xsl:copy-of select= "$CURRENT_ATTRIBUTE" /></nomatch>                      175
  </xsl:if>                                                                              176
  <xsl:copy-of select= "$MATCHES/eca:variable" />                                        177
</xsl:template>                                                                          178
```

Listing 6.10: Template für den Vergleich eines Attributes mit einem Muster

Generell existieren zwei Möglichkeiten für ein Attribut in einem Eventmuster. Entweder es ist ein gewöhnliches XML-Attribut oder es beinhaltet eine Variable. Diese beiden Möglichkeiten werden getrennt behandelt. Für die Auswahl sorgt eine Fallunterscheidung (Zeilen 135 und 164). Sollte eine Variable vorliegen, gibt es wiederum zwei Möglichkeiten. Entweder die Variable hat bereits einen Wert (Zeile 139) oder sie muss durch das Event an einen solchen gebunden werden (Zeile 147). In beiden Fällen wird im Event ein Attribut gesucht, das in Namespace und Name

mit dem des Musters übereinstimmt (Zeile 141 bzw. 149). Wenn die Variable gebunden ist, wird zusätzlich der Wert der passenden Variable mit dem des Attributes aus dem Event verglichen. Sollten diese Tests nicht erfolgreich sein, wird ein <nomatch>-Element ausgegeben. Im Falle, dass die Variable noch nicht gebunden ist, wird eine solche mit dem Wert des Attributes im Event erstellt und ausgegeben.

Wenn keine Variable im Attribut vorliegt, wird ein einfacher Vergleich durchgeführt. In „ATTRIBUTES“ wird ein Attribut gesucht, das mit dem Muster in Namespace, Name und Wert übereinstimmt (Zeile 166). Sollte ein solches Attribut nicht gefunden werden, wird ein <nomatch>-Element ausgegeben.

Beim Vergleich eines Variablenwertes mit einem Attribut wird die XPath-Funktion string() verwendet. Das führt dazu, dass beim Vergleich nicht das Markup des Variablenwertes verwendet wird, sondern der reine Textinhalt. Somit kann eine Variable in einem Attribut auch mit einem Wert verglichen werden, der ein XML-Fragment enthält.

Template zum Test von Textknoten

Das dritte Template aus dem Modus „equalityTest“ verarbeitet Textknoten. Dabei wird der Textknoten aus dem Muster, der das Template auslöst, mit dem ersten Textknoten des aktuellen Elements aus dem Event verglichen. Dieser Text wird über den Parameter „CHILDREN“ angegeben. Bei Identität dieser beiden Texte ist das Ergebnis positiv. Anderenfalls wird ein <nomatch>-Element erzeugt und ausgegeben.

```
<xsl:template match= "text()" mode= "equalityTest" >                                  184
  <xsl:param name= "CHILDREN" />                                                      185
                                                                                      186
  <xsl:if test= "not(normalize-space(.) = '')">                                       187
    <xsl:if test= "not(normalize-space(.) = normalize-space($CHILDREN))">             188
      <nomatch><xsl:value-of select="." /></nomatch>                                  189
    </xsl:if>                                                                         190
  </xsl:if>                                                                           191
</xsl:template>                                                                       192
```

Listing 6.11: Template für den Vergleich eines Events mit einem Muster

Statt die Texte auf Gleichheit zu prüfen, werden sie mit einer einfachen If-Abfrage auf Ungleichheit getestet (Zeile 188). Dadurch wird das übliche Verhalten mit Rückgabe einer Meldung, falls der Vergleich nicht zutrifft, umgesetzt. Vor dem Vergleich werden die Texte mittels der Funktion „normalize-space“ aufbereitet. Damit wird erreicht, dass Umbrüche und Einrückungen im Markup keine Auswirkungen auf den Vergleich haben. Textknoten, die erst durch solche Formatierungen zustande kommen, werden nicht zu einem Vergleich herangezogen (Zeile 187).

Template zum Variablen-Test

Eine spezielle Variante des Templates zum Vergleich von Elementen verarbeitet <eca:variable>-Tags. Wann immer ein solches Element im Muster auftritt wird überprüft, ob diese Variable bereits gebunden ist. Dazu werden alle Elemente aus dem Parameter „VARIABLES“ herangezogen. Danach wird nach einer Übereinstimmung mit einem Element aus „CHILDREN“ gesucht. Sollte keine Bindung vorliegen, wird sie erzeugt.

```
<xsl:template match= "eca:variable" mode= "equalityTest" xmlns:eca= "http://www.eca.org/eca-ml 199
    ">
  <xsl:param name= "CHILDREN" /> 200
  <xsl:param name= "VARIABLES" /> 201
  <xsl:variable name= "CURRENT_PATTERN" select= "." /> 202
 203
  <!--Wenn Bindungsmöglichkeiten existiren: die erste auswählen.--> 204
  <xsl:variable name="EVENTVALUE"> 205
    <xsl:variable name="BINDINGS"> 206
      <xsl:for-each select="$CHILDREN"> 207
        <!--Überprüfung von Kind aus dem Event und Kind des Variable-Tags.--> 208
        <xsl:variable name="TESTRESULT"> 209
          <xsl:apply-templates select="$CURRENT_PATTERN/child::*" mode=" 210
              equalityTest">
            <xsl:with-param name="CHILDREN" select="." /> 211
          </xsl:apply-templates> 212
        </xsl:variable> 213
        <xsl:if test="count($TESTRESULT/*) &gt; 0"> 214
          <xsl:copy-of select="." /> 215
        </xsl:if> 216
      </xsl:for-each> 217
    </xsl:variable> 218
    <xsl:if test="count($BINDINGS/*) &gt; 0"> 219
      <!--Den Wert, der im Event zu der Variablen passt ausgeben.--> 220
      <xsl:copy-of select="$BINDINGS/*[1]" /> 221
    </xsl:if> 222
  </xsl:variable> 223
 224
  <!-- Fallunterscheidung, ob die Variable schon gebunden ist oder nicht. --> 225
  <xsl:choose> 226
  <!-- Falls die Variable schon gebunden ist:--> 227
  <xsl:when test= "$VARIABLES[string(./@name) = string($CURRENT_PATTERN/@name)]" > 228
      <!--Variable mit dem Ergebnis des Vergleichs zwischen dem Variablenwert und seinem 229
          Gegenstück im Event.-->
      <xsl:variable name= "TESTRESULT"> 230
        <!--Vergleich des Variablenwertes mit dem Event.--> 231
        <xsl:apply-templates select="$VARIABLES[string(./@name) = string($ 232
            CURRENT_PATTERN/@name)]/node()" mode= "equalityTest" >
          <xsl:with-param name= "CHILDREN" select= "$EVENTVALUE/*" /> 233
        </xsl:apply-templates> 234
```

```
<!--Vergleich des Events mit dem Variablenwert.-->                                  235
<xsl:apply-templates select="$EVENTVALUE/*" mode= "equalityTest" >                  236
  <xsl:with-param name= "CHILDREN" select= "$VARIABLES[string(./@name) =            237
      string($CURRENT_PATTERN/@name)]/node()" />
</xsl:apply-templates>                                                              238
</xsl:variable>                                                                     239
<!--Auswertung der Variablen:-->                                                    240
<xsl:choose>                                                                        241
<!--Event und Variablenwert stimmen überein:-->                                     242
<xsl:when test="count($TESTRESULT/*) = 0">                                          243
  <match><xsl:copy-of select= "./child::*" /></match>                               244
</xsl:when>                                                                         245
<!--Anderenfalls:-->                                                                246
<xsl:otherwise>                                                                     247
  <nomatch><xsl:copy-of select= "./child::*" /></nomatch>                           248
</xsl:otherwise>                                                                    249
</xsl:choose>                                                                       250
</xsl:when>                                                                         251
<!--Die Variable ist bisher nicht gebunden.-->                                      252
<xsl:otherwise>                                                                     253
  <xsl:choose>                                                                      254
  <xsl:when test="count($EVENTVALUE/*) &gt; 0">                                     255
    <eca:variable>                                                                  256
      <xsl:copy-of select= "./@name" />                                             257
      <xsl:copy-of select="$EVENTVALUE/*" />                                        258
    </eca:variable>                                                                 259
    <match><xsl:copy-of select= "./child::*" /></match>                             260
  </xsl:when>                                                                       261
  <xsl:otherwise>                                                                   262
    <nomatch><xsl:copy-of select= "./child::*" /></nomatch>                         263
  </xsl:otherwise>                                                                  264
  </xsl:choose>                                                                     265
</xsl:otherwise>                                                                    266
</xsl:choose>                                                                       267
</xsl:template>                                                                     268
```

Listing 6.12: Template für den Vergleich eines Variablenwertes mit einem Event

Im ersten Block des Templates (Zeile 205 – 223) wird das Event nach dem Wert durchsucht, der auf die Variable passt. Dazu wird jedes Element aus „CHILDREN“ nach dem Muster aus dem Inneren des <eca:variable>-Elements durchsucht (Zeilen 207 – 217). Die erste solche Fundstelle wird dann in der XSLT-Variablen „EVENTVALUE“ abgelegt (Zeile 221).

Der zweite Block des Templates beginnt mit einer Fallunterscheidung. Unterschieden wird, ob die behandelte Variable bereits gebunden ist (Zeile 228) oder nicht (Zeile 253). Wenn die Variable gebunden ist, wird ein Vergleich des Wertes aus „EVENTVALUE“ mit dem Variablenwert durchgeführt (Zeilen 232 & 236). Dazu wird einmal der Variablenwert und einmal der Wert aus

„EVENTVALUE“ als Muster auf „equalityTest“ angewendet. So kann die beiderseitige Übereinstimmung der Werte festgestellt werden. Sollte allerdings die Variable nicht in „VARIABLES“ vorkommen, muss sie gebunden werden. Dazu wird zuerst geprüft, ob das Event überhaupt einen Wert für die Variable enthält (Zeile 255). Das ist der Fall, wenn in „EVENTVALUE“ ein Wert enthalten ist. Aus diesem Wert wird ein <eca:variable>-Element erzeugt (Zeilen 256 – 259). Sollte aber kein Wert für die Variable existieren, wird ein <nomatch>-Element ausgegeben (Zeile 262). Damit wird angezeigt, dass der Inhalt des Variablen-Tags aus dem Muster keine Entsprechung im Event hat und damit Muster und Event nicht übereinstimmen.

6.6 Variablenabgleich

Der vorangegangene Abschnitt befasste sich mit dem Vergleich von atomaren Events und ihren Mustern. In diesem geht es um ein Template, das überprüft, ob ein zusammengesetztes Event zu dem Auslöser eines Zustandsübergangs passt. Dazu ist es nötig, dass die im Event vorliegenden Variablen zu denen passen, die in der Automateninstanz angelegt sind. Das heißt, Variablen gleichen Namens müssen gleiche Werte haben. Diesen Abgleich führt das Template „variableTest“ durch. Mittels zweier Parameter wird dem Template sowohl die Instanz (INSTANCE) als auch das kombinierte Event (EVENT) angegeben. Die Ausgabe ist ein boolescher Wert (TRUE oder FALSE), der angibt, ob eine Übereinstimmung zwischen den Variablen gefunden wurde.

```
<xsl:template name="variableTest">                                                    275
  <xsl:param name="INSTANCE" />                                                       276
  <xsl:param name="EVENT" />                                                          277
  <!--Variable zum Ablegen der Ergebnisse der einzelnen Vergleiche für jede Variablenbindung.-->  278
  <xsl:variable name= "TESTRESULT">                                                   279
    <xsl:for-each select="$INSTANCE/ceda:instance/ceda:variables/eca:variable">       280
      <xsl:variable name="CURRENT_VARIABLE" select="."/>                              281
      <!--Jede Variable aus dem Event, die zu einer aus der Instanz passt, untersuchen.-->  282
      <xsl:for-each select="$EVENT/ceda:instance/ceda:variables/eca:variable[./@name = $   283
          CURRENT_VARIABLE/@name]">
        <!--Aufruf des Vergleichs für jeden Knoten innerhalb der Variablen aus dem Event.-->  284
        <xsl:apply-templates select="./node()" mode= "equalityTest" >                 285
          <xsl:with-param name= "CHILDREN" select= "$CURRENT_VARIABLE/node()"  286
              />
        </xsl:apply-templates>                                                        287
        <!--Aufruf des Vergleichs für jeden Knoten innerhalb der Variablen aus der Instanz.-->  288
        <xsl:apply-templates select="$CURRENT_VARIABLE/node()" mode= "equalityTest  289
            " >
          <xsl:with-param name= "CHILDREN" select= "./node()" />                      290
        </xsl:apply-templates>                                                        291
      </xsl:for-each>                                                                 292
    </xsl:for-each>                                                                   293
  </xsl:variable>                                                                     294
```

```
<!--Auswertung der Variablen:-->                                          295
<xsl:choose>                                                              296
<!--Sollte es nur Übereinstimmungen für die einzelnen Variablen geben, ist das Ergebnis positiv.--  297
    >
<xsl:when test="count($TESTRESULT/*) = 0">                                298
  <xsl:text>TRUE</xsl:text>                                               299
</xsl:when>                                                               300
<!--Anderenfalls ist er das nicht.-->                                     301
<xsl:otherwise>                                                           302
  <xsl:text>FALSE</xsl:text>                                              303
</xsl:otherwise>                                                          304
</xsl:choose>                                                             305
</xsl:template>                                                           306
```

Listing 6.13: Template für den Abgleich verschiedener Variablenmengen

Eine doppelte Schleife durchläuft alle Kombinationen von Variablen aus der Instanz und dem Event gleichen Namens (Zeilen 280 & 283). Die Werte dieser Kombinationen werden dann miteinander verglichen, indem die Templates des „equalityTest's“ verwendet werden. Da dieser Test nur prüft, ob die Knoten eines Wertes in dem Zweiten enthalten sind, muss der Test zweimal durchgeführt werden. Einmal mit der Variable aus der Instanz (Zeile 289) und einmal mit der des Events (Zeile 285) als Muster. Das Ergebnis wird in die Variable „TESTRESULT“ geschrieben. Nachdem alle Variablen so bearbeitet wurden, wird diese Variable ausgewertet. Sollte sie kein negatives Ergebnis enthalten, so ist der Abgleich geglückt und ‚TRUE‘ wird ausgegeben (Zeile 298). Anderenfalls wird ‚FALSE‘ ausgegeben.

6.7 Modul Algorithmus

Den Kern der Entwicklungsarbeit bildet das Modul „Algorithmus“. Es beinhaltet die Verarbeitung der Automaten und ihrer Instanzen.

6.7.1 Transformieren der Automateninstanzen

Die Auswahl der Instanzen, die transformiert werden müssen, und der Events, die diese Veränderung auslösen, wird im Template „rekursivStep“ durchgeführt. Dieses Template realisiert die rekursive Funktion aus Abschnitt 5.2. Das Template ist bei weitem das längste und enthält eine Reihe von Fallunterscheidungen. Daher wird es nicht wie die anderen Templates in einem Stück abgedruckt, sondern in einzelnen Teilen. Die jeweils fehlenden Teile werden durch Kommentare ersetzt, die durch Punkte eingefasst werden.

Begonnen wird mit den Parametern des Templates und dem rekursiven Aufruf. Mittels zweier Parameter führt das Template seine Arbeit aus. „OLD_SITUATION“ enthält den Stand der Detek-

tion zu Beginn des Templates. Dazu gehören die Automateninstanzen und die bisher detektierten Events. Der zweite Parameter „COUNTER“ beinhaltet die Nummer des Rekursionsschrittes, in dem sich das Template befindet.

```
<xsl:template name= "rekursivStep">
   <xsl:param name= "OLD_SITUATION" />
   <xsl:param name= "COUNTER" select= "'0'" />

   <!-- Variable mit der neuen, zu erstellenden Situation. -->
   <xsl:variable name= "NEW_SITUATION" >
      ...
      <!-- Erzeugen der neuen Situation. -->
      ...
      <!-- Die alten Events der neuen Situation hinzufügen. -->
      <xsl:copy-of select= "$OLD_SITUATION/eventwrapper" />
   </xsl:variable>
   <!-- Fallunterscheidung 2: Muss ein weiterer Rekursionsaufruf durchgeführt werden? -->
   <xsl:choose>
   <!-- Fall 2.1.a: Falls sich keine Änderungen ergeben haben, die Rekursion abbrechen. -->
   <xsl:when test= "count($NEW_SITUATION/child::*) = count($OLD_SITUATION/child::*)" >
      <!-- Neue Situation ausgeben. -->
      <xsl:copy-of select= "$NEW_SITUATION" />
   </xsl:when>
   <!-- Fall 2.1.b: Es gab Änderungen im aktuellen Schritt. Daher wird noch ein Rekusionsschritt
        durchgeführt. -->
   <xsl:otherwise>
      <xsl:call-template name= "rekursivStep" >
         <xsl:with-param name= "COUNTER" select= "$COUNTER + 1" />
         <xsl:with-param name= "OLD_SITUATION" select= "$NEW_SITUATION" />
      </xsl:call-template>
   </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Listing 6.14: Rekursionsaufruf des Templates „rekursivStep“

Im unteren Bereich des Listings sieht man wie sich das Template selbst aufruft. Dabei wird der COUNTER inkrementiert und die im Vorfeld erzeugte Situation zur Weiterverarbeitung an den neuen Aufruf übergeben. Damit allerdings überhaupt ein solcher Aufruf stattfindet, darf die Abbruchbedingung nicht erfüllt sein. Diese Bedingung wird in der Fallunterscheidung 2 überprüft. Sollte kein Unterschied zwischen der ursprünglichen (OLD_SITUATION) und der neu erzeugten Situation (NEW_SITUATION) vorliegen, wird die Rekursion abgebrochen. Da alle Elemente einer Situation nach ihrer Bearbeitung weiterhin vorliegen, müssen alle Veränderungen sich in neuen Elementen ausdrücken. Damit ist eine Differenz in der Anzahl der Elemente zweier Situationen ein ausreichendes Merkmal dafür, dass eine davon verändert wurde.

Die neue Situation wird in der Variablen „NEW_SITUATION“ erzeugt. In ihrem Inneren

wird jede Automateninstanz daraufhin überprüft, ob sie für die weitere Verarbeitung in Frage kommt. Eine Instanz disqualifiziert sich hierfür, wenn sie entweder in einem Endzustand ist oder die Semantik des dargestellten Operators eine weitere Bearbeitung verbietet. Dies ist immer dann der Fall, wenn Snoop das gleichzeitige Eintreten der Operanden untersagt und die betreffende Instanz bereits geändert wurde. Diese Eigenschaft wird mittels des Attributes „time-constraint“ auf die Automaten übertragen.

```
<xsl:for-each select= "$OLD_SITUATION/instancewrapper">
  <!-- Variable die den aktuelle betrachteten Status speichert. -->
  <xsl:variable name= "CURRENT_INSTANCE" select= "." />
  <!-- Fallunterscheidung 1 für die Verwendbarkeit einer Instanz. Eine Instanz ist verwendbar wenn
       er entweder zu einem Automaten mit zeitlicher Bedingung 'kleinergleich' oder 'non' gehört oder
       die Bedingung 'kleiner' und der Zustand zum aktuellen Zeitpunkt unverändert (@stepcounter
       gleich 0) ist.-->
  <xsl:choose>
  <!-- Fall 1.a: Die Instanz ist für die weitere Verarbeitung verwendbar. -->
  <xsl:when test= "$INPUTFILE/ceda:ceda/ceda:automaton[./@id=$CURRENT_INSTANCE/
       ceda:instance/@automaton][(./@time-constraint='less' and $CURRENT_INSTANCE/
       @stepcounter='0') or ./@time-constraint='equalorless' or ./@time-constraint='non']/
       ceda:state[@id = $CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref][./@typ !=
       'ende' and ./@typ != 'abort']" >
    ...
    <!-- Die Transformationen für diese Instanz ausführen. -->
    ...
  </xsl:when>
  <!-- Fall 1.b: Die Instanz ist für die weitere Verarbeitung nicht verwendbar. -->
  <xsl:otherwise>
    <!-- Kopieren des Unveränderten Zustands in den Output dieses Schrittes -->
    <xsl:copy-of select= "$CURRENT_INSTANCE" />
  </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

Listing 6.15: Auswahl der verwendbaren Automateninstanzen

Der XPath-Ausdruck zur Auswahl der Instanzen offenbart ein Problem bei der Verwendung der Funktion id(). In Variablen ist das Rückverfolgen von IDREF-Attributen nicht möglich. Daher ist es nötig einen etwas umständlichen Vergleich zu schreiben. Dieser Vergleich kann aber nicht in die Bedingung der For-Each-Schleife mit aufgenommen werden. Um die Instanzen auszuwählen wird daher, ausgehend von allen Automaten, derjenige ausgewählt, dessen ID zur aktuell betrachteten Instanz passt. Nach dieser Hürde wird überprüft, ob die Bedingungen bezüglich der zeitlichen Abfolge erfüllt sind. Nur solche Instanzen kommen weiter, die entweder seit Beginn des Algorithmus unverändert sind oder das gleichzeitige Auftreten von Events zulassen. Im zweiten Schritt des Ausdrucks werden diejenigen Instanzen ausgeklammert, die sich in einem Endzustand befinden. Sollte eine dieser Bedingungen nicht zutreffen, wird die Instanz unverändert in die neue Situation

übernommen.

Für alle Automateninstanzen, die ausgewählt wurden, durchläuft das Template zwei Phasen. In der Ersten wird eine Variable namens „USABLE_EVENTS“ erzeugt. Basierend auf ihrem Inhalt wird dann die eigentliche Transformation vorgenommen. Die Variable enthält dabei alle Events, die auf die Automateninstanz anwendbar sind. Das Erzeugen der Variable ist notwendig, damit die darauf folgende Fallunterscheidung durchgeführt werden kann. Ohne sie wäre es nicht möglich zu unterscheiden, ob eine Instanz in diesem Rekursionsschritt verändert wurde oder nicht. In letzterem Fall muss sie unverändert übertragen werden.

```
<xsl:variable name= "USABLE_EVENTS" >
   ...
   <!-- Bestimmt, ob es zur betrachteten Instanz ein anwendbares Event gibt. -->
   ...
</xsl:variable>
<xsl:choose>
<!-- Fall 1.1.2.a: Es existieren anwendbare Events in der Variablen USABLE_EVENTS. -->
<xsl:when test= "count($USABLE_EVENTS/*) > 0" >
   ...
   <!-- Anwenden der Events auf die Instanzen. -->
   ...
</xsl:when>
<!--Fall 1.1.2.b: Die Instanz kann nicht geändert werden.-->
<xsl:otherwise>
   <!-- Kopieren des unveränderten Zustands in den Output dieses Schrittes. -->
   <xsl:copy-of select= "$CURRENT_INSTANCE" />
</xsl:otherwise>
</xsl:choose>
```

Listing 6.16: Auswahl passender Events

Um die Variable „USABLE_EVENTS“ zu erzeugen, wird jede Transition der aktuellen Instanz mit jedem unverarbeiteten Event verglichen. Falls der Auslöser einer Transition mit einem atomaren Event verglichen wird, kommt das Template „equalityTest“ zum Einsatz. Sollte das Event ein zusammengesetztes sein, wird dagegen „variableTest“ angewandt. „equalityTest“ dient dazu feststellen, ob das atomare Event das gewünschte Muster erfüllt und berücksichtigt dabei vorliegende Variablenbindungen. Im Fall von zusammengesetzten Events gibt es keine Muster. Stattdessen sind solche Events über die ID des zugehörigen Detektionsautomaten identifizierbar. Zusätzlich zu gleichen ID's müssen aber auch die Join-Variablen übereinstimmen. Den Vergleich der Variablen aus der aktuell betrachteten Automatensituation mit denen, die im zusammengesetzten Event gebunden sind, übernimmt das Template „variableTest“. Für beide Arten von Events wird bei einem positiven Ergebnis das jeweilige Event in die Variable kopiert.

```
<xsl:variable name= "USABLE_EVENTS" >
```

```
<!-- Jede Transition der betrachteten Instanz behandeln. -->
<xsl:for-each select= "$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
    CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/ceda:transition" >
  <!-- Variable mit der behandelten Transition. -->
  <xsl:variable name= "CURRENT_TRANSITION" select= "." />
    <!-- Fallunterscheidung 1.1.1: Nach primitiven und kombinierten Events. -->
    <xsl:choose>
    <!-- Fall 1.1.1.a: Primitives Event. -->
    <xsl:when test= "./@typ = 'primitiv'" >
      <!-- Anfang des Vergleichs des Events mit dem Muster im Automat -->
      <xsl:for-each select= "$OLD_SITUATION/eventwrapper[./@typ = 'primitiv'][./
          @stepcounter = $COUNTER]" >
        <xsl:variable name= "EQUALITYTESTRESULT" >
          <!-- Dieses Template führt den Vergleich zwischen den atomaren Events und
              Eventpattern durch.-->
          <xsl:call-template name= "equalityTest" >
            <xsl:with-param name= "EVENT" select= "./*[1]" />
            <xsl:with-param name= "PATTERN" select= "$CURRENT_TRANSITION/
                child::*[1]" />
            <xsl:with-param name= "VARIABLES" select= "$CURRENT_INSTANCE/
                ceda:instance/ceda:variables/child::eca:variable" />
          </xsl:call-template>
        </xsl:variable>
        <!-- Falls eine Übereinstimmung vorliegt: -->
        <xsl:if test= "$EQUALITYTESTRESULT/text() = 'TRUE'" >
          <!-- Kopieren des Events, das auf die Instanz anwendbar ist. -->
          <xsl:copy-of select= "." />
        </xsl:if>
      </xsl:for-each>
    </xsl:when>
    <!-- Fall 1.1.1.b: Zusammengesetztes Event. -->
    <xsl:when test= "./@typ = 'composite'" >
      <!-- Jedes Composite Event, das zu dem Auslöser der gerade betrachteten Transition passt
          , kopieren. -->
      <xsl:for-each select= "$OLD_SITUATION/eventwrapper[./@typ = 'composite'][./
          @stepcounter = $COUNTER][./ceda:instance/@automaton = $
          CURRENT_TRANSITION/@subautomaton]" >
        <xsl:variable name="VARIABLETESTRESULT" >
          <xsl:call-template name="variableTest" >
            <xsl:with-param name="INSTANCE" select="$CURRENT_INSTANCE" />
            <xsl:with-param name="EVENT" select="." />
          </xsl:call-template>
        </xsl:variable>
        <xsl:if test="$VARIABLETESTRESULT/text()='TRUE'">
          <xsl:copy-of select="." />
        </xsl:if>
      </xsl:for-each>
    </xsl:when>
    </xsl:choose>
```

```
  </xsl:for-each>
</xsl:variable>
```

Listing 6.17: Erzeugen der Variable „USABLE_EVENTS“

Sollte die Variable „USABLE_EVENTS“ mindestens ein Element enthalten, wird die eigentliche Bearbeitung der Instanz begonnen. Wieder werden alle Transitionen mit den vorliegenden Events verglichen. Dabei kann die Auswahl der Events auf die Elemente aus „USABLE_EVENTS“ eingeschränkt werden, da dort bereits alle anwendbaren Events versammelt sind. Für jede dieser Kombinationen wird der Vergleich zwischen Auslöser und Event wiederholt und gegebenenfalls der Zustandsübergang ausgeführt. Diese Aufgabe ist in ein eigenes Template ausgelagert. An die Erzeugung der Instanzen mit den neuen Zuständen schließt sich das Kopieren der ursprünglichen Instanz an. Dieses Duplizieren wird für jede Instanz nur einmal ausgeführt, egal wie viele Events darauf angewandt wurden. Die Kopie kann als instabil markiert werden. Eine instabile Kopie wird immer dann erzeugt, wenn entweder das „duplicate“-Attribut der ursprünglichen Instanz auf ‚no‘ steht oder der Zustand eines der neu erzeugten Instanzen vom Typ ‚abort‘ ist. Letzterer Fall tritt beim Periodic- und Aperiodic-Event-Operator auf. Er sorgt dafür, dass das übliche Verhalten unterbunden wird. Normalerweise werden bei diesen Operatoren stabile Kopien erzeugt. Immer wenn die Kopie nicht explizit als instabil markiert wird, wird eine exakte Kopie erzeugt.

```
<xsl:when test= "count($USABLE_EVENTS/*) > 0" >
  <xsl:variable name="NEW_INSTANCES" >
    <!-- Die Transitionen des aktuellen Zustands der betrachteten Instanz anfassen. -->
    <xsl:apply-templates select="$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
        CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/ceda:transition" mode="
        transformation">
      <xsl:with-param name="USABLE_EVENTS" select="$USABLE_EVENTS" />
      <xsl:with-param name="CURRENT_INSTANCE" select="$CURRENT_INSTANCE" />
      <xsl:with-param name="COUNTER" select="$COUNTER" />
    </xsl:apply-templates>
  </xsl:variable>
  <!-- Fallunterscheidung 1.1.2.2. zur Duplizierung alter Instanzen. -->
  <xsl:choose>
  <!-- Fall 1.1.2.2.a: Es soll dupliziert werden. -->
  <xsl:when test= "($INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
      CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/@duplicate='yes') and ( not
      ( $NEW_INSTANCES/instancewrapper/ceda:instance/ceda:current-state[./@ref= $
      INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@typ='abort']/@id]))" >
    <xsl:copy-of select= "$CURRENT_INSTANCE" />
  </xsl:when>
  <!--Fall 1.1.2.2.b: Es darf nicht dupliziert werden -->
  <xsl:otherwise>
    <xsl:call-template name= "createInstableCopy" >
      <xsl:with-param name= "OLD_INSTANCE" select= "$CURRENT_INSTANCE" />
```

```
      </xsl:call-template>
    </xsl:otherwise>
    </xsl:choose>
    <xsl:copy-of select="$NEW_INSTANCES/*" />
</xsl:when>
```

Listing 6.18: Transformieren der Instanzen

Beispiel 6.2 *Anhand dieses Beispiels wird das Vorgehen weniger abstrakt verdeutlicht. Der Eventausdruck $B;(A\Delta B)$ liegt vor. Aus ihm gehen zwei Automaten $\mathcal{A}^{SEQ}$ und $\mathcal{A}^{\Delta}$ hervor. Für dieses Beispiel liegen die Instanzen $\mathcal{A}_1^{SEQ}$, $\mathcal{A}_1^{\Delta}$ und $\mathcal{A}_2^{\Delta}$ vor. Die Instanzen mit Index 1 entsprechen den initialen Instanzen; in $\mathcal{A}_2^{\Delta}$ ist das Event A bereits eingetreten. Als nächstes atomares Event tritt B auf. In dieser Situation wird „rekursivStep“ aufgerufen. Zu Beginn wird $\mathcal{A}_1^{SEQ}$ behandelt. Diese Instanz ist verwendbar, da sie eine der ursprünglichen Instanzen ist. Die einzige Transition des aktuellen Zustands reagiert auf das Event B. Da dies vorliegt, wird die Instanz transformiert. Das Ergebnis sind die ursprüngliche Instanz $\mathcal{A}_1^{SEQ}$ und eine neue $\mathcal{A}_2^{SEQ}$. Die beiden weiteren Instanzen führen gleichermaßen zu den Instanzen $\mathcal{A}_1^{\Delta}$ und $\mathcal{A}_3^{\Delta}$ sowie $\mathcal{A}_2^{\Delta}$ und $\mathcal{A}_4^{\Delta}$. $\mathcal{A}_4^{\Delta}$ löst allerdings das zusammengesetztes Event $\mathcal{E}_{A\Delta B}$ aus. Damit wurde die Situation um ein Event und zwei neue Automateninstanzen erweitert. Im zweiten Aufruf von „rekursivStep“ muss für jede Instanz überprüft werden, ob $\mathcal{E}_{A\Delta B}$ anwendbar ist. Der aktuelle Zustand von $\mathcal{A}_2^{SEQ}$ besitzt eine Transition, die für $\mathcal{E}_{A\Delta B}$ schalten müsste. Allerdings erlaubt der Sequenzoperator nicht, dass Events, die zum selben Zeitpunkt auftreten, angewandt werden. Deshalb kommt $\mathcal{E}_{A\Delta B}$ hier nicht in Frage. Auch sonst ergeben sich in diesem Schritt keine Änderungen, so dass der Algorithmus sich beenden kann.*

6.7.2 Zustandsübergänge ausführen

Um den Zustandsübergang einer Automateninstanz zu ermöglichen, ist dieses Template gedacht. Es wählt aus einer Menge von Events ein passendes aus und führt gemäß der aktuellen Transition den Übergang aus. Sollte kein passendes Event gefunden werden, wird keine Veränderung der Situation vorgenommen. Die Liste der Events ist im Parameter „USABLE_EVENTS“ enthalten. Darüber hinaus sind die Parameter „CURRENT_INSTANCE“ und „COUNTER“ vorhanden. Beide beinhalten Informationen, die benötigt werden, um die neuen Automateninstanzen zu erzeugen. Einerseits die Hülle der Instanz, die geändert werden soll, und andererseits die Nummer des Rekursionsschritts, in dem die Transformation vorgenommen wird. Das Ergebnis ist wiederum eine Instanz-Hülle mit der neu erstellten Instanz. Sollte sich aus dem neuen Zustand allerdings ein Event ergeben, wird dieses anstelle der Instanz ausgegeben. Auch dieses Event wird in eine Hülle verpackt.

```
<xsl:template match="ceda:transition" mode="transformation" >                    164
  <xsl:param name="USABLE_EVENTS" />                                             165
```

```
<xsl:param name="CURRENT_INSTANCE" />
<xsl:param name="COUNTER" />
<xsl:variable name= "CURRENT_TRANSITION" select= "." />

<!-- Fallunterscheidung 1: Ist der Auslöser der Transition primitiv oder composite. -->
<xsl:choose>
<!-- Fall 1.a: Primitives Events als Auslöser einer Transition. -->
<xsl:when test= "./@typ = 'primitiv'" >
  <!-- Anfang des Vergleichs des Events mit dem Muster im Automat. -->
  <xsl:for-each select= "$USABLE_EVENTS/eventwrapper[./@typ = 'primitiv']">
    <!-- Diese Variable nimmt das Ergebnis des Vergleichs auf. -->
    <xsl:variable name= "EQUALITYTESTRESULT" >
      <!-- Dieses Template führt den Vergleich zwischen den atomaren Events und Eventpattern
           durch.-->
      <xsl:call-template name= "equalityTest" >
        <xsl:with-param name= "EVENT" select= "./*[1]" />
        <xsl:with-param name= "PATTERN" select= "$CURRENT_TRANSITION/child::
            *[1]" />
        <xsl:with-param name= "VARIABLES" select= "$CURRENT_INSTANCE/
            ceda:instance/ceda:variables/child::eca:variable" />
      </xsl:call-template>
    </xsl:variable>
    <!-- Falls eine Übereinstimmung vorliegt: -->
    <xsl:if test= "$EQUALITYTESTRESULT/text() = 'TRUE'" >
      <!-- Überprüft, ob der neue Zustand zu einem Event führt. -->
      <xsl:call-template name= "checkIfNewInstanceIsEvent">
        <xsl:with-param name= "NEW_INSTANCE">
          <!--Dieser Aufruf erzeugt eine neue Instanz mit dem geänderten Zustand.-->
          <xsl:call-template name= "createNewInstanceWithAtomicEvent" >
            <xsl:with-param name= "OLD_INSTANCE" select= "$
                CURRENT_INSTANCE" />
            <xsl:with-param name= "EVENT" select= "." />
            <xsl:with-param name= "TARGET" select= "$CURRENT_TRANSITION/
                @target" />
            <xsl:with-param name= "STEP" select= "$COUNTER + 1" />
            <xsl:with-param name= "NEW_VARIABLES" select= "$
                EQUALITYTESTRESULT/eca:variable" />
          </xsl:call-template>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:for-each>
</xsl:when><!-- Ende Fall 1.a -->
<!-- Fall 1.b: Eine Transition mit Composite Event als Auslöser. -->
<xsl:when test= "./@typ = 'composite'" >
  <!-- Vergleich jedes neuen kombinierten Events mit jeder Transition der betrachteten Instanz.
       -->
  <xsl:for-each select= "$USABLE_EVENTS/eventwrapper[./@typ = 'composite']">
    <xsl:variable name="VARIABLETESTRESULT" >
```

```
208   <xsl:call-template name="variableTest" >
209     <xsl:with-param name="INSTANCE" select="$CURRENT_INSTANCE" />
210     <xsl:with-param name="EVENT" select="." />
211   </xsl:call-template>
212 </xsl:variable>
213 <xsl:if test="$VARIABLETESTRESULT/text()='TRUE'">
214   <!-- Überprüft, ob der neue Zustand zu einem Event führt. -->
215   <xsl:call-template name= "checkIfNewInstanceIsEvent">
216     <xsl:with-param name= "NEW_INSTANCE">
217       <!--Dieser Aufruf erzeugt eine neue Instanz mit dem geänderten Zustand.-->
218       <xsl:call-template name= "createNewInstanceWithCompositeEvent" >
219         <xsl:with-param name= "OLD_INSTANCE" select= "$
              CURRENT_INSTANCE" />
220         <xsl:with-param name= "EVENT" select= "." />
221         <xsl:with-param name= "TARGET" select= "$CURRENT_TRANSITION/
              @target" />
222         <xsl:with-param name= "STEP" select= "$COUNTER + 1" />
223       </xsl:call-template>
224     </xsl:with-param>
225   </xsl:call-template>
226 </xsl:if>
227 </xsl:for-each>
228 </xsl:when><!-- Ende Fall 1.b -->
229 </xsl:choose><!-- Ende Fallunterscheidung 1 -->
230 </xsl:template>
```

Listing 6.19: Template für die Ausführung der Zustandsübergänge

Um seine Aufgabe wahrzunehmen, werden die Transitionen für atomare (Zeile 173) und zusammengesetzte (Zeile 204) Events unterschieden. Für erstere wird der „equalityTest“ verwendet, um das passende Event zu finden (Zeile 179) und für zweiteres das Template „variableTest“ (Zeile 208). Bei positivem Ergebnis werden jeweils die Templates „checkIfNewInstanceIsEvent“ und „createNewStateWithCompositeEvent“ aufgerufen, die die neue Instanz erzeugen und prüfen, ob ein Event vorliegt.

6.7.3 Erkennen von kombinierten Events

Mittels dem obigen Template können zwar die Automaten transformiert werden, aber es fehlt eine direkte Möglichkeit Events zu erkennen, die sich aus der neuen Situation ergeben. Um diese Funktion zu realisieren, wird stattdessen nach jedem Zustandsübergang ein weiteres Template aufgerufen. Dieses prüft, ob aus dem neuen Zustand ein Event hervorgeht. Das Template heißt „checkIfNewInstanceIsEvent“. Im einzigen Parameter „NEW_INSTANCE“ wird die zu prüfende Automateninstanz übergeben. Das Ergebnis dieses Templates ist die interne Darstellung eines zusammengesetzten Events oder, falls ein solches nicht vorliegt, die Instanz aus der Eingabe.

```
<xsl:template name= "checkIfNewInstanceIsEvent" >                                    13
   <xsl:param name= "NEW_INSTANCE" />                                                14
                                                                                     15
   <!--Fallunterscheidung ob der Zustand der Instanz ein Event auslöst.-->           16
   <xsl:choose>                                                                      17
   <!--Die Automateninstanz repräsentiert ein Event.-->                              18
   <xsl:when test= "$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[@id = $NEW_INSTANCE 19
       /instancewrapper/ceda:instance/ceda:current-state/@ref][@detect = 'yes']" >
      <!--Erzeugen der Repräsentation des Events für die weitere Verarbeitung-->     20
      <xsl:call-template name= "createCompositeEvent" ><!--Modul Event-->            21
         <xsl:with-param name= "INSTANCE" select= "$NEW_INSTANCE/instancewrapper/     22
             ceda:instance" />
         <xsl:with-param name= "STEP" select= "$NEW_INSTANCE/instancewrapper/         23
             @stepcounter" />
      </xsl:call-template>                                                           24
   </xsl:when>                                                                       25
   <!--Die Automateninstanz repräsentiert kein Event.-->                             26
   <xsl:otherwise>                                                                   27
      <xsl:copy-of select= "$NEW_INSTANCE" />                                        28
   </xsl:otherwise>                                                                  29
   </xsl:choose>                                                                     30
</xsl:template>                                                                      31
```

Listing 6.20: Template für die Bestimmung eines Events

Die Überprüfung, ob ein zusammengesetztes Event vorliegt, wird in Zeile 19 vorgenommen. Dazu wird aus der Instanz der aktuelle Status bestimmt und dessen Attribut „detect“ ausgelesen. Wenn es den Wert ‚yes‘ hat, liegt ein Event vor. Dann wird das Event durch Aufruf des Templates „createCompositeEvent“ aus dem Modul Event erzeugt (Zeile 21). Sollte allerdings kein Event vorliegen, wird die Automateninstanz aus dem Eingabeparameter in die Ausgabe kopiert (Zeile 28).

Kapitel 7

Java-Umgebung

Als letztes Kapitel in diesem Buch wird die Implementierung einer Umgebung besprochen, die es ermöglicht, die bisherigen Ergebnisse im ECA-Framework (siehe Kapitel 2) verfügbar zu machen. Dabei wird vor allem darauf eingegangen, wie die Schnittstellen zu anderen Komponenten des Frameworks umgesetzt werden.

Um dieses Ziel zu erreichen, wird die Programmiersprache Java verwendet. Aus der Enterprise Edition kommt dabei die Technologie der „Servlets“ zum Einsatz. Für die Unterstützung von SOAP werden die Java-API's SAAJ und „Java for XML Messaging“ (JAXM) verwendet. Ersteres ermöglicht es SOAP-Nachrichten zu verarbeiten, zweiteres implementiert die Kommunikation mittels HTTP. Zur Verarbeitung von XML und XSLT wird das API „Java for XML Processing“ (JAXP) zusammen mit JDOM verwendet. Die Quellen für diese Sprachen und API's sind in Anhang A aufgeführt

Für die Anbindung des XSLT-Prozessors wird die JAXP-API verwendet. Dadurch wird die Auswahl an Prozessoren auf diejenigen beschränkt, die diese Schnittstelle unterstützen. Diese Auswahl umfasst zwei Versionen von Xalan und den Prozessor Saxon. Die beiden Xalan's sind der Prozessor, der in JAXP integriert ist und die wesentlich neuere, aktuelle Version 2.6. Leider sind beide Xalan's nicht einsetzbar, da aus unerfindlichen Gründen die XSLT-Stylesheets aus dieser Arbeit nicht verarbeitet werden. Die Fehlermeldung des Xalan 2.6 lautet dabei wie folgt:
„file:///.../ceda-algorithm.xsl; Zeilennummer51; Spaltennummer58; XSLT- Fehler (javax.xml. transform.TransformerException): java.lang.ClassCastException: org.apache.xpath.objects. XRTreeFrag“.
Damit bleibt nur der Saxon-Prozessor übrig, um die XSLT-Stylesheets anzuwenden.

Obwohl in Java eine DOM-Implementierung für die Verarbeitung von XML-Daten enthalten ist, wird für diese Aufgabe JDOM verwendet. Das hat mehrere Gründe. Zum einen ist sie speziell für Java entwickelt worden, was bei DOM nicht der Fall ist. Dadurch wird eine bessere Performanz

erreicht. Ein anderer Grund liegt in der Verwendung von Saxon als XSLT-Prozessor. JAXP stellt zwar Schnittstellen bereit, die es erlauben XSLT zusammen mit DOM-Bäumen zu verwenden, aber Saxon unterstützt diese nicht. JDOM hingegen wird sowohl durch JAXP als auch durch Saxon unterstützt. Ein weiterer Grund ist, dass es mit JDOM auf einfache Art möglich ist, die Dokumente zu serialisieren. Eine Möglichkeit, die DOM nur über Umwege bereitstellt.

Die Quellcodes der im Folgenden beschriebenen Java-Klassen können vollständig dem Anhang D entnommen werden. Hinweise zur Installation der Servlets und ihrer Verwendung sind in Anhang A enthalten.

7.1 Basisklassen

Als Basisklassen werden diejenigen Klassen bezeichnet, die im Hintergrund Daten und Funktionen bereitstellen. Mit solchen Klassen wird ein gemeinsamer Datenbestand für die einzelnen Servlets und ein Wrapper für die Composite-Event-Automaten realisiert. Wie diese Klassen verwendet werden, kann man bei der Beschreibung der Servlets sehen. Hier werden, ihre Realisierungen vorgestellt. Gemeinsam haben sie, dass sie sich im Paket „rewerse.ced“ befinden.

7.1.1 Automatons

Die Klasse „Automatons“ repräsentiert ein XML-Dokument mit Automaten zur Detektion von Events. In der Klasse wird der Name der repräsentierten Datei abgelegt. Zusätzlich enthält sie das Stylesheet um Ausdrücke in Automaten umzuwandeln. Die Methoden dieser Klasse ermöglichen es weitere Automaten in das Dokument mit aufzunehmen und zu entfernen, solche unter Zuhilfenahme von XSLT zu erzeugen und das geänderte Dokument wieder zurück in seine Datei zu schreiben. Außerdem bietet sie direkten Zugriff auf das XML-Dokument mittels JDOM.

Da bei einer Transformation der Automaten Endzustände und zusammengesetzte Events entstehen können, diese aber nicht weiter behandelt werden, gibt es zwei Methoden um das Dokument „aufzuräumen“. Die erste Methode liest die Instanzen aus, die für ein detektiertes Event stehen. Erkennbar ist das über das Attribut „detect“ des aktuellen Zustands. Die zweite Methode löscht alle Instanzen, die sich in einem Endzustand befinden. Beide Methoden delegieren die Arbeit an das XSLT-Stylesheet „ced-organisation.xsl“. Welche Aufgabe das Stylesheet wahrnimmt, wird über den Parameter „MODE“ geregelt.

Beim Auslesen der Events wird durch das Stylesheet auch die Struktur erzeugt, mit der die Events an die ECA-Engine verschickt werden.

```
<xsl:template name= "getEvents">                                              36
  <xsl:for-each select= "/ceda:ceda/ceda:instance[./ceda:current-state/@ref = //ceda:state[./   37
      @detect='yes']/@id]" >
```

```
<eca:answer xmlns:eca="http://www.eca.org/eca-ml" component="event">                    38
  <xsl:attribute name= "internalid"><xsl:value-of select= "./@automaton" /></xsl:attribute  39
      >
  <eca:result>                                                                          40
    <!-- Kopiert die Abfolge der Events. -->                                           41
    <xsl:for-each select="./ceda:parameterstore/*[./@arrivaltime]␣|␣./ceda:parameterstore//  42
        ceda:composite-event/*[./@arrivaltime]" >
      <xsl:sort select="./@arrivaltime" />                                            43
      <xsl:copy-of select= "." />                                                     44
    </xsl:for-each>                                                                   45
  </eca:result>                                                                       46
  <!-- Überprüft, ob Variablenbindungen vorliegen. Wenn ja wird ein eca:variable-bindings  47
      Element erzeugt. -->
  <xsl:if test= "./ceda:variables/eca:variable">                                      48
    <eca:variable-bindings>                                                           49
      <eca:tuple>                                                                     50
        <!-- Kopiert die Variablenbindungen -->                                       51
        <xsl:copy-of select= "./ceda:variables/eca:variable" />                       52
      </eca:tuple>                                                                    53
    </eca:variable-bindings>                                                          54
  </xsl:if>                                                                           55
</eca:answer>                                                                         56
</xsl:for-each>                                                                       57
</xsl:template>                                                                       58
```

Listing 7.1: Template „getEvents“ aus ced-organisation

Die Auswahl der Instanzen erfolgt in Zeile 37. Der XPath-Ausdruck wäre eleganter mittels der Funktion „id()“ zu formulieren, allerdings lässt sich diese nicht verwenden, da das Automatenformat nicht auf seine DTD validierbar ist (siehe Kapitel 4 Seite 37). Wie man sieht, wird das ECA-Markup aus [BFMS05] verwendet, um eine Antwort zu erzeugen. In <eca:result> werden die atomaren Events abgelegt, die zu dem kombinierten Events geführt haben (Zeile 40). Dabei werden nur die atomaren Events ausgegeben. Eventuell vorhandene strukturierende Elemente wie <eca:composite-event> werden entfernt (Zeile 42). Darüber hinaus werden die Elemente nach dem Zeitpunkt ihres Eintreffens bei der Snoop-Einheit sortiert (Zeile 43). Für die Variablen, falls diese existieren, wird ein <eca:variable-bindings>-Element angelegt (Zeile 49). Außerdem werden die Variablen zu einem Tupel zusammengefasst und dort abgelegt (Zeilen 50 und 53). Zu beachten ist, dass das Ergebnis dieses Templates kein XML-Dokument ist, sondern eine Reihe von Elementen.

Instanzen, die einen Endzustand oder einen Abbruchzustand erreicht haben, können diesen nicht mehr verlassen. Daher sind sie für die weitere Detektion uninteressant. Somit können sie ohne weitere Verluste entfernt werden. Vor dem Entfernen der Instanzen muss allerdings sichergestellt werden, dass alle Composite Events bereits zur ECA-Engine gesandt wurden, denn die Endzustände können auch Auslöser für Events sein. Das Entfernen der Instanzen übernimmt die

Methode „removeCompositeEvents()“ mittels dem Template „deleteFinalInstances“:

```
<xsl:template name= "deleteFinalInstances">                                          61
   <ceda:ceda>                                                                       62
      <!-- Kopieren der ursprünglichen Attribute in das neue ceda-Element.-->        63
      <xsl:copy-of select= "ceda:ceda/@*" />                                         64
      <!-- Kopieren der Automatenspezifikationen, da sie unveränderlich sind.-->     65
      <xsl:copy-of select= "ceda:ceda/ceda:automaton" />                             66
      <!-- Kopiert all die Instanzen, die nicht in einem Endzustand sind. -->        67
      <xsl:copy-of select= "/ceda:ceda/ceda:instance[not(./ceda:current-state/@ref = //ceda:state[./   68
          @typ='ende' or ./@typ='abort']/@id)]" />
   </ceda:ceda>                                                                      69
</xsl:template>                                                                      70
```

Listing 7.2: Template „deleteFinalInstances“ aus ced-organisation

Das Wurzelelement und die Automaten werden dabei unverändert übertragen (Zeilen 62 – 66). Zur Auswahl der weiterhin benötigten Instanzen kommt wieder ein XPath-Ausdruck zum Einsatz (Zeile 68). Auch hier kann die „id()“-Funktion aus oben genanntem Grund nicht verwendet werden. Das Ergebnis des Templates ersetzt das Dokument mit den bisherigen Instanzen.

Eine weitere Methode, die ihre Aufgabe von dem Organisations-Stylesheet erledigen lässt, ist „removeEventExpression()“. Sie dient dazu die Automaten und Instanzen, die aus einem Eventausdruck entstanden sind, wieder zu entfernen. Mittels eines Parameters wird die ID des Automaten übergeben, der gelöscht werden soll. Dieser Parameter wird direkt an das XSLT-Stylsheet weitergegeben. Dort führt das Template „deregisterExpression“ das Entfernen der Elemente aus.

```
<xsl:template name= "deregisterExpression">                                          73
   <xsl:param name="ID"/>                                                            74
                                                                                     75
   <ceda:ceda>                                                                       76
      <!-- Kopieren der ursprünglichen Attribute in das neue ceda-Element.-->        77
      <xsl:copy-of select= "ceda:ceda/@*" />                                         78
      <!-- Kopieren der Automaten, die nicht zur gewählten ID passen. -->            79
      <xsl:copy-of select= "ceda:ceda/ceda:automaton[not( starts-with(@id, $ID) )]" />  80
      <!-- Kopiert alle Instanzen, die nicht zur gewälten ID passen. -->             81
      <xsl:copy-of select= "/ceda:ceda/ceda:instance[not( starts-with(@automaton, $ID) )]" />  82
   </ceda:ceda>                                                                      83
</xsl:template>                                                                      84
```

Listing 7.3: Template „deregisterExpression“ aus ced-organisation

Eigentlich löscht das Template nicht die gewünschten Automaten, sondern kopiert nur die restlichen in die Ausgabe. Gleiches gilt für die Instanzen. Da beim Umwandeln eines Ausdrucks in Automaten alle Automaten-ID‘s mit demselben Bezeichner beginnen, werden alle Automaten beim Kopieren ausgelassen, die mit dem in „ID“ angegebenen String anfangen (Zeile 80). Auf die gleiche Weise werden die Instanzen ausgewählt (Zeile 82).

7.1.2 Event-Consumer

Ein Event-Consumer hat nur eine Aufgabe: Die Transformation der Automaten von einer Situation in eine neue aufgrund eines eingetretenen atomaren Events. Dazu beinhaltet die Klasse ein Objekt vom Typ Automatons und eine Methode, um die darin enthaltenen Automaten zu transformieren. Die Transformation wird durch das Stylesheet „ceda-io.xsl“ durchgeführt. Die Methode ruft über JAXP den XSLT-Prozessor auf und überschreibt das Automatons-Objekt mit dem Ergebnis.

7.1.3 ApplicationData

Da mehrere Servlets auf den selben Daten arbeiten, muss eine Möglichkeit geschaffen werden die Zugriffe der Daten zu synchronisieren, damit keine Änderungen parallel ausgeführt werden und sich damit gegenseitig behindern. Dazu enthält die Klasse ApplicationData mehrere statische Variablen. Diese enthalten unter anderem ein Objekt der Klasse Automatons, auf dessen Basis die Events detektiert werden. In anderen Variablen sind zusätzliche Daten abgelegt. Dazu gehören die Daten, welche Automaten zu welchen ECA-Regeln gehören und an welche URL die erkannten Events zu schicken sind. Beide werden in „Permissions“ abgelegt. Permissions sind eine Java-Klasse, um Paare aus Schlüsseln und Werten aufzunehmen. Außerdem bieten sie die Möglichkeit die Daten in Dateien abzulegen und aus solchen zu lesen. Die Variablen sind als „volatile“ deklariert. Dadurch werden alle Zugriffe auf die Variablen serialisiert durchgeführt.

7.2 Servlets

Die Realisierung der Schnittstellen erfolgt mit Hilfe von Java-Servlets. Diese sind alle im Paket „rewerse.ced.servlets“ enthalten. Dabei wird für jede der Schnittstellen ein Servlet erschaffen. Als Basis dieser Servlets dient die abstrakte Klasse „CEDServlet“. Ableitungen dienen dazu atomare Events bzw. Eventausdrücke entgegenzunehmen und Eventausdrücke wieder zu entfernen. So stehen drei Servlets zur Verfügung. Mit „ReceiveAtomicEventServlet“ lassen sich neue atomare Events aus dem Netzwerk entgegen nehmen. Für die Registrierung neue Eventausdrücke ist „RegisterEventExpressionServlet“ zuständig. Um solche wieder zu entfernen, muss man „DeregisterEventExpressionServlet“ aufrufen.

Die Vorgehensweise zur Verarbeitung der SOAP-Nachrichten ist in allen Servlets ähnlich. Das API JAXM stellt eine Servlet-Klasse bereit, die sich um die Abwicklung der SOAP-Kommunikation kümmert. Von diesem wird CEDServlet abgeleitet. Damit steht die SOAP-Funktionalität in allen Servlets zur Verfügung.

```
package rewerse.ced.servlets;                                    1
...                                                              2
//Importe aus JAXP-API                                           3
```

```
import javax.xml.messaging.JAXMServlet;                                    4
import javax.xml.messaging.ReqRespListener;                                5
                                                                           6
//Deklaration der Klasse                                                   7
public abstract class CEDServlet extends JAXMServlet implements ReqRespListener {  8
   ...                                                                     9
}//end of Class CEDServlet                                                10
```

Listing 7.4: Auszug aus „CEDServlet“ mit Import der JAXP-Funktionalität

JAXMServlet und ReqRespListener ermöglichen es eine einfache Anfrage-Antwort-Struktur für die Kommunikation zu verwenden. Bei Eingang einer SOAP-Nachricht wird automatisch die Methode „ReqRespListener.onMessage()“ aufgerufen, die eine DOM Repräsentation der Nachricht als Parameter enthält. In dieser Methode verarbeiten die einzelnen Servlets die Nachricht. Dazu werden zuerst die Inhalte aus dem Header und dem Body der Nachricht ausgelesen. Diese werden dann durch den Aufruf eines XSLT-Prozessors weiterverarbeitet. Am Ende gibt die Methode eine Antwort-Nachricht zurück, die automatisch versandt wird. Sollte allerdings ein Fehler auftreten, wird statt dessen eine Fehlernachricht (SOAP-Fault) erzeugt und versandt.

Da die Servlets die selben Daten verarbeiten, nämlich die Automaten zu den Eventausdrücken, müssen diese einheitlich zur Verfügung stehen. Dazu wird die Klasse „ApplicationData“ zur Verfügung gestellt, die eine Reihe statischer Variablen enthält. Diese Variablen werden durch die Servlets initialisiert. Dazu wird die Methode „init()“ jedes Servlets erweitert.

```
public void init(ServletConfig servletconfig) throws ServletException{          1
   super.init(servletconfig); //JAXM-Servlet initialisieren                      2
   this.dateformat = new SimpleDateFormat("MM/dd/yyyy_kk/mm");                   3
   //includes the Context-Path for all Fileoperations.                           4
   this.servletcontext = servletconfig.getServletContext();                      5
   try{//try to load the Application Data                                        6
      if (ApplicationData.getAutomatons() == null) ApplicationData.initData(this.servletcontext.   7
            getRealPath("."), "XML/automatons.xml", "WEB-INF/XSLT/create_automaton.xsl");
   }//end try                                                                    8
   catch (Exception e){                                                          9
      throw new ServletException(e);                                             10
   }//end catch Exception                                                       11
}//end of Method init                                                           12
```

Listing 7.5: Auszug aus CEDServlet.init()

Wie man sieht, wird in Zeile 7 überprüft, ob bereits eine Initialisierung vorgenommen wurde. Wenn das nicht der Fall ist, wird die Methode „initData()“ aus der Klasse ApplicationData aufgerufen, die diese Arbeit ausführt.

7.2.1 ReceiveAtomicEventServlet

Das Servlet, das durch „ReceiveAtomicEventServlet“ implementiert wird, reagiert auf SOAP-Messages mit neuen atomaren Events.

Beispiel 7.1 (SOAP-Nachricht mit atomaren Events)
Das folgende Listing enthält eine beispielhafte SOAP-Nachricht zur Übertragung von atomaren Events an die SNOOP-Engine.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header />
  <SOAP-ENV:Body>
    <foo:C xmlns:foo="http://rewerse.net/event-ml">
      Textinhalt
    </foo:C>
    <foo:B xmlns:foo="http://rewerse.net/event-ml">
      <foo:Child>Textinhalt</foo:Child>
      <foo:Child>Textinhalt</foo:Child>
    </foo:B>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Im Inneren des SOAP-Bodys befinden sich zwei atomare Events B und C. Beide sind nur abstrakte Beispiele, um die Struktur der Nachricht zu verdeutlichen.

Eine Antwort auf obige Events sieht wie folgt aus. Das Listing enthält nur den Inhalt des SOAP-Bodys. Die restlichen Elemente der SOAP-Nachricht sind identisch mit denen des obigen Listings.

```
<ced:response xmlns:ced="http://rewerse.net/ced-organisation-ml">
  Received Atomic Events!
</ced:response>
```

Ein einfaches Element aus einem speziellen Namespace repräsentiert die Antworten der Snoop-Engine. Sein Text enthält die Art der Antwort. In diesem Fall eine Bestätigung.

Auf jede Nachricht, wie derjenigen aus Bsp. 7.1, wird die Methode „onMessage()“ angewandt.

```
public SOAPMessage onMessage(SOAPMessage message){                                  85
  SOAPBody messagebody = null; //Body of received SOAP-Message                      86
  SOAPMessage result = null; //Answer for received SOAP-Message                     87
                                                                                    88
  try{                                                                              89
    //Get the SOAP-Body Elements from the Message.                                  90
    messagebody = message.getSOAPPart().getEnvelope().getBody();                    91
    //Creats the Answer Message.*/                                                  92
    result = MessageFactory.newInstance().createMessage();                          93
    SOAPElement responseelement = SOAPFactory.newInstance().createElement("response", "ced",   94
        "http://rewerse.net/ced-organisation-ml").addTextNode("Receive Atomar Events!");
```

```
    result.getSOAPPart().getEnvelope().getBody().addChildElement(responseelement);
}
catch (SOAPException soape){
    if (ApplicationData.debug())soape.printStackTrace(System.err);
    log("CED_Primitiv_Event_Servlet:_It_is_not_a_correct_SOAP-Message_" + soape);
    //Creats a SOAP-Fault-Message because the arrived Message is not correct SOAP
    return this.createFaultMessage();
}//ende try-catch for SOAP-Exception

//Reads the Events from the SOAP-Message (each XML-Fragment insite the SOAP-Body shout be
    one).
Iterator eventiterator = messagebody.getChildElements();
for (;eventiterator.hasNext();){
    Node event = (Node) eventiterator.next();
    if (event instanceof SOAPBodyElement){//only Element-Nodes are from Interesting

        //Creates the XML-File with the Event. With this File the XSLT-Stylesheet can load the
            Event.
        try{
            this.parseEvent((Element)event);
        }//ende try for serializing a Event
        catch (ParserConfigurationException pce){
            log("CED_Primitiv_Event_Servlet:_Can't_generate_the_Event-XML-File!\n___" + pce);
            if (ApplicationData.debug()) System.out.println(pce);
            return this.createFaultMessage();
        }
        catch (java.io.IOException ioe){
            log("CED_Primitiv_Event_Servlet:_Can't_save_the_Event_to_File!\n___" + ioe);
            if (ApplicationData.debug()) System.out.println(ioe);
            return this.createFaultMessage();
        }//end catch MalformedURLException

        //The transformation of the Automatons with the Event is start hier.
        try{
            this.consume();
        }//ende try transform Automaton
        catch (TransformerException te){
            if (ApplicationData.debug())te.printStackTrace(System.err);
            log("CED_Primitiv_Event_Servlet:_Transformation_does_not_work!\n" + te);
            continue;
        }//ende catch TransformerException
        catch (IOException ioe){
            if (ApplicationData.debug())ioe.printStackTrace(System.err);
            log("CED_Primitiv_Event_Servlet:_Can't_write_the_new_Automatonfile!\n" + ioe);
            continue;
        }//end catch IOException
        catch (org.jdom.JDOMException jdome){
            if (ApplicationData.debug())jdome.printStackTrace(System.err);
```

```
            log("CED_Primitiv_Event_Servlet:_Can't_read_the_Events_from_CEDA-Document" +
                jdome);
            continue;
          }
          catch (SOAPException soape){
            if (ApplicationData.debug())soape.printStackTrace(System.err);
            log("CED_Primitiv_Event_Servlet:_Can't_send_a_detected_Event" + soape);
            return this.createFaultMessage();
          }//ende try-catch for SOAP-Exception
        }//ende if Child is SOAPElement
      }//ende for all Events in the SOAP-Message

      /*Returns the Answer-Message*/
      return result;
    }//end Method onMessage(SOAPMessage)
```

Listing 7.6: Methode onMessage der Klasse ReceiveAtomicEventServlet

In dieser Methode wird der Inhalt der Nachricht ausgewertet und die Detektion der zusammengesetzten Events mit den so gewonnenen atomaren Events fortgesetzt.

Zu Beginn der Methode wird eine Antwort-Nachricht erstellt, die nur aus <ced:response>Receive Atomar Events!</ced:response> besteht. Diese Nachricht wird am Ende (Zeile 153) an den Servletcontainer zurückgegeben, der sie dann zustellt. Nachdem die Antwort erzeugt ist, wird eine Liste der Elemente aus der Nachricht erstellt (Zeile 105). Jedes dieser Elemente wird als atomares Event interpretiert. In der Reihenfolge, in der die Events in der Nachricht stehen, wird für jedes von ihnen die Detektion zusammengesetzer Events durchgeführt. Dazu wird zuerst jedes Event durch die Methode „parseEvent()“ aufbereitet und in die Datei „Event.xml“ geschrieben (Zeile 112). Im Anschluss wird die Transformation der Automaten durch die Methode „consume()“ ausgeführt (Zeile 127). Diese Methode verschickt auch die detektierten Events.

Die Methode „parseEvent“ nimmt ein DOM-Element entgegen, um dieses als Event für den XSLT-Prozessor verfügbar zu machen.

```
private void parseEvent(Element event)throws ParserConfigurationException, IOException{
  FileOutputStream out = null;
  try{
    /*create the DOM-Document with on of the Events from the SOAP-Message.*/
    DocumentBuilderFactory domfactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = domfactory.newDocumentBuilder();
    Document eventdoc = builder.newDocument();
    //Creates an attribute an add it to the Event
    event.setAttribute("arrivaltime", this.dateformat.format(new Date()));
    //Add's the Event-Node to the new XML-Document
    eventdoc.appendChild(eventdoc.importNode(event, true));
    //Save the Event in File 'Event.xml' with JDOM.
    org.jdom.output.XMLOutputter jdomoutputter = new org.jdom.output.XMLOutputter();
```

```
        org.jdom.input.DOMBuilder jdombuilder = new org.jdom.input.DOMBuilder();                    174
        out = new FileOutputStream(new File(this.servletcontext.getRealPath("WEB-INF/XSLT/          175
            Event.xml")));
        jdomoutputter.output(jdombuilder.build(eventdoc),out);                                      176
    }                                                                                               177
    finally{                                                                                        178
        try{                                                                                        179
            out.close();                                                                            180
        }catch(Exception e){                                                                        181
            if (ApplicationData.debug())e.printStackTrace(System.out);                              182
            log("CED Primitiv Event Servlet: Can't Close the File with the Event!\n" + e);          183
        }                                                                                           184
    }                                                                                               185
}//end Method parseEvent                                                                          186
```

Listing 7.7: Methode parseEvent der Klasse ReceiveAtomicEventServlet

Für das Event wird zunächst ein Dokument erzeugt (Zeilen 165 ff). Diesem Dokument wird der Methoden-Parameter „event“ als Wurzelelement hinzugefügt (Zeile 171). Zuvor wird das Event allerdings noch um das Attribut „arrivaltime“ erweitert (Zeile 169). Dieses Attribut enthält den Zeitpunkt, zu dem das Event bei dem Servlet eingetroffen ist. Zum Abschluss der Methode wird das gerade erzeugte Dokument in eine Datei geschrieben (Zeile 173 – 176).

In der Methode „consume()“ wird die Detektion mit dem vorher durch „parseEvent()“ erzeugen Event durchgeführt.

```
private void consume() throws TransformerException, org.jdom.JDOMException, IOException,       194
    SOAPException{
    //Transforms the Automatons                                                                 195
    this.consumer.transformAutomaton();                                                         196
    //send new Composite Events to ECA-Engines                                                  197
    List compositeevents = ApplicationData.getAutomatons().getCompositeEvents();               198
    for (int i = 0; i < compositeevents.size(); i++){                                           199
        org.jdom.Element compositeevent = (org.jdom.Element) compositeevents.get(i);            200
        String id = compositeevent.getAttribute("internalid").getValue();                       201
        String ruleid = ApplicationData.loadRuleID(id);                                         202
        if (ruleid != null){                                                                    203
            compositeevent.setAttribute("ref", ruleid);                                         204
            compositeevent.removeAttribute("internalid");                                       205
            java.net.URL endpoint = new java.net.URL(ApplicationData.loadAnswerURL(id));        206
            SOAPMessage eventdetectmessage = MessageFactory.newInstance().createMessage();      207
            Document eventdom = new org.jdom.output.DOMOutputter().output(new org.jdom.         208
                Document(compositeevent));
            eventdetectmessage.getSOAPPart().getEnvelope().getBody().appendChild( eventdetectmessage 209
                .getSOAPPart().importNode(eventdom.getDocumentElement(), true) );
            SOAPConnection soapconn = SOAPConnectionFactory.newInstance().createConnection();   210
            SOAPMessage response = soapconn.call(eventdetectmessage, endpoint);                 211
            soapconn.close();                                                                   212
        }//end if no ruleid                                                                     213
```

```
    }//end for all Composite Events                                    214
    //remove States with detected Events                               215
    ApplicationData.getAutomatons().removeCompositeEvents();           216
    //save the new Situation                                           217
    ApplicationData.getAutomatons().serialize();                       218
}//end Method consume();                                               219
```

Listing 7.8: Methode consume der Klasse ReceiveAtomicEventServlet

Um die benötigten Transformationen auszuführen, wird die Methode „transformAutomaton()“ der Klasse „consumer“ aufgerufen (Zeile 196). Diese startet wiederum den XSLT-Prozessor mit dem Stylesheet „ceda-io.xsl“. Da die Events und abgeschlossenen Automateninstanzen am Ende der Transformation im Automaten-Dokument verbleiben, werden sie nach der Transformation aus dem Dokument ausgelesen und weiterverarbeitet. Dieses geschieht durch eine Instanz der Klasse Automaton (Zeile 198). In den Zeilen 199 bis 214 wird für jedes detektierte Event eine SOAP-Nachricht erzeugt, die an die ECA-Einheit verschickt wird, die den Eventausdruck zum erkannten Event registriert hat.

Beispiel 7.2 (SOAP-Nachricht mit einem Composite-Event)

Diese SOAP-Nachricht ist ein Beispiel für die Übermittlung eines zusammengesetzten Events von der Snoop-Einheit an die ECA-Einheit.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <eca:answer component="event" ref="testingECARule1" xmlns:eca="http://www.eca.org/eca-ml
        ">
      <eca:result>
        <foo:A arrivaltime="2006/06/02_3:25:15" xmlns:foo="http://rewerse.net/event-ml">
          <foo:child attribut="Variablenwert">Textinhalt</foo:child>
        </foo:A>
        <foo:B arrivaltime="2006/06/02_19:57:46" xmlns:foo="http://rewerse.net/event-ml">
          <foo:child attribut="Variablenwert"/>
          <foo:child variable="zweite␣Variable">textinhalt</foo:child>
        </foo:B>
      </eca:result>
      <eca:variable-bindings>
        <eca:tuple>
          <eca:variable name="Variablenname">Variablenwert</eca:variable>
          <eca:variable name="Variablenname2">zweite Variable</eca:variable>
        </eca:tuple>
      </eca:variable-bindings>
    </eca:answer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In dieser Nachricht wird ein Event übermittelt, das aus den atomaren Events A und B kombiniert

wurde. Dabei binden diese Events zwei Variablen.

Um das Automaten-Dokument möglichst klein zu halten, werden in Zeile 216 die Instanzen in Endzuständen entfernt. Anschließend wird das manipulierte Automaten-Dokument wieder zurück in eine Datei geschrieben (Zeile 218). Damit ist das Ergebnis der Transformation gesichert und die Verarbeitung des nächsten atomaren Events kann beginnen.

7.2.2 RegisterEventExpressionServlet

Dieses Servlet verarbeitet eingehende Eventausdrücke in Snoop-Syntax.

Beispiel 7.3 (SOAP-Nachricht mit Snoop-Eventausdrücken)
Die SOAP-Nachricht aus diesem Beispiel überträgt die Daten über einen Eventausdruck an die SNOOP-Einheit.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <answer-url>http://soap.irgendwo.org</anwser-url>
    <rule-id>RULEX</rule-id>
    <eca:variable-bindings xmlns:eca="http://www.eca.org/eca-ml">
      <eca:tuple>
        <eca:variable name="erste">Erster Wert</eca:variable>
        <eca:variable name="zweite">Zeiter Wert</eca:variable>
      </eca:tuple>
      <eca:tuple>
        <eca:variable name="erste">Dritter Wert</eca:variable>
        <eca:variable name="zweite">Vierter Wert</eca:variable>
      </eca:tuple>
    </eca:variable-bindings>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <snoop:Eventdeclaration xmlns:foo="http://rewerse.net/event-ml" xmlns:snoop="http://rewerse.
        net/snoop-expression-ml">
    <snoop:Any number-of-elements="3">
      <snoop:Atomic-Event>
        <foo:EventC/>
      </snoop:Atomic-Event>
      <snoop:And>
        <snoop:Atomic-Event>
          <foo:EventD1 attribute="$erster"/>
        </snoop:Atomic-Event>
        <snoop:Atomic-Event>
          <foo:EventD2/>
        </snoop:Atomic-Event>
      </snoop:And>
      <snoop:Sequence>
        <snoop:Atomic-Event>
```

```
            <foo:EventE1/>
          </snoop:Atomic-Event>
          <snoop:Atomic-Event>
            <foo:EventE2 attribute="$erster"/>
          </snoop:Atomic-Event>
        </snoop:Sequence>
        <snoop:Atomic-Event>
          <foo:EventG attribute="$zweiter"/>
        </snoop:Atomic-Event>
      </snoop:Any>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In dieser Nachricht wird der Eventausdruck $Any(C, (D1 \nabla D2), (E1; E2), G)$ an die Snoop-Einheit übergeben. Dieser Ausdruck wird dort unter der ID „RULEX“ registriert. Sollte ein Event gemäß dem Ausdruck detektiert werden, wird es an die Adresse „http://soap.irgendwo.org“ gesendet. Die im Ausdruck vorkommenden Variablen werden bei der Registrierung mit jeweils zwei Werten vorbelegt. Dadurch wird die Auswahl der atomaren Events $D1$, $E2$ und G eingeschränkt. Die Variablenbelegung geschieht im Header. Alternativ könnten sie im Inneren des <snoop:Eventdeclaration>-Elements im Body angegeben werden. Wenn man das <eca:variable-bindings>-Element dorthin verschiebt, bleibt das Ergebnis das selbe. Beide Angaben können auch kombiniert werden.

Die Antwort auf solche Nachrichten sieht wie folgt aus:

```
<ced:registration xmlns:ced="http://rewerse.net/ced-organisation-ml">
  <ced:rule-id>RULEX</ced:rule-id>
  <ced:internal-id>2006/06/02_19:51_#1</ced:internal-id>
  <ced:timestamp>2006/06/02_19:51</ced:timestamp>
</ced:registration>
```

Mittels eines „Registration“-Elements aus einem speziellen Namespace wird die Antwort angegeben. In der Antwort ist die ID der Regel enthalten, die das Event registrieren lässt. Darüber hinaus wird die interne ID des Events und die Zeit angegeben, zu der die Registrierung eingegangen ist.

Diese Ausdrücke werden in Automaten umgewandelt und zu den bereits bestehenden Automaten hinzugefügt. Damit stehen sie für die Detektion von kombinierten Events bereit. Die anfallenden Arbeiten führt die Methode „onMessage()“ aus.

```
public SOAPMessage onMessage(SOAPMessage message){                              59
  SOAPHeader messageheader = null;//Header of received SOAP-Message             60
  SOAPMessage result = null; //Answer for received SOAP-Message                 61
  Iterator expressioniterator = null;                                           62
                                                                                63
  //Try to get the SOAP-Body and initial the answer Message                     64
  try{                                                                          65
    messageheader = message.getSOAPPart().getEnvelope().getHeader();//get SOAP-Header  66
    //create new SOAP-Message                                                   67
```

```
    result = MessageFactory.newInstance().createMessage();
    //reads the Event Expressions from the SOAP-body
    expressioniterator = message.getSOAPPart().getEnvelope().getBody().getChildElements(message.
        getSOAPPart().getEnvelope().createName("Eventdeclaration", "snoop", "http://rewerse.net
        /snoop-expression-ml"));
  }
  catch (SOAPException soape){
    if (ApplicationData.debug())soape.printStackTrace(System.err);
    log("CED Composite Event Servlet: It is not a correct SOAP-Message " + soape);
    //Creats a SOAP-Fault-Message because the arrived Message is not correct
    return this.createFaultMessage();
  }//ende try-catch for SOAP-Exception

  //reads the Event Expressions from the SOAP-body
  for (;expressioniterator.hasNext();){
    Element expression = (Element) expressioniterator.next();
    try{
      Date arrivaltimestamp = new Date();
      //Creates a new ID for the Automaton
      String idstring = this.dateformat.format(arrivaltimestamp) + "_#" + this.automatoncounter
          ++;
      String ruleid = ((Element)messageheader.getChildElements(message.getSOAPPart().
          getEnvelope().createName("rule-id")).next()).getTextContent();
      String answerurl = ((Element)messageheader.getChildElements(message.getSOAPPart().
          getEnvelope().createName("answer-url")).next()).getTextContent();
      //default Variable-Bindings
      Iterator variableIterator = messageheader.getChildElements(message.getSOAPPart().
          getEnvelope().createName("variable-bindings", "eca", "http://www.eca.org/eca-ml"));

      //Aufruf der Registrierungsmethode
      this.register(expression, idstring, ruleid, answerurl, variableIterator);

      //create a registration-Element and put it to the Answer-Message
      SOAPElement registrationelement = SOAPFactory.newInstance().createElement("registration"
          , "ced", "http://rewerse.net/ced-organisation-ml");
      registrationelement.addChildElement("rule-id", "ced", "http://rewerse.net/ced-organisation
          -ml").addTextNode( ((Element)messageheader.getChildElements(message.getSOAPPart
          ().getEnvelope().createName("rule-id")).next()).getTextContent() );
      registrationelement.addChildElement("internal-id", "ced", "http://rewerse.net/ced-
          organisation-ml").addTextNode(idstring);
      registrationelement.addChildElement("timestamp", "ced", "http://rewerse.net/ced-
          organisation-ml").addTextNode(this.dateformat.format(arrivaltimestamp));
      result.getSOAPPart().getEnvelope().getBody().addChildElement(registrationelement);
    }//end try
    catch (IOException ioe){
      if (ApplicationData.debug())ioe.printStackTrace(System.err);
      log("CED Composite Event Servlet: Can not serialize the Automaton-XML-File or can
          not write the Configuration.\n" + ioe);
      continue;
```

```
105     }//end catch IOException
106     catch (TransformerException te){
107       if (ApplicationData.debug())te.printStackTrace(System.err);
108       log("CED Composite Event Servlet:  \n" + te);
109       continue;
110     }//end catch TransformerException
111     catch (SOAPException soape){//Creats a SOAP-Fault-Message because the arrived Message is
            not correct
112       if (ApplicationData.debug())soape.printStackTrace(System.err);
113       log("CED Composite Event Servlet: Can't create a Answer:   \n" + soape);
114       return this.createFaultMessage();
115     }//ende try-catch for SOAP-Exception
116   }//ende for all Events in the SOAP-Message
117
118   //Returns a Result Message with the ID's of the registratet Expressions
119   return result;
120 }//end Method onMessage(SOAPMessage)
```

Listing 7.9: Methode onMessage der Klasse RegisterEventExpressionServlet

Die Arbeit erfolgt in zwei Blöcken. Zuerst wird die SOAP-Nachricht ausgewertet. In Zeile 65 werden mittels SAAJ alle Eventausdrücke aus dem SOAP-Body ausgelesen. Diese werden über das XML-Element <snoop:Eventdeclaration> aus dem Namespace „http://rewerse.net/snoop-expression-ml“ identifiziert (Zeile 70).

Der zweite Block der Methode nimmt für jeden Eventausdruck die Verarbeitung vor (Zeilen 80 – 116). Zuerst wird eine interne ID für den neuen Automaten erstellt, die sich aus dem aktuellen Zeitpunkt und einer laufenden Nummer zusammensetzt (Zeile 85). Die Informationen aus dem Header werden ausgelesen und in String-Variablen abgelegt. Die URL, an die die detektierten Events gesandt werden sollen, wird dem Element <answer−url> entnommen (Zeile 86). Das gleiche geschieht mit der ID der ECA-Regel aus <rule−id> (Zeile 87). Um die Variablenbindungen aus dem Header in die spätere Erzeugung der Automaten mit aufnehmen zu können, werden sie in einem Iterator abgelegt (Zeile 89). Mit diesen Informationen und dem Eventausdruck wird die Methode „register()“ aufgerufen, die die weitere Verarbeitung übernimmt (Zeile 92). Wenn diese Methode ihre Arbeit erfolgreich abgeschlossen hat, wird ein Registrierungselement <ced:registration> erzeugt, das drei Elemente enthält (Zeilen 95 – 99). <ced:rule−id>, <ced:internal−id> und <ced:timestamp> aus dem Namespace „http://rewerse.net/ced-organisation-ml“ enthalten die ID der Regel, die interne ID der Snoop-Engine und die Zeit, zu dem der Ausdruck eingegangen ist. Diese Informationen werden für jeden Ausdruck in der Nachricht erzeugt und in eine SOAP-Antwort eingefügt. Diese wird zum Abschluss an den Absender der Eventausdrücke übertragen (Zeile 119).

Die Methode „register()“ kapselt die Verarbeitung eines Eventausdrucks von der Kommunikationsschnittstelle.

```
private void register(Element eventexpression, String id, String ruleid, String answerurl,Iterator 128
    variabletuples)
        throws IOException, TransformerException{ 129
  //stores the Id of the Rule from wher the Eventexpression are. 130
  ApplicationData.storeRuleID(id, ruleid); 131
  //stores the Answer URL for the new Event Expression 132
  ApplicationData.storeAnswerURL(id, answerurl); 133
  if ( variabletuples.hasNext() ){ 134
    Element variables = (Element) variabletuples.next(); 135
    eventexpression.appendChild(variables); 136
  } 137
  //creates the JDOM-Representation of the Element 138
  org.jdom.input.DOMBuilder builder = new org.jdom.input.DOMBuilder(); 139
  org.jdom.Element jdomexpression = builder.build((Element)eventexpression); 140
  //Transform the Event Expression to an Automaton 141
  org.jdom.Document newautomatons = ApplicationData.getAutomatons().createAutomaton( 142
      jdomexpression, id);
  //add the new Automatons into the ceda-Document 143
  ApplicationData.getAutomatons().addAutomatons(newautomatons); 144
  //save the new Situation 145
  ApplicationData.getAutomatons().serialize(); 146
}//ende Method register 147
```

Listing 7.10: Methode register der Klasse RegisterEventExpressionServlet

Mittels des Parameters „eventexpression“ wird der Eventausdruck an die Methode übergeben. Die weiteren Parameter nehmen die interne ID, unter der der Ausdruck registriert wird, die ID der ECA-Regel und deren Antwort-URL auf. Darüber hinaus stehen in „variabletuples“ vorbelegte Variablenbindungen zur Verfügung. Die Methode beginnt damit die Regel-ID und die URL unter der internen ID in den Konfigurationsdateien abzulegen (Zeilen 131 & 133). Als nächstes werden die vorbelegten Variablensätze in den Eventausdruck übertragen (Zeilen 134 – 137). Dies geschieht, da keine andere Möglichkeit besteht die Daten an das XSLT-Stylesheet zu übergeben. Die nächste Aufgabe ist der Start der XSLT-Verarbeitung. Dazu wird der Ausdruck vom DOM-Format nach JDOM gewandelt (Zeilen 139f). Dies ist aus oben genannten Gründen nötig. Zeile 142 delegiert dann die eigentliche Transformation an die durch „ApplicationData“ bereitgestellte Instanz der Klasse Automatons. Im Anschluss wird das Ergebnis dieser Transformation mit den bestehenden Automaten in einem XML-Document vereinigt (Zeile 144) und dieses in eine Datei geschrieben (Zeile 146). Diese Aufgaben übernimmt ebenfalls die Klasse Automatons.

7.2.3 DeregisterEventExpressionServlet

Beispiel 7.4 (SOAP-Nachricht zum Löschen eines Eventausdrucks)

Die SOAP-Nachricht aus diesem Beispiel veranlast das Entfernen eines vorhandenen Automaten

zu der ECA-Regel ‚RuleX'.

```
<SOAP−ENV:Envelope xmlns:SOAP−ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP−ENV:Header>
    <rule−id>RULEX</rule−id>
  </SOAP−ENV:Header>
  <SOAP−ENV:Body />
<SOAP−ENV:Envelope>
```

Das dritte Servlet ermöglicht es Eventausdrücke, die bei der Snoop-Engine registriert wurden, wieder von der Verarbeitung auszuschließen. Wie in den anderen Servlets übernimmt die Methode „onMessage()" die Auswertung der SOAP-Message.

```
public SOAPMessage onMessage(SOAPMessage message){                                  56
  SOAPHeader messageheader = null;//Header of received SOAP−Message                 57
  SOAPMessage result = null; //Answer for received SOAP−Message                     58
  Iterator messageelementiterator = null;                                           59
                                                                                    60
  //Try to get the SOAP−Body and initial the answer Message                         61
  try{                                                                              62
    messageheader = message.getSOAPPart().getEnvelope().getHeader();//get SOAP−Header 63
    //create new SOAP−Message                                                       64
    result = MessageFactory.newInstance().createMessage();                          65
    SOAPElement responseelement = SOAPFactory.newInstance().createElement("response", "ced",  66
        "http://rewerse.net/ced−organisation−ml").addTextNode("Derigister␣Event␣Expressions!");
    result.getSOAPPart().getEnvelope().getBody().addChildElement(responseelement);  67
    //reads the Event Expressions from the SOAP−body                                68
    messageelementiterator = messageheader.getChildElements(message.getSOAPPart().getEnvelope  69
        ().createName("rule−id"));
  }                                                                                 70
  catch (SOAPException soape){                                                      71
    if (ApplicationData.debug())soape.printStackTrace(System.err);                  72
    log("CED␣Composite␣Event␣Deregistration␣Servlet:␣It␣is␣not␣a␣correct␣SOAP−Message␣" +  73
        soape);
    //Creats a SOAP−Fault−Message because the arrived Message is not correct        74
    return this.createFaultMessage();                                               75
  }//ende try−catch for SOAP−Exception                                             76
                                                                                    77
  //Starts the Deregistration for each Element in the SOAP−Body                     78
  for (;messageelementiterator.hasNext();){                                         79
    Element expression = (Element) messageelementiterator.next();                   80
    String externalid = expression.getTextContent();                                81
    try{                                                                            82
      this.deregistration(externalid);                                              83
    }catch(TransformerException te){                                                84
      if (ApplicationData.debug())te.printStackTrace(System.err);                   85
      log("CED␣Composite␣Event␣Deregistration␣Servlet:␣Can␣not␣deregistration␣Event␣  86
          Expression␣" + te);
      //Creats a SOAP−Fault−Message because the arrived Message is not correct      87
```

```
            return this.createFaultMessage();                                    88
        }catch(IOException ioe){                                                 89
            if (ApplicationData.debug())ioe.printStackTrace(System.err);         90
            log("CED Composite Event Deregistration Servlet: Can not write deregitration to File" +  91
                ioe);
            //Creats a SOAP-Fault-Message because the arrived Message is not correct  92
            return this.createFaultMessage();                                    93
        }                                                                        94
    }//ende for all Events in the SOAP-Message                                   95
                                                                                 96
    //Returns a Result Message with the ID's of the deregistratet Expressions   97
    return result;                                                               98
}//end Method onMessage(SOAPMessage)                                           99
```

Listing 7.11: Methode onMessage der Klasse DeregisterEventExpressionServlet

Obigem Listing kann man entnehmen, dass aus dem Header der SOAP-Nachricht die <rule−id>-Elemente ausgelesen werden. Der Textinhalt dieser Elemente wird als ID einer ECA-Regel aufgefasst. Mit dieser ID wird die Methode „deregistration()“ aufgerufen. Sie löscht dann die zu der ID gehörenden Automaten.

```
private void deregistration(String ruleid) throws TransformerException, IOException{  106
    Vector<String> eventids = ApplicationData.loadEventIDs(ruleid);                    107
    if( eventids != null){                                                             108
        for (int i = 0; i < eventids.size(); i++){                                     109
            String eventid = eventids.elementAt(i);                                    110
            //Deleting the ID's from the Config-Files.                                 111
            ApplicationData.deleteRuleID(eventid);                                     112
            //Remove from Automatons and Instances.                                    113
            ApplicationData.getAutomatons().removeEventExpression(eventid);            114
            //Save the new Situation.                                                  115
            ApplicationData.getAutomatons().serialize();                               116
        }                                                                              117
    }                                                                                  118
}//end Method deregistration(String)                                                 119
```

Listing 7.12: Methode deregistration aus DeregisterEventExpressionServlet

Zuerst wird in dieser Methode die Liste der internen Automaten-ID‘s ausgelesen, die zu der angegebenen ECA-Regel gehören (Zeile 107). Für jedes dieser Elemente werden die entsprechenden Einträge aus den Konfigurations-Dateien entfernt (Zeile 112). Anschließend wird der Automat zur eben bestimmten ID, seine Sub-Automaten und die Instanzen dieser Automaten aus dem XML-Dokument entfernt (Zeile 114). Zum Abschluss wird das so manipulierte Dokument wieder gespeichert (Zeile 116).

Anhang A

Installation & Anwendung

In diesem Anhang sind alle Informationen zusammengetragen, die nötig sind, um die Java-Programme und XSLT-Stylesheets, die im Verlauf dieser Arbeit entstanden sind, in Betrieb zu nehmen. Das Augenmerk liegt dabei auf der Installation der Java-Komponenten. Aber auch die eigenständige Anwendung der Stylesheets wird kurz erläutert.

A.1 Verwendung der XSLT-Stylesheets

Die XSLT-Stylesheets aus dieser Arbeit können nicht nur durch die Java-Servlets verwendet werden, sondern sind auch eigenständig lauffähig. Dazu kann z.B. der XSLT-Prozessor SAXON verwendet werden. Während der Entwicklung der Stylesheets wurde die Version 8.4 verwendet.

Erzeugen der Automaten

Um die Erzeugung der Automaten aus einem Ausdruck zu starten, muss das Stylesheet „create_automaton.xsl“ aufgerufen werden. Das Stylesheet „create_any-operator-automaton“ wird automatisch importiert. Dazu muss die Datei allerdings im selben Verzeichnis liegen.

Standardmäßig ist der Basisname der Automaten „AutomatonX“. Um den Namen an die eigenen Bedürfnisse anzupassen, ist der Parameter „AUTOMATON_NAME“ zu setzen.

Unter Windows XP bei Verwendung von SAXON sieht der gesamte Befehl etwa so aus:

```
java -jar %SAXON%saxon8.jar -novw ausdruck.xml ↩
create_automaton.xsl AUTOMATON_NAME=name
```

Dabei ist „ausdruck.xml“ die Datei mit dem Snoop-Ausdruck und %SAXON% enthält den Pfad zu „saxon8.jar“.

Transformieren der Automaten

Die vier Dateien, die gemeinsam die Transformation durchführen, sind „ceda-algorithm.xsl“, „ceda-io.xsl“, „ceda-event.xsl“, und „ceda-instance.xsl“. Diese Dateien müssen im gleichen Verzeichnis liegen. Dann kann durch den Aufruf von „ceda-io.xsl“ die Umwandlung gestartet werden. Die drei anderen Stylesheets werden dann importiert. Damit das Stylesheet die Datei mit dem Event findet, muss „event.xml“ im selben Verzeichnis vorhanden sein. Parallel zur XML-Datei mit den Automaten, muss die DTD „ceda.dtd“ vorliegen.

Beispielhaft sei hier wieder der Aufruf für die Datei „automaten.xml“ mit Saxon unter Windows XP genannt:

```
java -jar %SAXON%saxon8.jar -novw automaten.xml ceda-io.xsl
```

A.2 Installation der Java-Umgebung

Um die Java-Servlets verwenden zu können, benötigt man eine Java-Laufzeitumgebung der Version 5 und einen passenden Servlet-Container. Der Autor verwendet hierzu Tomcat in der Version 5.5. In das Verzeichnis „webapps“ aus dem Installationsverzeichnis von Tomcat kann das Archiv „ced.war“ kopiert werden. Damit sind die Servlets einsatzfähig. Das Archiv ist auf der CD-Rom im Verzeichnis „Web-Application“ enthalten.

Um ein Archiv aus den eventuell geänderten Codes selbst zu erstellen, ist eine feste Verzeichnisstruktur zu errichten. Die Struktur ist dieselbe, die durch die Servletspezifikation vorgesehen wird. Einige Verzeichnisse und Dateien wurden allerdings zusätzlich aufgenommen. Das Wurzelverzeichnis der Web-Application enthält zwei Verzeichnisse: XML und WEB-INF. In XML müssen ein XML-Dokument im Automatenformat mit Namen automaton.xml und die beiden DTD's zu Event-Ausdrücken und -Automaten enthalten sein. WEB-INF enthält alle XSLT-Dateien, die Java-Klassen, Bibliotheken und Konfigurationsdateien. Die Stylesheets werden im Ordner XSLT untergebracht. Die Konfigurationsdateien rulemap.xml und urlmap.xml enthalten die Mappings der Automaten-ID auf die der ECA-Regel bez. die URL der zugehörigen ECA-Engine. Diese Dateien müssen im Verzeichnis conf liegen. Die Klassen samt ihrer Packetstruktur liegen in classes und die Bibliotheken in lib. In der Datei web.xml kann die Web-Applikation konfiguriert werden. Dort können die URL's angepasst werden, unter denen die Servlets erreichbar sind.

Diese Anordnung an Dateien kann entweder direkt im Servlet-Container eingesetzt werden oder in ein JAR-Archiv mit der Endung „.war“ verpackt werden.

Die verwendeten Bibliotheken stammen aus verschiedenen Quellen. Die Bibliotheken zu den API's SAAJ, JAXP und JAXM gehören zum erweiterten Umfang von Java und werden von

Sun bereitgestellt[1]. Aus der Enterprise Edition von Java stammen die Bibliotheken mail.jar und activation.jar, die Netzwerkfunktionen bereitstellen. JDOM ist auf der Webseite www.jdom.org erhältlich. Der XSLT-Prozessor Saxon, bestehend aus einer Reihe von JAR-Archiven, ist unter saxon.sourceforge.net zu finden. Hier werden die beiden Archive saxon8.jar und saxon8-dom.jar verwendet.

Sämtliche besprochenen Dateien befinden sich bereits in der benötigten Struktur auf der beiliegenden CD-Rom im Verzeichnis „Web-Application/ced“. Die Quellcodes der Java-Klassen hingegen liegen in „Sources“.

Die folgende Übersicht gibt die Struktur wieder, in der die Dateien für die Web-Application angeordnet sein müssen:

```
ced
  WEB-INF
    classes
      rewerse
        ced
          servlets
            CEDServlet.class
            DeregisterEventexpressionServlet.class
            ReceiveAtomicEventServlet.class
            RegisterEventExpressionServlet.class
          ApplicationData.class
          Automatons.class
          EventConsumer.class
    lib
      activation.jar
      jaxm-api.jar
      jaxm-runtime.jar
      jdom.jar
      mail.jar
      saaj-1_2-fr-api.jar
      saaj-impl.jar
      saxon8.jar
      saxon8-dom.jar
    conf
      rulemap.xml
      urlmap.xml
    XSLT
      create_automaton.xsl
      create_any-operator-automaton.xsl
      ceda-io.xsl
      ceda-instance.xsl
      ceda-event.xsl
      ceda-algorithm.xsl
```

[1] URL: http://www.java.sun.com

```
        ced-organisation.xsl
    web.xml
  XML
    automatons.xml
    ceda.dtd
    snoop.dtd
  index.html
```

A.3 Verwendung der Java-Umgebung

Für den Einsatz der Web-Applikation sind drei URL's von Bedeutung. „ced/speci“ ermöglicht es neue Event-Spezifikationen registrieren zu lassen. Mittels „ced/delete“ lassen sie sich wieder entfernen. Um neue atomare Events an die Servlets zu schicken, steht der Pfad „ced/event“ zur Verfügung. Diesen drei Angaben ist noch die URL des Tomcat-Servers voranzustellen, unter dem die Servlets erreichbar sind. Die gesamte URL könnte dann beispielsweise so aussehen:

`http://localhost:8080/ced/event`

Die SOAP-Nachrichten, die an diese URL's gesandt werden müssen, werden in Kapitel 7 vorgestellt.

Die gewählten URL'S können jederzeit in der Datei „web.xml“ angepasst werden.

Unter „ced/index.html“ findet sich eine rudimentäre HTML-Seite. Diese bietet Zugriff auf die XML-Datei mit den Automaten und ihren Instanzen. Der aktuell Stand der Detektion lässt sich so einsehen.

Anhang B

Anpassen der Kommunikationsschnittstellen

Als diese Arbeit bereits weit fortgeschritten war, wurde die Kommunikationsform innerhalb des ECA-Frameworks geändert. Die einzelnen Komponenten kommunizieren nun nicht mehr über SOAP-Nachrichten, sondern mittels des HTTP-Protokolls. Dadurch soll eine bessere Performance erreicht werden. Diese Änderung in der Software zu diesem Buch nachzuvollziehen, war leider zeitlich nicht mehr möglich. Daher soll in diesem Anhang kurz dargelegt werden, wo Änderungen am Quellcode nötig sind, um die Netzwerkkommunikation anzupassen.

Für das Senden und Empfangen von Nachrichten in und aus dem Netzwerk sind die Servlets zuständig. Servlets gibt es für verschiedene Protokolle. In dieser Arbeit werden solche für SOAP verwendet. Diese Servlets leiten sich von JAXMServlet ab. JAXMServlet ist eine Klasse aus dem Java-API JAXM der Firma Sun. Mit Hilfe dieser Klasse werden die SOAP-Nachrichten in Java-Objekte umgewandelt. Diese können dann weiterverarbeitet werden. Die verwendeten Servlets führen für jede eintreffende Nachricht eine Methode „onMessage()“ aus.

Von JAXMServlet ist die abstrakte Klasse CEDServlet abgeleitet. Von dieser wiederum erben alle weiteren Servlets der Snoop-Einheit. Um die Kommunikationsform zu ändern, muss CEDServlet von einer anderen Klasse erben. Dazu muss in der Deklaration von CEDServlet eine passende Klasse für die Vererbung gewählt werden. Die Klasse von der CEDServlet erbt, wird hier als „Kom-Servlet“ bezeichnet. Zusätzlich ist das Interface „ReqRespListener“ zu entfernen. Dieses Interface ist nur im Zusammenhang mit SOAP von Interesse.

Im Allgemeinen ruft jedes Servlet eine Methode auf, sobald eine Nachricht auf dem unterstützten Protokoll eintrifft. Diese Methode ist bei DeregisterEventExpressionServlet, ReceiveAtomicEventServlet und RegisterEventExpressionServlet onMessage(). Diese Methode enthält die SOAP-Nachricht als Parameter. In jeder dieser drei Klassen wird diese Nachricht ausgewertet und mit

den ihn ihr enthaltenen Daten weitere Methoden aufgerufen. Diese Methoden sind unabhängig von der Kommunikationsschnittstelle. Da onMessage() speziell für SOAP-Nachrichten gedacht ist, muss sie durch eine andere Methode ersetzt werden. Welche das ist, ist von dem Kom-Servlet abhängig, das eingesetzt wird. Durch diese Methode müssen in jedem Fall einige weitere Methoden aufgerufen werden, damit die Servlets ihre Aufgabe erfüllen können. Im Fall von DeregisterEventExpressionServlet ist dies deregistration(String ruleid). Bei RegisterEventExpressionServlet heißt die entsprechende Methode register(Element eventexpression, String id, String ruleid, String answerurl, Iterator variabletuples). Für ReceiveAtomicEventServlet müssen zwei Methoden aufgerufen werden: parseEvent(Element event) und consume(). Die Werte für die Parameter werden aus der eingegangenen Nachricht bestimmt.

Die Änderungen beschränken sich also auf das Anpassen der Vererbung in CEDServlet und das Ersetzen der onMessage()-Methoden in den anderen Klassen aus dem Packet „rewerse.ced.servlets“.

Anhang C

XSLT-Stylesheets

C.1 Verarbeitung der Automaten

C.1.1 Stylesheet „ceda-io“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<!-- Dieses Stylesheet ist ein Modul für den Algorithmus der ein atomares Events auf die "Composite Event
     Detection Automatons" anwendet. -->
<!-- Dieses Modul ist zustaendig fuer die Ein- und Ausgabe des Algorithmus. Hier werden die globalen Variablen
     zur verfuegung gestellt, die anderen Module importiert die Formatierung der Ausgabe festgelegt sowie die
     Eingabe- und Ausgabedaten aufbereitet. -->
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:ceda= "http://rewerse.net/
     event-automaton-ml" version= "1.0" >
<xsl:strip-space elements= "*" />
<!-- Laden der Templates aus den anderen Modulen. -->
<xsl:include href= "ceda-algorithm.xsl" />
<xsl:include href= "ceda-event.xsl" />
<xsl:include href= "ceda-instance.xsl" />
<!-- Output spezifiziert die zu erzeugenden Ausgabe. In diesem Fall ein XML-Dokument der Version 1.0 in
     UTF-8-Kodierung mit Einrückungen, einer XML-Deklaration, dem Medien-Typ "text/xml" und der DTD
     ceda.dtd. -->
<xsl:output method= "xml" version= "1.0"
            encoding= "UTF-8" omit-xml-declaration= "no"
            standalone= "yes" media-type= "text/xml"
            indent= "yes" cdata-section-elements= ""
            doctype-system= "./ceda.dtd" />
<!-- Pfad der Datei aus der das Event gelesen werden soll. -->
<xsl:variable name= "EVENTFILE" select= "'Event.xml'" />
<!-- Variable die den Zugriff auf die Eingabedatei sicherstellt. -->
<xsl:variable name= "INPUTFILE" select= "/" />
<!-- Template das Überprüft ob der Inhalt des Ausgansdokument ein CEDA-Dokument ist. Wenn nicht wird eine
     entsprechende Meldung ausgegeben und die Bearbeitung abgebrochen. -->
<xsl:template match= "/">
   <xsl:choose>
   <xsl:when test= "/ceda:ceda">
      <xsl:apply-templates select= "ceda:ceda" />
   </xsl:when>
   <xsl:otherwise>
      <xsl:message terminate= "yes">
         Das Event-Detection-Stylesheet wurde nicht auf Automatendeclarationen für Composite Events angewannt.
      </xsl:message>
   </xsl:otherwise>
   </xsl:choose>
</xsl:template>
<!-- Dieses Template verarbeitet das ceda Element: Es erzeugt die erste Situation, liest die Datei mit dem Event
     ein startet den eigentlichen Algorithmus und bereitet die letzte Situation für die Ausgabe auf. -->
<xsl:template match= "/ceda:ceda">
   <!-- Erzeugt die Ausgangssituation mit dem Event und speichert sie in einer Variablen. -->
   <!-- Variable mit der Ausgangssituation. -->
   <xsl:variable name= "CURRENT_SITUATION" >
      <!-- Instanzen aus der Eingabe für die interne Darstellung aufbereiten. -->
      <xsl:apply-templates select= "./ceda:instance" /><!--Modul Instanz-->
      <!-- Das Primitv-Event aus der Übergabedatei in die aktuelle Situation kopieren. -->
      <xsl:call-template name= "createPrimitiveEvent" ><!--Modul Event-->
```

```
            <xsl:with-param name= "EVENT" select= "document($EVENTFILE)/child::*" />
        </xsl:call-template>
    </xsl:variable><!-- End Variable CURRENT_SITUATION -->
    <!-- Den Algorithmus zum Detektieren der Composite-Events anstossen. -->
    <!-- Variable mit dem Ergebnis des Algorithmus. - Der letzen Situation. -->
    <xsl:variable name= "NEW_SITUATION" >
        <!-- Aufruf der rekursiven Verarbeitung des Events. -->
        <xsl:call-template name= "rekursivStep" ><!--Modul Algorithm-->
            <xsl:with-param name= "OLD_SITUATION" select= "$CURRENT_SITUATION" />
            <xsl:with-param name= "COUNTER" select= "'0'" />
        </xsl:call-template>
    </xsl:variable>
    <!-- Hier wird die Ausgabe erzeugt. -->
    <!-- Anlegen des ceda-Elements des Ausgabedokuments. -->
    <ceda:ceda>
        <!--Kopieren der urspruenglichen Attribute in das neue ceda-Element.-->
        <xsl:copy-of select= "@*" />
        <!--Kopieren der Automatenspezifikationen, da sie unveränderlich sind.-->
        <xsl:copy-of select= "ceda:automaton" />
        <!-- Kopieren der stabilen Statie aus der neuen Situation in die Ausgabe. -->
        <xsl:copy-of select= "$NEW_SITUATION/instancewrapper[./@stable = 'yes']/ceda:instance" />
        <!-- Kopiert die detektierten Events in die Ausgabe. -->
        <xsl:copy-of select= "$NEW_SITUATION/eventwrapper[@typ='composite']/ceda:instance" />
    </ceda:ceda>
</xsl:template>
</xsl:stylesheet>
```

C.1.2 Modul „ceda-algorithm“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<!-- Dieses Stylesheet ist ein Modul für den Algorithmus der ein atomares Events auf die "Composite Event
     Detection Automatons" anwendet. -->
<!-- Dieses Modul ist zuständig für die Transformation der einzelnen Automateninstanzen in neue Zustände
     aufgrund eines atomaren Evnets. -->
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:ceda= "http://rewerse.net/
     event-automaton-ml" xmlns:eca= "http://www.eca.org/eca-ml" version= "1.0">
<!-- Dieses Template überprüft ob ein Status auf einen Zustand verweist der vom Typ 'detect' ist. Damit kann
     erkannt werden ob die geänderte Instanz ein Composite Event repräsentiert.
     @param NEW_INSTANCE Die Instanz die auf ein Event gepfüft werden soll.
     @return Das eingegebene Event wird in einen 'eventwrapper'-Umschlag verpackt wenn der zugehörige Zustand
          vom Typ 'detect' ist. Anderenfalls wird der Status selbst wieder ausgegeben. -->
<xsl:template name= "checkIfNewInstanceIsEvent" >
  <xsl:param name= "NEW_INSTANCE" />
  <!--Fallunterscheidung ob der Zustand der Instanz ein Event auslöst.-->
  <xsl:choose>
  <!--Die Automateninstanz repräsentiert ein Event.-->
  <xsl:when test= "$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[@id = $NEW_INSTANCE/instancewrapper
       /ceda:instance/ceda:current-state/@ref][@detect = 'yes']" >
    <!--Erzeugen der Repräsentation des Events für die weitere Verarbeitung-->
    <xsl:call-template name= "createCompositeEvent" ><!--Modul Event-->
      <xsl:with-param name= "INSTANCE" select= "$NEW_INSTANCE/instancewrapper/ceda:instance" />
      <xsl:with-param name= "STEP" select= "$NEW_INSTANCE/instancewrapper/@stepcounter" />
    </xsl:call-template>
  </xsl:when>
  <!--Die Automateninstanz repräsentiert kein Event.-->
  <xsl:otherwise>
    <xsl:copy-of select= "$NEW_INSTANCE" />
  </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<!-- Dieses Template startet die Transformation einer bestehenden Situation in eine Neue. Dabei wird dieses
     Template mehrfach rekursiv aufgerufen bis das Endergebnis erreicht ist.
     @param OLD_SITUATION Die (temporaure) Situation in der der die Automaten sich zur Zeit befinden.
     @param COUNTER Der Parameter gibt an wie oft dieses Template bereits rekursiv aufgerufen wurde.
     @return Eine neue (temporäre) Situation der Automaten die sich aus der Verarbeitung der in
          OLD_SITUATION gespeicherten ergibt. -->
<xsl:template name= "rekursivStep">
  <xsl:param name= "OLD_SITUATION" />
  <xsl:param name= "COUNTER" select= "'0'" />
  <!-- Variable mit der neuen, zu erstellenden Situation. -->
  <xsl:variable name= "NEW_SITUATION" >
    <!-- Wähle alle Statie der alten Situation aus und durchläuft für jeden die Suche nach anwendbaren Events.
         -->
    <xsl:for-each select= "$OLD_SITUATION/instancewrapper">
      <!-- Variable die den aktuelle betrachteten Status speichert. -->
      <xsl:variable name= "CURRENT_INSTANCE" select= "." />
      <!-- Fallunterscheidung 1 für die Verwendbarkeit einer Instanz. Eine Instanz ist verwendbar wenn er
           entweder zu einem Automaten mit zeitlicher Bedingung 'kleinergleich' oder 'non' gehört oder die
           Bedingung 'kleiner' und der Zustand zum aktuellen Zeitpunkt unverändert (@stepcounter gleich 0) ist.
           -->
      <xsl:choose>
      <!-- Fall 1.a: Die Instanz ist für die weitere Verarbeitung verwendbar. -->
      <xsl:when test= "$INPUTFILE/ceda:ceda/ceda:automaton[./@id=$CURRENT_INSTANCE/ceda:instance/
           @automaton][(./@time-constraint='less' and $CURRENT_INSTANCE/@stepcounter='0') or ./
           @time-constraint='equalorless' or ./@time-constraint='non']/ceda:state[@id = $
           CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref][./@typ != 'ende' and ./@typ != '
           abort']" >
        <!--Diese Variable enthält für jedes Event (teilweise Mehrmals) das auf eine Transition anwendbar ist.
             -->
        <xsl:variable name= "USABLE_EVENTS" >
          <!-- Jede Transition des Zustands des aktuell betrachteten Status betrachten -->
          <xsl:for-each select= "$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
               CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/ceda:transition" >
            <xsl:variable name= "CURRENT_TRANSITION" select= "." />
            <!-- Fallunterscheidung 1.1.3: Nach primitiven und kombinierten Events. -->
            <xsl:choose>
            <!-- Fall 1.1.3.a: Primitives Event. -->
            <xsl:when test= "./@typ = 'primitiv'" >
              <!-- Anfang des Vergleichs des Events mit dem Muster im Automat -->
              <xsl:for-each select= "$OLD_SITUATION/eventwrapper[./@typ = 'primitiv'][./@stepcounter
                   = $COUNTER]" >
                <xsl:variable name= "EQUALITYTESTRESULT" >
                  <!-- Dieses Template führt den Vergleich zwichen den atomaren Events und
                       Eventpattern durch.-->
                  <xsl:call-template name= "equalityTest" >
                    <xsl:with-param name= "EVENT" select= "./*[1]" />
                    <xsl:with-param name= "PATTERN" select= "$CURRENT_TRANSITION/child::
```

```
                        *[1]" />
                    <xsl:with-param name= "VARIABLES" select= "$CURRENT_INSTANCE/
                        ceda:instance/ceda:variables/child::eca:variable" />
                </xsl:call-template>
            </xsl:variable>
            <!-- Falls eine Übereinstimmung vorliegt: -->
            <xsl:if test= "$EQUALITYTESTRESULT/text() = 'TRUE'" >
                <!-- Kopieren des Events das auf die aktuell betrachtete Instanz anwendbar ist. -->
                <xsl:copy-of select= "." />
            </xsl:if>
        </xsl:for-each>
    </xsl:when>
    <!-- Fall 1.1.3.b: Zusammengesetztes Event. -->
    <xsl:when test= "./@typ = 'composite'" >
        <!-- Jedes composite Event das zu dem Auslöser der gerade betrachteten Transition passt
            kopieren -->
        <xsl:for-each select= "$OLD_SITUATION/eventwrapper[./@typ = 'composite'][./@stepcounter
            = $COUNTER][./ceda:instance/@automaton = $CURRENT_TRANSITION/
            @subautomaton]" >
            <xsl:variable name="VARIABLETESTRESULT" >
                <xsl:call-template name="variableTest" >
                    <xsl:with-param name="INSTANCE" select="$CURRENT_INSTANCE" />
                    <xsl:with-param name="EVENT" select="." />
                </xsl:call-template>
            </xsl:variable>
            <xsl:if test="$VARIABLETESTRESULT/text()='TRUE'">
                <xsl:copy-of select="." />
            </xsl:if>
        </xsl:for-each>
    </xsl:when>
    </xsl:choose>
</xsl:for-each>
</xsl:variable><!--Ende USABLE_EVENTS -->
<!-- Fallunterscheidung 1.1.4 ob es anwendbare Events gibt. -->
<xsl:choose>
<!-- Fall 1.1.4.a: Es gibt anwendbare Events.-->
<xsl:when test= "count($USABLE_EVENTS/*) > 0" >
    <xsl:variable name="NEW_INSTANCES" >
    <!-- Die Transitionen des aktuellen Zustands der betrachteten Instanz anfassen. -->
    <xsl:apply-templates select="$INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
        CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/ceda:transition" mode="
        transformation">
        <xsl:with-param name="USABLE_EVENTS" select="$USABLE_EVENTS" />
        <xsl:with-param name="CURRENT_INSTANCE" select="$CURRENT_INSTANCE" />
        <xsl:with-param name="COUNTER" select="$COUNTER" />
    </xsl:apply-templates>
    </xsl:variable>
    <!-- Fallunterscheidung 1.1.2.2. zur Duplizierung alter Instanzen. -->
    <xsl:choose>
    <!-- Fall 1.1.2.2.a: Es soll dupliziert werden. -->
    <xsl:when test= "($INPUTFILE/ceda:ceda/ceda:automaton/ceda:state[./@id = $
        CURRENT_INSTANCE/ceda:instance/ceda:current-state/@ref]/@duplicate='yes') and (not( $
        NEW_INSTANCES/instancewrapper/ceda:instance/ceda:current-state[./@ref=$INPUTFILE/
        ceda:ceda/ceda:automaton/ceda:state[./@typ='abort']/@id]))" >
        <xsl:copy-of select= "$CURRENT_INSTANCE" />
    </xsl:when><!-- Ende Fall 1.1.2.2.a -->
    <!--Fall 1.1.2.2.b: Es darf nicht dupliziert werden -->
    <xsl:otherwise>
        <xsl:call-template name= "createInstableCopy" >
            <xsl:with-param name= "OLD_INSTANCE" select= "$CURRENT_INSTANCE" />
        </xsl:call-template>
    </xsl:otherwise>
    </xsl:choose><!--Ende Fallunterscheidung 1.1.2.2-->
    <xsl:copy-of select="$NEW_INSTANCES/*" />
</xsl:when><!-- Ende Fall 1.1.4.a -->
<!-- Fall 1.1.4.b: Es gibt kein Event das auf die betrachtete Instanz anwendbar ist. -->
<xsl:otherwise>
    <!-- Kopieren des unveränderten Zustands in den Output dieses Schrittes -->
    <xsl:copy-of select= "$CURRENT_INSTANCE" />
</xsl:otherwise>
</xsl:choose><!-- Ende Fallunterscheidung 1.1.4-->
</xsl:when><!-- Ende Fall 1.a -->
<!-- Fall 1.b: Die Instanz ist für die weitere Verarbeitung nicht verwendbar. -->
<xsl:otherwise>
    <!-- Kopieren des Unveränderten Zustands in den Output dieses Schrittes -->
    <xsl:copy-of select= "$CURRENT_INSTANCE" />
</xsl:otherwise>
</xsl:choose><!-- Ende Fallunterscheidung 1 -->
</xsl:for-each><!-- Ende der Schleife durch alle Instanz -->
<!-- Die alten Events der neuen Situation hinzufügen -->
```

```
      <xsl:copy-of select= "$OLD_SITUATION/eventwrapper" />
    </xsl:variable><!-- Ende der Variablen "NEW_SITUATION" -->
    <!-- Rekursiver Aufruf der Verarbeitung -->
    <!-- Fallunterscheidung 2: Muss ein weiterer Rekursionsaufruf durchgeführt werden? -->
    <xsl:choose>
    <!-- Fall 2.1.a: Falls sich keine Änderungen ergeben haben die Rekursion abbrechen. -->
    <xsl:when test= "count($NEW_SITUATION/child::*) = count($OLD_SITUATION/child::*)" >
      <!-- Neue Situation ausgeben. -->
      <xsl:copy-of select= "$NEW_SITUATION" />
    </xsl:when>
    <!-- Fall 2.1.b: Es gab Änderungen im aktuellen Schritt. Daher wird noch ein Rekusionsschritt durchgeführt.
        -->
    <xsl:otherwise>
      <xsl:call-template name= "rekursivStep" >
        <xsl:with-param name= "COUNTER" select= "$COUNTER + 1" />
        <xsl:with-param name= "OLD_SITUATION" select= "$NEW_SITUATION" />
      </xsl:call-template>
    </xsl:otherwise>
    </xsl:choose><!-- Ende Fallunterscheidung 2 -->
</xsl:template><!-- Ende Template rekursivStep -->
<!--Dieses Template führt die Zustandsübergänge für eine Automateninstanz basierend auf einer Menge von Events
      durch.
    @param USABLE_EVENTS Die Menge der Events die auf die Instanz angewandt werden sollen.
    @param CURRENT_INSTANCE Die Instanz zu der Zustandsübergänge durchgeführt werden sollen.
    @param COUNTER Der Rekursionsschritt aus dem diesese Template aufgerufen wird.-->
<xsl:template match="ceda:transition" mode="transformation" >
    <xsl:param name="USABLE_EVENTS" />
    <xsl:param name="CURRENT_INSTANCE" />
    <xsl:param name="COUNTER" />
    <xsl:variable name= "CURRENT_TRANSITION" select= "." />
    <!-- Fallunterscheidung 1: Ist der Auslöser der Transition primitiv oder composite. -->
    <xsl:choose>
    <!-- Fall 1.a: Primitives Events als Auslöser einer Transition. -->
    <xsl:when test= "./@typ = 'primitiv'" >
      <!-- Anfang des Vergleichs des Events mit dem Muster im Automat. -->
      <xsl:for-each select= "$USABLE_EVENTS/eventwrapper[./@typ = 'primitiv']">
        <!-- Diese Variable nimmt das Ergebnis des Vergleichs auf. -->
        <xsl:variable name= "EQUALITYTESTRESULT" >
          <!-- Dieses Template führt den Vergleich zwischen den atomaren Events und Eventpattern durch.-->
          <xsl:call-template name= "equalityTest" >
            <xsl:with-param name= "EVENT" select= "./*[1]" />
            <xsl:with-param name= "PATTERN" select= "$CURRENT_TRANSITION/child::*[1]" />
            <xsl:with-param name= "VARIABLES" select= "$CURRENT_INSTANCE/ceda:instance/
                ceda:variables/child::eca:variable" />
          </xsl:call-template>
        </xsl:variable>
        <!-- Falls eine Übereinstimmung vorliegt: -->
        <xsl:if test= "$EQUALITYTESTRESULT/text() = 'TRUE'" >
          <!-- Überprüft, ob der neue Zustand zu einem Event führt. -->
          <xsl:call-template name= "checkIfNewInstanceIsEvent">
            <xsl:with-param name= "NEW_INSTANCE">
              <!--Dieser Aufruf erzeugt eine neue Instanz mit dem geänderten Zustand.-->
              <xsl:call-template name= "createNewInstanceWithAtomicEvent" >
                <xsl:with-param name= "OLD_INSTANCE" select= "$CURRENT_INSTANCE" />
                <xsl:with-param name= "EVENT" select= "." />
                <xsl:with-param name= "TARGET" select= "$CURRENT_TRANSITION/@target" />
                <xsl:with-param name= "STEP" select= "$COUNTER + 1" />
                <xsl:with-param name= "NEW_VARIABLES" select= "$EQUALITYTESTRESULT/
                    eca:variable" />
              </xsl:call-template>
            </xsl:with-param>
          </xsl:call-template>
        </xsl:if>
      </xsl:for-each>
    </xsl:when><!-- Ende Fall 1.a -->
    <!-- Fall 1.b: Eine Transition mit Composite Event als Auslöser. -->
    <xsl:when test= "./@typ = 'composite'" >
      <!-- Vergleich jedes neuen kombinierten Events mit jeder Transition der betrachteten Instanz.-->
      <xsl:for-each select= "$USABLE_EVENTS/eventwrapper[./@typ = 'composite']">
        <xsl:variable name="VARIABLETESTRESULT" >
          <xsl:call-template name="variableTest" >
            <xsl:with-param name="INSTANCE" select="$CURRENT_INSTANCE" />
            <xsl:with-param name="EVENT" select="." />
          </xsl:call-template>
        </xsl:variable>
        <xsl:if test="$VARIABLETESTRESULT/text()='TRUE'">
          <!-- Überprüft, ob der neue Zustand zu einem Event führt. -->
          <xsl:call-template name= "checkIfNewInstanceIsEvent">
            <xsl:with-param name= "NEW_INSTANCE">
```

```
                <!--Dieser Aufruf erzeugt eine neue Instanz mit dem geänderten Zustand.-->
                <xsl:call-template name= "createNewInstanceWithCompositeEvent" >
                    <xsl:with-param name= "OLD_INSTANCE" select= "$CURRENT_INSTANCE" />
                    <xsl:with-param name= "EVENT" select= "." />
                    <xsl:with-param name= "TARGET" select= "$CURRENT_TRANSITION/@target" />
                    <xsl:with-param name= "STEP" select= "$COUNTER + 1" />
                </xsl:call-template>
            </xsl:with-param>
        </xsl:call-template>
    </xsl:if>
  </xsl:for-each>
 </xsl:when><!-- Ende Fall 1.b -->
 </xsl:choose><!-- Ende Fallunterscheidung 1 -->
</xsl:template>
</xsl:stylesheet>
```

C.1.3 Modul „ceda-event“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "yes"?>
<!-- Dieses Stylesheet ist ein Modul für den Algorithmus der ein atomaren Events auf die "Composite Event Detection Automatons" anwendet. -->
<!-- Dieses Modul ist zuständig für die Verarbeitung der Events. Es besitzt Templates um sowohl atomare als auch zusammengesetzte Events in eine für den Algorithmus verwertbare Darstellung zu bringen. Dazu kommen Template die zusammen den Vergleich zwischen Events und Event-Mustern realisieren. -->
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:ceda= "http://rewerse.net/
    event-automaton-ml" xmlns:eca= "http://www.eca.org/eca-ml" version= "1.0">
<!-- Dieses Template erzeugt die interne Darstellung eines atomaren Events. Dazu wird das Event, das über den Parameter EVENT übergeben wird in einen Event-Umschlag verpackt. Dieser erhält zusätzlich zwei Attribute. Zum einen, um es im Verlaufe des Algorithmus von den composite Events zu unterscheiden, die Information das es sich um ein atomares Event handelt und zum Anderen einen Stepcounter von 0. Dieser ist immer Null da nur zu Begin atomare Events in den Algorithmus eingehen können.
    @param EVENT Ein XML-Fragment das im Verlaufe des Algorithmus als Event betrachtet werden soll.
    @return Ein für den Algorithmus als Primitiv-Event erkennbares XML-Fragment. Beginnend mit einem "primitivevent"-Tag. -->
<xsl:template name= "createPrimitiveEvent" >
  <xsl:param name= "EVENT" />
  <!-- Anlegen des Umschlags für das eigentliche Event. -->
  <xsl:element name= "eventwrapper">
    <xsl:attribute name= "typ" >primitiv</xsl:attribute>
    <xsl:attribute name= "stepcounter">0</xsl:attribute>
    <!-- Kopieren des Events in den Umschlag. -->
    <xsl:copy-of select= "$EVENT" />
  </xsl:element>
</xsl:template>
<!-- Template zur Erzeugung der internen Darstellung eines composite Events. Dazu wird das Event das als Automateninstanz (dieser sollte natürlich in einem 'Detect'-Zustand sein) in dem Parameter INSTANCE übergeben wird in einen Event-Umschlag verpackt. Dieser erhält zusätzlich zwei Attribute. Zum einen, um es im Verlaufe des Algorithmus von den atomaren Events zu unterscheiden, die Information das es sich um ein composite Event handelt und zum anderen einen Stepcounter. Dieser wird auf den Wert des Parameters STEP gesetzt.
    @param INSTANCE Das <instance>-Element der Automateninstanz die einen 'Detect'-Zustand erreicht hat.
    @param STEP Der Rekursionsschritt in dem die Instanz den 'Detect'-Zustand erreicht hat.
    @return Ein für den Algorithmus als Composite-Event erkennbares XML-Fragment. Beginnend mit einem "compositeevent"-Tag. -->
<xsl:template name= "createCompositeEvent" >
  <xsl:param name= "INSTANCE" />
  <xsl:param name= "STEP" />
  <!-- Anlegen des Umschlags für das eigentliche Event. -->
  <xsl:element name= "eventwrapper" >
    <xsl:attribute name= "typ" >composite</xsl:attribute>
    <xsl:attribute name= "stepcounter" >
      <xsl:value-of select= "$STEP" />
    </xsl:attribute>
    <xsl:copy-of select= "$INSTANCE" />
  </xsl:element>
</xsl:template>
<!-- Template zum Vergleich eines atomaren Events mit einem Eventpattern. Zu diesem Zweck prüft das Template Namen, Namespace und Attribute des Wurzelements des Musters sowie rekursiv dessen Kindelemente ob diese auch in dem Event vorkommen.
    @param EVENT Das atomare Event das für den Vergleich herangezogen werden soll.
    @param PATTERN Das Eventpatern das für den Vergleich herangezogen werden soll.
    @param VARIABLES Die bereits vorhandenen Variablenbindungen. Der Wert ist als Menge von <Variable>-Tags anzugeben.
    @return TRUE und eine Menge von Variablenbindungen in Form von <Variable>-Tags falls der Vergleich eine Übereinstimmung ergibt. Nichts oder FALSE wenn keine Übereinstimmung gefunden wurde. -->
<xsl:template name= "equalityTest" >
  <xsl:param name= "EVENT" />
  <xsl:param name= "PATTERN" />
  <xsl:param name= "VARIABLES" />
  <!-- Variable, die für jede Übereinstimmung die nicht erreicht wird, ein Element enthält. -->
  <xsl:variable name= "NOMATCHES" >
    <!-- Für die Wurzel, ihre Attribute und Kinder Übereinstimmung des Musters mit dem Event prüfen. -->
    <xsl:apply-templates select= "$PATTERN" mode= "equalityTest">
      <xsl:with-param name= "CHILDREN" select= "$EVENT" />
      <xsl:with-param name= "VARIABLES" select= "$VARIABLES" />
    </xsl:apply-templates>
  </xsl:variable>
  <!-- Fallunterscheidung, ob ein Übereinstimmung zwischen Pattern und Event vorliegt. Diese wird aufgrund der Ergebnisse des Tests der einzelnen Elemente, die in der Variablen NOMATCHES gespeichert sind, gefällt.
    -->
  <xsl:choose>
  <xsl:when test= "count($NOMATCHES/nomatch) = 0" >
    <xsl:text>TRUE</xsl:text>
    <xsl:copy-of select= "$NOMATCHES/eca:variable" />
  </xsl:when>
```

```
    <xsl:otherwise>
      <xsl:text>FALSE</xsl:text>
    </xsl:otherwise>
    </xsl:choose>
</xsl:template>
<!-- Dieses Template prüft ob ein Element von einer Menge mit einem Element eines Eventmusters übereinstimmt
     . Dazu wird in der Menge ein Element gesucht das mit dem Muster-Element in Name, Namespace, Attributen,
      Kindern, Texten und Variablen übereinstimmt. Das Aktuelle Element mit dem das Template aufgerufen wird
     ist das Muster-Element.
     @param CHILDREN Die Elemente in denen nach einer Übereinstimmung mit dem aktuellen Element gesucht
          werden soll.
     @param VARIABLES Die bereits vorhandenen Variablenbindungen. Der Wert ist als Menge von <variable>-
          Tags anzugeben.
     @return Ein <nomatch>-Tag falls in CHILDREN keine Übereinstimmung mit dem aktuellen Element gefunden
          wurde. Ansonsten gibt es die eventuell aufgetretenen Variablenbindungen zurück. -->
<xsl:template match= "child::*" mode= "equalityTest" >
    <xsl:param name= "CHILDREN" />
    <xsl:param name= "VARIABLES" />
    <xsl:variable name= "CURRENT_PATTERN" select= "." />
    <xsl:variable name= "MATCHES" >
      <xsl:for-each select= "$CHILDREN" >
        <xsl:if test= "( local-name(.) = local-name($CURRENT_PATTERN) ) and ( namespace-uri(.) =
             namespace-uri($CURRENT_PATTERN) )" >
          <!-- Variable, die für jede Übereinstimmung die nicht erreicht wird, ein Element enthält. -->
          <xsl:variable name= "NOMATCHES" >
            <!-- Attribute testen: Für jedes Attribut im Muster eine Übereinstimmung im Event suchen. -->
            <xsl:apply-templates select= "$CURRENT_PATTERN/@*" mode= "equalityTest" >
              <xsl:with-param name= "ATTRIBUTES" select= "./@*" />
              <xsl:with-param name= "VARIABLES" select= "$VARIABLES" />
            </xsl:apply-templates>
            <!-- Kinder testen: Für jedes Kind im Muster eine Übereinstimmung im Event suchen. -->
            <xsl:apply-templates select= "$CURRENT_PATTERN/child::*" mode= "equalityTest">
              <xsl:with-param name= "CHILDREN" select= "./child::*" />
              <xsl:with-param name= "VARIABLES" select= "$VARIABLES" />
            </xsl:apply-templates>
            <!-- Texte testen: Für jeden Textknoten im Muster eine Übereinstimmung im Event suchen. -->
            <xsl:apply-templates select= "$CURRENT_PATTERN/text()[1]" mode= "equalityTest">
              <xsl:with-param name= "CHILDREN" select= "./text()" />
            </xsl:apply-templates>
          </xsl:variable>
          <xsl:if test= "count($NOMATCHES/nomatch) = 0" >
            <match><xsl:copy-of select= "$CURRENT_PATTERN" /></match>
          </xsl:if>
          <xsl:copy-of select= "$NOMATCHES/eca:variable" />
        </xsl:if>
      </xsl:for-each>
    </xsl:variable>
    <xsl:if test= "count($MATCHES/match) = 0" >
      <nomatch>
        <xsl:copy-of select= "$CURRENT_PATTERN" />
      </nomatch>
    </xsl:if>
    <xsl:copy-of select= "$MATCHES/eca:variable" />
</xsl:template>
<!-- Template zum Vergleich eines Attributes mit einem Muster. Zu diesem Zweck prüft das Template Namen,
     Namespace und Wert des Attributes im Muster mit einer Liste von Attributen ob es mit einem aus der Liste
     übereinstimmt. Dabei werden eventuell vorhandene Variablen berücksichtigt.
     @param ATTRIBUTES Die Liste der Attribute in denen eine Übereinstimmung gesucht werden soll. Der Wert
          muss eine Menge von Attribute-Nodes sein.
     @param VARIABLES Die bereits vorhandenen Variablenbindungen. Der Wert ist als Menge von <variable>-
          Tags anzugeben.
     @return Ein <nomatch>-Tag wenn keine Übereinstimmung gefunden wurde ansonsten eine Reihe von <
          variable>-Tags mit den neu aufgetrettenen Variablenbindungen.-->
<xsl:template match= "@*" mode= "equalityTest" >
    <xsl:param name= "ATTRIBUTES" />
    <xsl:param name= "VARIABLES" />
    <xsl:variable name= "CURRENT_ATTRIBUTE" select= "." />
    <!-- Variable zur aufnahme von Elementen die anzeigen das eine Übereinstimmung gefunden wurde. -->
    <xsl:variable name= "MATCHES" >
      <!-- Fallunterscheidung ob bei diesem Attribute ein Parameter oder ein fixer Wert vorliegt. -->
      <xsl:choose>
      <!-- Fall 1: Es liegt eine Variable im Template vor. -->
      <xsl:when test= "starts-with(string($CURRENT_ATTRIBUTE),'$')" >
        <!-- Fallunterscheidung, ob der Parameter bereits einen Wert hat. -->
        <xsl:choose>
        <!-- Fall 1: Die Variable hat einen Wert. -->
        <xsl:when test= "$VARIABLES[string(./@name) = substring(string($CURRENT_ATTRIBUTE),2)]" >
          <xsl:for-each select= "$ATTRIBUTES" >
```

```
                <xsl:if test= "(local-name(.)=local-name($CURRENT_ATTRIBUTE) and (namespace-uri(.)
                    =namespace-uri($CURRENT_ATTRIBUTE) and (string(.)=string($VARIABLES[string(./
                    @name)=substring(string($CURRENT_ATTRIBUTE),2)]) )" >
                  <match><xsl:copy-of select= "." /></match>
                </xsl:if>
              </xsl:for-each>
            </xsl:when><!-- Ende Fall 1 der Unterscheidung, ob ein Wert gesetzt ist. -->
            <!-- Fall 2: Es liegt bisher kein Wert für die Variable vor. -->
            <xsl:otherwise>
              <xsl:for-each select= "$ATTRIBUTES" >
                <xsl:if test= "(local-name(.)=local-name($CURRENT_ATTRIBUTE) and (namespace-uri(.)
                    =namespace-uri($CURRENT_ATTRIBUTE) )" >
                  <match><xsl:copy-of select= "." /></match>
                  <!-- Erzeugen der Variablenbindung. -->
                  <eca:variable>
                    <xsl:attribute name= "name">
                      <xsl:value-of select= "substring(string($CURRENT_ATTRIBUTE),2)" />
                    </xsl:attribute>
                    <xsl:value-of select= "string(.)" />
                  </eca:variable>
                </xsl:if>
              </xsl:for-each>
            </xsl:otherwise><!-- Ende Fall 2 der Unterscheidung, ob ein Wert gesetzt ist. -->
            </xsl:choose><!-- Ende Fallunterscheidung, ob ein Wert für den Parameter gesetzt ist. -->
          </xsl:when><!-- Ende Fall 1 der Unterscheidung, ob ein Parameter vorliegt. -->
            <!-- Fall 2: Es liegt kein Parameter vor. -->
          <xsl:otherwise>
            <xsl:for-each select= "$ATTRIBUTES" >
              <xsl:if test= "(local-name(.)=local-name($CURRENT_ATTRIBUTE) and (namespace-uri(.)=
                  namespace-uri($CURRENT_ATTRIBUTE) and (.=$CURRENT_ATTRIBUTE)" >
                <match><xsl:copy-of select= "." /></match>
              </xsl:if>
            </xsl:for-each>
          </xsl:otherwise><!-- Ende Fall 2 der Unterscheidung, ob ein Parameter vorliegt oder nicht. -->
          </xsl:choose><!-- Ende der Fallunterscheidung ob ein Parameter vorliegt oder nicht. -->
          </xsl:variable><!-- Ende der Variable "MATCHES". -->
    <!-- Diese Abfrage überprüft, ob es keine Übereinstimmung gab und erzeugt mit diesem Wissen das Ergebnis.
        -->
    <xsl:if test= "count($MATCHES/match)=0" >
      <nomatch><xsl:copy-of select= "$CURRENT_ATTRIBUTE" /></nomatch>
    </xsl:if>
    <xsl:copy-of select= "$MATCHES/eca:variable" />
  </xsl:template>
  <!-- Dieses Template prüft ob ein Textknoten aus dem Eventmuster mit dem, an entsprechender Stelle in einem
       Event stehenden, übereinstimmt. Dazu werden führende und anschließende Leerzeichen aus allen beteiligten
       Texten entfernt. Die so entstehenden Texte werden auf gleichheit geprüft.
       @param CHILDREN Die Texte eines Elements aus einem atomaren Event. Von diesen wird das erste
            verwendet.
       @return Ein <nomatch>-Tag wenn keine Übereinstimmung gefunden wurde. Ansonsten ist die Ausgabe leer.
            -->
  <xsl:template match= "text()" mode= "equalityTest" >
    <xsl:param name= "CHILDREN" />
    <xsl:if test= "not(normalize-space(.)='')">
      <xsl:if test= "not(normalize-space(.)=normalize-space($CHILDREN))">
        <nomatch><xsl:value-of select="." /></nomatch>
      </xsl:if>
    </xsl:if>
  </xsl:template>
  <!-- Dieses Template behandelt in einem Eventmuster auftretende Variablen. Es wird beim vergleich eines solchen
       Musters mit einem Event aufgerufen und prüft ob der Wert der Variablen mit dem entsprechenden Wert im
       Event übereinstimmt. Sollte die Variable allerdings an keinen Wert gebunden sein nimmt dieses Template die
       Bindung an einen Wert aus dem betrachteten Event vor.
       @param CHILDREN Die Kinder des Events in denen nach einem passenden Wert zu der Variablen gesucht
            werden soll.
       @param VARIABLES Die bisher vorliegenen Variablenbindungen.
       @return Ein <nomatch>-Tag wenn kein, zu der angegebenen Variable passender Wert gefunden wurde.
            Ansonsten eventuell eine neue Variablenbindung.-->
  <xsl:template match= "eca:variable" mode= "equalityTest" xmlns:eca= "http://www.eca.org/eca-ml">
    <xsl:param name= "CHILDREN" />
    <xsl:param name= "VARIABLES" />
    <xsl:variable name= "CURRENT_PATTERN" select= "." />
    <!--Wenn Bindungsmöglichkeiten existiren: die erste auswählen.-->
    <xsl:variable name="EVENTVALUE">
      <xsl:variable name="BINDINGS">
        <xsl:for-each select="$CHILDREN">
          <!--Überprüfung von Kind aus dem Event und Kind des Variable-Tags.-->
          <xsl:variable name="TESTRESULT">
            <xsl:apply-templates select="$CURRENT_PATTERN/child::*" mode="equalityTest">
              <xsl:with-param name="CHILDREN" select="." />
```

```
                </xsl:apply-templates>
            </xsl:variable>
            <xsl:if test="count($TESTRESULT/*) = 0">
                <xsl:copy-of select="." />
            </xsl:if>
        </xsl:for-each>
    </xsl:variable>
    <xsl:if test="count($BINDINGS/*) &gt; 0">
        <!--Den Wert, der im Event zu der Variablen passt ausgeben.-->
        <xsl:copy-of select="$BINDINGS/*[1]" />
    </xsl:if>
</xsl:variable>
<!-- Fallunterscheidung, ob die Variable schon gebunden ist oder nicht. -->
<xsl:choose>
<!-- Falls die Variable schon gebunden ist:-->
<xsl:when test= "$VARIABLES[string(./@name) = string($CURRENT_PATTERN/@name)]" >
        <!--Variable mit dem Ergebnis des Vergleichs zwischen dem Variablenwert und seinem Gegenstück im
            Event.-->
        <xsl:variable name= "TESTRESULT">
            <!--Vergleich des Variablenwertes mit dem Event.-->
            <xsl:apply-templates select="$VARIABLES[string(./@name) = string($CURRENT_PATTERN/
                @name)]/node()" mode= "equalityTest" >
                <xsl:with-param name= "CHILDREN" select= "$EVENTVALUE/*" />
            </xsl:apply-templates>
            <!--Vergleich des Events mit dem Variablenwert.-->
            <xsl:apply-templates select="$EVENTVALUE/*" mode= "equalityTest" >
                <xsl:with-param name= "CHILDREN" select= "$VARIABLES[string(./@name) = string($
                    CURRENT_PATTERN/@name)]/node()" />
            </xsl:apply-templates>
        </xsl:variable>
    <!--Auswertung der Variablen:-->
    <xsl:choose>
    <!--Event und Variablenwert stimmen überein:-->
    <xsl:when test="count($TESTRESULT/*) = 0">
        <match><xsl:copy-of select= "./child::*" /></match>
    </xsl:when>
    <!--Anderenfalls:-->
    <xsl:otherwise>
        <nomatch><xsl:copy-of select= "./child::*" /></nomatch>
    </xsl:otherwise>
    </xsl:choose>
</xsl:when>
<!--Die Variable ist bisher nicht gebunden.-->
<xsl:otherwise>
    <xsl:choose>
    <xsl:when test="count($EVENTVALUE/*) &gt; 0">
        <eca:variable>
            <xsl:copy-of select= "./@name" />
            <xsl:copy-of select="$EVENTVALUE/*" />
        </eca:variable>
        <match><xsl:copy-of select= "./child::*" /></match>
    </xsl:when>
    <xsl:otherwise>
        <nomatch><xsl:copy-of select= "./child::*" /></nomatch>
    </xsl:otherwise>
    </xsl:choose>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
<!-- Dieses Template überprüft ob die Variablenbindungen einer Automateninstanz mit denen eines Composite
    Events übereinstimmen. Die Variablen die nur in einem der beiden vorkommen sind dabei ohne belang. Bei
    denen allerdings die im Namen übereinstimmen wird eine Prüfung ihrer Inhalts auf Identität vorgenommen.
    @param INSTANCE Die Automateninstanz die zum Vergleich herangezogen wird.
    @param EVENT Das Composite Event das zum Vergleich herangezogen wird.
    @return TRUE oder FALSE je nach Ergebnis des Vergleichs.-->
<xsl:template name="variableTest">
<xsl:param name="INSTANCE" />
<xsl:param name="EVENT" />
<!--Variable zum Ablegen der Ergebnisse der einzelnen Vergleiche für jede Variablenbindung.-->
<xsl:variable name= "TESTRESULT">
    <xsl:for-each select="$INSTANCE/ceda:instance/ceda:variables/eca:variable">
        <xsl:variable name="CURRENT_VARIABLE" select="."/>
        <!--Jede Variable aus dem Event, die zu einer aus der Instanz passt, untersuchen.-->
        <xsl:for-each select="$EVENT/ceda:instance/ceda:variables/eca:variable[./@name = $
            CURRENT_VARIABLE/@name]">
            <!--Aufruf des Vergleichs für jeden Knoten innerhalb der Variablen aus dem Event.-->
            <xsl:apply-templates select="./node()" mode= "equalityTest" >
                <xsl:with-param name= "CHILDREN" select= "$CURRENT_VARIABLE/node()" />
            </xsl:apply-templates>
            <!--Aufruf des Vergleichs für jeden Knoten innerhalb der Variablen aus der Instanz.-->
```

```
            <xsl:apply-templates select="$CURRENT_VARIABLE/node()" mode= "equalityTest" >
                <xsl:with-param name= "CHILDREN" select= "./node()" />
            </xsl:apply-templates>
        </xsl:for-each>
    </xsl:for-each>
</xsl:variable>
<!--Auswertung der Variablen:-->
<xsl:choose>
<!--Sollte es nur Übereinstimmungen für die einzelnen Variablen geben, ist das Ergebnis positiv.-->
<xsl:when test="count($TESTRESULT/*) = 0">
    <xsl:text>TRUE</xsl:text>
</xsl:when>
<!--Anderenfalls ist er das nicht.-->
<xsl:otherwise>
    <xsl:text>FALSE</xsl:text>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

C.1.4 Stylesheet „ceda-instance“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "yes"?>
<!-- Dieses Stylesheet ist ein Modul für den Algorithmus der ein atomaren Events auf die "Composite_Event_
     Detection_Automatons" anwendet. -->
<!-- Dieses Modul ist zuständig für die Verarbeitung der einzelnen Instanzen von Automaten. Es besitzt Templates
     um die Instanzen aus dem CEDA-Format in das interne Format des Algorithmus umzuwandeln. Darüber
     hinaus ist es in der Lage weitere Instanzen aus Events zu erzeugen und Kopien von bestehenden Instanzen zu
     erzeugen. -->
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:ceda= "http://rewerse.net/
    event-automaton-ml" version= "1.0">
<!-- Dieses Template erzeugt aus dem ceda:instance Element die interne Representation für den Algorithmus.
     Dabei wird die Automateninstanz in ein Element namens 'instancewrapper' verpackt. Dieses Element nimmt
     mittels Attributen die internen Informationen des Algorithmus auf. -->
<xsl:template name= "createAutomatoninstance" match="/ceda:ceda/ceda:instance">
  <xsl:element name= "instancewrapper" >
    <xsl:attribute name= "stepcounter" >0</xsl:attribute>
    <xsl:attribute name= "new" >no</xsl:attribute>
    <xsl:attribute name= "stable" >yes</xsl:attribute>
    <!-- Die eigentliche Automateninstanz wird hier hinzugefügt. -->
    <xsl:copy-of select= "." />
  </xsl:element>
</xsl:template>
<!-- Dieses Template schaltet die in OLD_INSTANCE übergebene Automateninstanze mit dem Event EVENT in
     den Zustand der in TARGET angegeben ist. Dabei muss EVENTein atomares Event sein.
     @param OLD_INSTANCE Der Wrapper der Instanze die verändert werden soll.
     @param EVENT Der Wrapper des atomaren Events das die Änderung ausgelöst hat.
     @param TARGET Der neue Zustand auf den die Automateninstanze geschaltet werden soll.
     @param STEP Der interne Schrittzähler des Algorithmus. Gibt an wann die Instanz entstanden ist.
     @param NEW_VARIABLES Die Variablenbindungen die sich durch das Anwenden von EVENT ergeben haben.
          -->
<xsl:template name= "createNewInstanceWithAtomicEvent" >
  <xsl:param name= "OLD_INSTANCE" />
  <xsl:param name= "EVENT" />
  <xsl:param name= "TARGET" />
  <xsl:param name= "STEP" />
  <xsl:param name= "NEW_VARIABLES" />
  <!-- Wrapper anlegen und mit dem neuen Kontext des Automaten befüllen.-->
  <xsl:element name= "instancewrapper">
    <xsl:attribute name= "stepcounter"><xsl:value-of select= "$STEP" /></xsl:attribute>
    <xsl:attribute name= "new">yes</xsl:attribute>
    <xsl:attribute name= "stable">yes</xsl:attribute>
    <!-- Element 'instance' anlegen und mit den geänderten Daten bestücken. -->
    <ceda:instance>
      <!--Die Referenz auf den Automaten, dessen Kontext durch den vorliegenden Status ausgetrückt wird,
          kopieren.-->
      <xsl:attribute name= "automaton">
        <xsl:value-of select= "$OLD_INSTANCE/ceda:instance/@automaton" />
      </xsl:attribute>
      <!--Das Element mit der Referenz auf den neuen Zustand anlegen.-->
      <ceda:current-state>
        <!--Das Attribut mit der Referenz auf den neuen Zustand anlegen.-->
        <xsl:attribute name="ref">
          <!--Bestimmung des Wertes der Referenz auf den neuen Zustand.-->
          <xsl:value-of select= "$TARGET" />
        </xsl:attribute>
      </ceda:current-state>
      <!-- Den Parameterstore aktualisieren. -->
      <ceda:parameterstore>
        <xsl:copy-of select= "$OLD_INSTANCE/ceda:instance/ceda:parameterstore/*" />
        <!--Das Event dem Parameterstore hinzufügen. -->
        <xsl:copy-of select= "$EVENT/child::*" />
      </ceda:parameterstore>
      <!-- Die Variablen dem Variables-Tags hinzufügen. -->
      <ceda:variables>
        <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$OLD_INSTANCE/ceda:instance/
            ceda:variables/eca:variable" />
        <xsl:copy-of select= "$NEW_VARIABLES" />
      </ceda:variables>
    </ceda:instance>
  </xsl:element>
</xsl:template>
<!-- Dieses Template schaltet die in OLD_INSTANCE übergebene Automateninstanze mit dem Event EVENT in
     den Zustand der in TARGET angegeben ist. Dabei muss EVENT ein Composite Event sein.
     @param OLD_INSTANCE Der Wrapper der Instanze die verändert werden soll.
     @param EVENT Der Wrapper des Composite Events das die Änderung ausgelöst hat.
     @param TARGET Der neue Zustand auf den die Automateninstanze geschaltet werden soll.
     @param STEP Der interne Schrittzähler des Algorithmus. -->
<xsl:template name= "createNewInstanceWithCompositeEvent" >
  <xsl:param name= "OLD_INSTANCE" />
```

```
    <xsl:param name= "EVENT" />
    <xsl:param name= "TARGET" />
    <xsl:param name= "STEP" />
    <!-- Wrapper anlegen und mit dem neuen Kontext des Automaten befüllen.-->
    <xsl:element name= "instancewrapper">
      <xsl:attribute name= "stepcounter"><xsl:value-of select= "$STEP" /></xsl:attribute>
      <xsl:attribute name= "new">yes</xsl:attribute>
      <xsl:attribute name= "stable">yes</xsl:attribute>
      <!-- Element 'instance' anlegen und mit den geaenderten Daten bestücken. -->
      <ceda:instance>
        <!--Die Referenz auf den Automaten dessen Kontext durch den vorliegenden Status ausgetrückt wird
            kopieren.-->
        <xsl:attribute name= "automaton">
          <xsl:value-of select= "$OLD_INSTANCE/ceda:instance/@automaton" />
        </xsl:attribute>
        <!-- Das Element mit der Referenz auf den neuen Zustand anlegen.-->
        <xsl:element name= "ceda:current-state">
          <!-- Das Attribut mit der Referenz auf den neuen Zustand anlegen.-->
          <xsl:attribute name="ref">
            <!-- Bestimmung des Wertes der Referenz auf den neuen Zustand.-->
            <xsl:value-of select= "$TARGET" />
          </xsl:attribute>
        </xsl:element>
        <!-- Den Parameterstore aktualisieren. -->
        <xsl:element name= "ceda:parameterstore">
          <!-- Alten Parameterstore kopieren -->
          <xsl:copy-of select= "$OLD_INSTANCE/ceda:instance/ceda:parameterstore/*" />
          <!-- Das composite Event dem Parameterstore hinzufuegen. -->
          <xsl:element name= "ceda:composite-event">
            <xsl:copy-of select= "$EVENT/ceda:instance/ceda:parameterstore/*" />
          </xsl:element>
        </xsl:element>
        <!-- Die neuen Variablen dem Variables-Tag hinzufügen -->
        <ceda:variables>
          <!--Die alten Variablen übernehmen.-->
          <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$OLD_INSTANCE/ceda:instance/
              ceda:variables/eca:variable" />
          <!--Die Variablen aus dem hinzugefügten, Composite Event hinzufügen sofern sie nicht schon
              vorhanden sind.-->
          <xsl:copy-of xmlns:eca= "http://www.eca.org/eca-ml" select= "$EVENT/ceda:instance/ceda:variables/
              eca:variable[not(./@name = $OLD_INSTANCE/ceda:instance/ceda:variables/eca:variable/@name)]"
              />
        </ceda:variables>
      </ceda:instance>
    </xsl:element>
  </xsl:template>
  <!-- Dieses Template erzeugt eine Kopie einer Automateninstanz. Diese Kopie wird am Ende von des Algorithmus
       nicht in die Ausgabe übergehen.
      @param OLD_INSTANCE Der Wrapper der Instanz die kopiert werden soll. -->
  <xsl:template name= "createInstableCopy">
    <xsl:param name= "OLD_INSTANCE" />
    <xsl:element name= "instancewrapper">
      <xsl:attribute name= "stepcounter"><xsl:value-of select= "$OLD_INSTANCE/@stepcounter" /></
          xsl:attribute>
      <xsl:attribute name= "new"><xsl:value-of select= "$OLD_INSTANCE/@new" /></xsl:attribute>
      <!-- Das Attribute stable auf no setzen. Damit wird angezeigt, dass diese Automateninstanz nicht in die
          Ausgabe übergeht. -->
      <xsl:attribute name= "stable">no</xsl:attribute>
      <!-- Die Instanz in den geänderten Wrapper kopieren. -->
      <xsl:copy-of select= "$OLD_INSTANCE/child::ceda:instance" />
    </xsl:element>
  </xsl:template>
  </xsl:stylesheet>
```

C.2 Erzeugung der Automaten

C.2.1 Stylesheet „Create Automaton“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<!-- Dieses Stylesheet ist Teil der Transformation von Eventausdrücken zu Automaten. Es übernimmt die
     Erzeugung der Automaten zu den Snoop-Operatorn ausser Any. Dieser Operator kann mit einem eigenen
     Stylesheet, das hier importiert wird, transformiert werden. -->
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:snoop= "http://rewerse.net/
    snoop-expression-ml" xmlns:ceda= "http://rewerse.net/event-automaton-ml" version= "1.0">
<xsl:strip-space elements= "" />
<!-- In dieser Parameter wird der Name des Automaten gesetzt der aus einem Ausdruck erzeugt wird. -->
<xsl:param name= "AUTOMATON_NAME">AutomatonX</xsl:param>
<!--Inklusionen-->
<xsl:include href= "create_any-operator-automaton.xsl" />
<!-- Output Übernimmt die Spezifizierung der zu erzeugenden Ausgabe. In diesem Fall ein XML-Dokument der
     Version 1.0 in UTF-8-Kodierung mit Einrückungen, einer XML-Deklaration und dem Medien-Typ "text/
     xml".-->
<xsl:output method= "xml" version= "1.0"
            encoding= "UTF-8" indent= "yes"
            cdata-section-elements= ""
            omit-xml-declaration= "no"
            standalone= "no"
            doctype-system= "./ceda.dtd"
            media-type= "text/xml" />
<!--Template das Überprüft ob der Input eine Eventdeclaration enthält. Wenn nicht wird mitels xsl:message eine
     Entsprechende Nachricht ausgegeben und die Verarbeitung abgebrochen.-->
<xsl:template match= "/">
   <xsl:choose>
      <xsl:when test= "/snoop:Eventdeclaration">
         <xsl:apply-templates select= "/snoop:Eventdeclaration" />
      </xsl:when>
      <xsl:otherwise>
         <xsl:message terminate= "yes">
            <event>
               Das Stylesheet wurde nicht auf die Declaration eines Snoop-Ausdrucks angewannt.
            </event>
         </xsl:message>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
<!--Ausgangspunkt der Erzeugung der Automaten. Hier wird das Element Eventdeclaration verarbeitet. Dazu wird
     ein ceda-Element erzeugt das die Automaten aufnimmt. Die Automten werden dann von weiteren Templates
     hinzugefügt.-->
<xsl:template match= "/snoop:Eventdeclaration">
   <ceda:ceda>
      <xsl:apply-templates select= "./child::*" >
         <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_NAME" /></
             xsl:with-param>
      </xsl:apply-templates>
   </ceda:ceda>
</xsl:template>
<!-- Dieses Template verarbeitet jeden AND-Operatoren zu einem Automaten. Dazu werden die ersten beiden
     Kinder des Operator-Elements als Operanden verwendet.
     @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:And" >
   <xsl:param name= "AUTOMATON_ID" />
   <!-- Erzeugen des Automaten zu einem AND-Operator. -->
   <ceda:automaton time-constraint= "equalorless">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification> AND </ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
         <xsl:attribute name= "id">
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</xsl:text>
         </xsl:attribute>
         <ceda:transition>
            <xsl:attribute name= "target">
               <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</xsl:text>
            </xsl:attribute>
            <xsl:choose>
            <xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
               <xsl:attribute name= "typ">primitiv</xsl:attribute>
               <xsl:copy-of select= "./child::*[1]/child::*[1]" />
            </xsl:when>
            <xsl:otherwise>
               <xsl:attribute name= "typ">composite</xsl:attribute>
               <xsl:attribute name= "subautomaton">
                  <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub1</xsl:text>
               </xsl:attribute>
```

```
            </xsl:otherwise>
            </xsl:choose>
        </ceda:transition>
        <ceda:transition>
            <xsl:attribute name= "target">
                <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text>
            </xsl:attribute>
            <xsl:choose>
            <xsl:when test= "./child::*[2][name(.)=’snoop:Atomic-Event’]">
                <xsl:attribute name= "typ">primitiv</xsl:attribute>
                <xsl:copy-of select= "./child::*[2]/child::*[1]" />
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name= "typ">composite</xsl:attribute>
                <xsl:attribute name= "subautomaton">
                    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub2</xsl:text>
                </xsl:attribute>
            </xsl:otherwise>
            </xsl:choose>
        </ceda:transition>
    </ceda:state>
    <ceda:state typ= "normal" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id">
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</xsl:text>
        </xsl:attribute>
        <ceda:transition>
            <xsl:attribute name= "target">
                <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>
            </xsl:attribute>
            <xsl:choose>
            <xsl:when test= "./child::*[2][name(.)=’snoop:Atomic-Event’]">
                <xsl:attribute name= "typ">primitiv</xsl:attribute>
                <xsl:copy-of select= "./child::*[2]/child::*[1]" />
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name= "typ">composite</xsl:attribute>
                <xsl:attribute name= "subautomaton">
                    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub2</xsl:text>
                </xsl:attribute>
            </xsl:otherwise>
            </xsl:choose>
        </ceda:transition>
    </ceda:state>
    <ceda:state typ= "normal" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id">
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text>
        </xsl:attribute>
        <ceda:transition>
            <xsl:attribute name= "target">
                <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>
            </xsl:attribute>
            <xsl:choose>
            <xsl:when test= "./child::*[1][name(.)=’snoop:Atomic-Event’]">
                <xsl:attribute name= "typ">primitiv</xsl:attribute>
                <xsl:copy-of select= "./child::*[1]/child::*[1]" />
            </xsl:when>
            <xsl:otherwise>
                <xsl:attribute name= "typ">composite</xsl:attribute>
                <xsl:attribute name= "subautomaton">
                    <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub1</xsl:text>
                </xsl:attribute>
            </xsl:otherwise>
            </xsl:choose>
        </ceda:transition>
    </ceda:state>
    <ceda:state typ= "ende" detect= "yes" >
        <xsl:attribute name= "id">
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text>
        </xsl:attribute>
    </ceda:state>
</ceda:automaton>
<!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
<xsl:apply-templates select= "./child::*[1][not(name(.)=’snoop:Atomic-Event’)]">
    <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
        _sub1</xsl:text></xsl:with-param>
</xsl:apply-templates>
<xsl:apply-templates select= "./child::*[2][not(name(.)=’snoop:Atomic-Event’)]">
    <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
        _sub2</xsl:text></xsl:with-param>
</xsl:apply-templates>
```

```
    <!-- Erzeugen der initialen Instanzen. -->
    <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
</xsl:template>
<!--Dieses Template verarbeitet den Operator OR. Dazu werden die ersten beiden Kinder des Operator-Elements
      als Operanden verwendet.
    @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:Or" >
    <xsl:param name= "AUTOMATON_ID" />
    <!-- Erzeugen des Automaten zu einem AND-Operator. -->
    <ceda:automaton time-constraint= "non">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification> OR </ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes">
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub1</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[2][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[2]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub2</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "ende" detect= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</
            xsl:text></xsl:attribute>
      </ceda:state>
    </ceda:automaton>
    <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
    <xsl:apply-templates select= "./child::*[1][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub1</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[2][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub2</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <!-- Erzeugen der initialen Instanzen. -->
    <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
</xsl:template>
<!--Dieses Template verarbeitet den Operator SEQUENZ. Dazu werden die ersten beiden Kinder des
      Operator-Elements als Operanden verwendet.
    @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:Sequence" >
    <xsl:param name= "AUTOMATON_ID" />
    <ceda:automaton time-constraint= "less">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification> ; </ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
```

```
            </xsl:when>
            <xsl:otherwise>
               <xsl:attribute name= "typ">composite</xsl:attribute>
               <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                  _sub1</xsl:text></xsl:attribute>
            </xsl:otherwise></xsl:choose>
         </ceda:transition>
      </ceda:state>
      <ceda:state typ= "normal" detect= "no" duplicate= "yes" >
         <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</
            xsl:text></xsl:attribute>
         <ceda:transition>
            <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3
               </xsl:text></xsl:attribute>
            <xsl:choose><xsl:when test= "./child::*[2][name(.) = 'snoop:Atomic-Event']">
               <xsl:attribute name= "typ">primitiv</xsl:attribute>
               <xsl:copy-of select= "./child::*[2]/child::*[1]" />
            </xsl:when>
            <xsl:otherwise>
               <xsl:attribute name= "typ">composite</xsl:attribute>
               <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                  _sub2</xsl:text></xsl:attribute>
            </xsl:otherwise></xsl:choose>
         </ceda:transition>
      </ceda:state>
      <ceda:state typ= "ende" detect= "yes" >
         <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3</
            xsl:text></xsl:attribute>
      </ceda:state>
   </ceda:automaton>
   <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
   <xsl:apply-templates select= "./child::*[1][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
         _sub1</xsl:text></xsl:with-param>
   </xsl:apply-templates>
   <xsl:apply-templates select= "./child::*[2][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
         _sub2</xsl:text></xsl:with-param>
   </xsl:apply-templates>
   <!-- Erzeugen der initialen Instanzen. -->
   <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
   </xsl:call-template>
</xsl:template>
<!--Dieses Template verarbeitet den Operator ANY*. Dazu wird das erste Kind des Operator-Elements als
   Operand verwendet.
   @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:Multi_Occurrences" >
   <xsl:param name= "AUTOMATON_ID" />
   <!-- Erzeugen des Automaten zu einem AND-Operator. -->
   <ceda:automaton time-constraint= "less">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification>Any(<xsl:value-of select= "./@number-of-occurrences" />, *)</
         ceda:eventspecification>
      <xsl:call-template name= "Multi-Occurrences-Step" >
         <xsl:with-param name= "STEPS" select= "./@number-of-occurrences" />
         <xsl:with-param name= "STEPCOUNTER" select= "1" />
         <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
      </xsl:call-template>
   </ceda:automaton>
   <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
   <xsl:apply-templates select= "./child::*[1][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
         _sub</xsl:text></xsl:with-param>
   </xsl:apply-templates>
   <!-- Erzeugen der initialen Instanzen. -->
   <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
   </xsl:call-template>
</xsl:template>
<!--Dieses Template verarbeitet den Operator APERIODIC. Dazu werden die ersten drei Kinder des
   Operator-Elements als Operanden verwendet.
   @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:Aperiodic" name= "aperiodic_event" >
   <xsl:param name= "AUTOMATON_ID" />
   <!-- Erzeugen des Automaten zu einem AND-Operator. -->
   <ceda:automaton time-constraint= "equalorless" >
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification>A(,,)</ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
```

```
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-start</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "normal" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[2][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[2]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-ausloeser</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[3][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[3]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-stop</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "ende" detect= "yes" ><xsl:attribute name= "id"><xsl:value-of select= "$
          AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text></xsl:attribute></ceda:state>
      <ceda:state typ= "abort" detect= "no" ><xsl:attribute name= "id"><xsl:value-of select= "$
          AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text></xsl:attribute></ceda:state>
    </ceda:automaton>
    <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
    <xsl:apply-templates select= "./child::*[1][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-start</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[2][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-ausloeser</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[3][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-stop</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <!-- Erzeugen der initialen Instanzen. -->
    <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
  </xsl:template>
  <!--Dieses Template verarbeitet den Operator APERIODIC*. Dazu werden die ersten drei Kinder des
      Operator-Elements als Operanden verwendet.
    @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
  <xsl:template match= "//snoop:Cumulative_Aperiodic" name= "Cumulative_aperiodic_event">
    <xsl:param name= "AUTOMATON_ID"/>
    <!-- Erzeugen des Automaten zu einem AND-Operator. -->
    <ceda:automaton time-constraint= "equalorless">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification>A*(,,)</ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</
```

```
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[1][name(.)=' snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-starter</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "normal" detect= "no" duplicate= "no" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[2][name(.)=' snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[2]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-collecter</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3
              </xsl:text></xsl:attribute>
          <xsl:choose><xsl:when test= "./child::*[3][name(.)=' snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[3]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-stoper</xsl:text></xsl:attribute>
          </xsl:otherwise></xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "ende" detect= "yes" ><xsl:attribute name= "id"><xsl:value-of select= "$
          AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text></xsl:attribute></ceda:state>
    </ceda:automaton>
    <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
    <xsl:apply-templates select= "./child::*[1][not(name(.)=' snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-starter</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[2][not(name(.)=' snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-collecter</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[3][not(name(.)=' snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-stoper</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <!-- Erzeugen der initialen Instanzen. -->
    <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
  </xsl:template>
  <!--Dieses Template verarbeitet den Periodic-Event-Operator. Da die Automten zu denen des
      Aperiodic-Event-Operators identisch sind wird dessen Template verwendet.
    @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
  <xsl:template match= "//snoop:Periodic" >
    <xsl:param name= "AUTOMATON_ID" />
    <xsl:call-template name= "aperiodic_event" >
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
  </xsl:template>
  <!--Dieses Template verarbeitet den Periodic*-Event-Operator. Da die Automten zu denen des Aperiodic*-
      Event-Operators identisch sind wird dessen Template verwendet.
    @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
  <xsl:template match= "//snoop:Cumulative_Periodic" >
    <xsl:param name= "AUTOMATON_ID"/>
```

```
    <xsl:call-template name= "Cumulative_aperiodic_event" >
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
</xsl:template>
<!--Dieses Template verarbeitet den NOT-Operator. Dazu werden die ersten drei Kinder des Operator-Elements
    als Operanden verwendet.
   @param AUTOMATON_ID Diesers Parameter enthaelt den Namen des zu erzeugenden Automaten.-->
<xsl:template match= "//snoop:Not" >
    <xsl:param name= "AUTOMATON_ID" />
    <!-- Erzeugen des Automaten zu einem AND-Operator. -->
    <ceda:automaton time-constraint="equalorless">
      <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
      <ceda:eventspecification>Not()[]</ceda:eventspecification>
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand1</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2
              </xsl:text></xsl:attribute>
          <xsl:choose>
          <xsl:when test= "./child::*[1][name(.)=’snoop:Atomic-Event’]">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-starter</xsl:text></xsl:attribute>
          </xsl:otherwise>
          </xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "normal" detect= "no" duplicate= "no">
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand2</
            xsl:text></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand3
              </xsl:text></xsl:attribute>
          <xsl:choose>
          <xsl:when test= "./child::*[2][name(.)=’snoop:Atomic-Event’]">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[2]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-not</xsl:text></xsl:attribute>
          </xsl:otherwise>
          </xsl:choose>
        </ceda:transition>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand4
              </xsl:text></xsl:attribute>
          <xsl:choose>
          <xsl:when test= "./child::*[3][name(.)=’snoop:Atomic-Event’]">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[3]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub-stoper</xsl:text></xsl:attribute>
          </xsl:otherwise>
          </xsl:choose>
        </ceda:transition>
      </ceda:state>
      <ceda:state typ= "abort" detect= "no" ><xsl:attribute name= "id"><xsl:value-of select= "$
          AUTOMATON_ID" /><xsl:text>_zustand3</xsl:text></xsl:attribute></ceda:state>
      <ceda:state typ= "ende" detect= "yes" ><xsl:attribute name= "id"><xsl:value-of select= "$
          AUTOMATON_ID" /><xsl:text>_zustand4</xsl:text></xsl:attribute></ceda:state>
    </ceda:automaton>
    <!-- Aufruf der Kinder im Operatorbaum, um aus ihnen wiederum Automaten zu erzeugen. -->
    <xsl:apply-templates select= "./child::*[1][not(name(.)=’snoop:Atomic-Event’)]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-starter</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select= "./child::*[2][not(name(.)=’snoop:Atomic-Event’)]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-not</xsl:text></xsl:with-param>
    </xsl:apply-templates>
```

```
    <xsl:apply-templates select= "./child::*[3][not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
          _sub-stoper</xsl:text></xsl:with-param>
    </xsl:apply-templates>
    <!-- Erzeugen der initialen Instanzen. -->
    <xsl:call-template name="createInitialInstance">
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
    </xsl:call-template>
  </xsl:template>
  <!-- Dieses Template wird bei der Verarbeitung eines Any*-Operator verwendet. Er erzeugt in einer Rekursiven
       Abfolge die sequentielle Abfolge der Zustände.
       @param STEPS Dieser Parameter entspricht dem Wert des Attributes 'number-of-occurrences'.
       @param STEPCOUNTER Dieser Parameter zählt die Anzahl der Rekursionsschritte.
       @param AUTOMATON_ID Der Name des Automaten der erzeugt werden soll. -->
  <xsl:template name= "Multi-Occurrences-Step" >
    <xsl:param name= "STEPS" />
    <xsl:param name= "STEPCOUNTER" select= "1"/>
    <xsl:param name= "AUTOMATON_ID" />
    <xsl:choose>
    <xsl:when test= "$STEPCOUNTER = 1" >
      <ceda:state typ= "initial" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand</
            xsl:text><xsl:value-of select= "$STEPCOUNTER" /></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand<
              /xsl:text><xsl:value-of select= "$STEPCOUNTER + 1" /></xsl:attribute>
          <xsl:choose>
          <xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
              <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub</xsl:text></xsl:attribute>
          </xsl:otherwise>
          </xsl:choose>
        </ceda:transition>
      </ceda:state>
    </xsl:when>
    <xsl:when test= "$STEPCOUNTER &gt; 1 and $STEPCOUNTER &lt; $STEPS + 1" >
      <ceda:state typ= "normal" detect= "no" duplicate= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand</
            xsl:text><xsl:value-of select= "$STEPCOUNTER" /></xsl:attribute>
        <ceda:transition>
          <xsl:attribute name= "target"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand<
              /xsl:text><xsl:value-of select= "$STEPCOUNTER + 1" /></xsl:attribute>
          <xsl:choose>
          <xsl:when test= "./child::*[1][name(.) = 'snoop:Atomic-Event']">
            <xsl:attribute name= "typ">primitiv</xsl:attribute>
            <xsl:copy-of select= "./child::*[1]/child::*[1]" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name= "typ">composite</xsl:attribute>
            <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>
                _sub</xsl:text></xsl:attribute>
          </xsl:otherwise>
          </xsl:choose>
        </ceda:transition>
      </ceda:state>
    </xsl:when>
    <xsl:when test= "$STEPCOUNTER = $STEPS + 1" >
      <ceda:state typ= "ende" detect= "yes" >
        <xsl:attribute name= "id"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustand</
            xsl:text><xsl:value-of select= "$STEPCOUNTER" /></xsl:attribute>
      </ceda:state>
    </xsl:when>
    </xsl:choose>
    <xsl:if test= "$STEPCOUNTER &lt;= $STEPS" >
      <xsl:call-template name= "Multi-Occurrences-Step" >
        <xsl:with-param name= "STEPS" select= "$STEPS" />
        <xsl:with-param name= "STEPCOUNTER" select= "$STEPCOUNTER + 1" />
        <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
  <!-- Dieses Template sorgt dafür das die variable-bindings-Elemente ignoriert werden die keinen eigenen
       Automten erzeugen. -->
  <xsl:template match= "/snoop:Eventdeclaration/eca:variable-bindings" xmlns:eca= "http://www.eca.org/eca-ml">
    </xsl:template>
```

```
<!-- Dieses Template erzeugt für jedes Tuple von Variablenbindungen eine Instanz des Automaten im initialen
     Zustand.
     @param AUTOAMTON_ID Die ID des Automaten zu dem eine Instanz erzeugt werden soll.
     @param INITIAL_STATE Das Postfix zur ID des Automaten mit dem der initiale Zustand versehen wird.
          Standartmäßig ist dieser Wert 'zustand2'.
     @return Ein oder mehrere Instancen des gewünschten Automaten. -->
<xsl:template name= "createInitialInstance" >
  <xsl:param name= "AUTOMATON_ID" />
  <xsl:param name= "INITIAL_STATE" select= "'_zustand1'"/>
  <xsl:choose>
  <!-- Wenn Variablen vorbelegt sind, dann soll für jedes Variablen-Tuple eine Instanz erzeugt werden. -->
  <xsl:when test= "/snoop:Eventdeclaration/eca:variable-bindings" xmlns:eca= "http://www.eca.org/eca-ml" >
    <xsl:for-each select="/snoop:Eventdeclaration/eca:variable-bindings/eca:tuple" >
      <ceda:instance>
        <xsl:attribute name= "automaton"><xsl:value-of select= "$AUTOMATON_ID" /></xsl:attribute>
        <ceda:current-state>
          <xsl:attribute name= "ref">
            <xsl:value-of select= "$AUTOMATON_ID" /><xsl:value-of select= "$INITIAL_STATE" />
          </xsl:attribute>
        </ceda:current-state>
        <ceda:parameterstore />
        <ceda:variables>
          <xsl:copy-of select= "./eca:variable" />
        </ceda:variables>
      </ceda:instance>
    </xsl:for-each>
  </xsl:when>
  <!-- Anderenfalls muss nur eine Instanz ohne bestehende Variablenbindungen erzeugt werden. -->
  <xsl:otherwise>
    <ceda:instance>
      <xsl:attribute name= "automaton"><xsl:value-of select= "$AUTOMATON_ID" /></xsl:attribute>
      <ceda:current-state>
        <xsl:attribute name= "ref">
          <xsl:value-of select= "$AUTOMATON_ID" /><xsl:value-of select= "$INITIAL_STATE" />
        </xsl:attribute>
      </ceda:current-state>
      <ceda:parameterstore />
      <ceda:variables />
    </ceda:instance>
  </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

C.2.2 Stylesheet „Create Any-Operator-Automaton“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<!-- Dieses Stylesheet ist Teil der Transformation von Eventausdrücken zu Automaten. Es trägt die
     Transformation des Snoop-Operators Any zum gesamten Verfahren bei. -->
<xsl:stylesheet version= "1.0" xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:snoop= "http://rewerse
     .net/snoop-expression-ml" xmlns:ceda= "http://rewerse.net/event-automaton-ml" >
<!-- Bei diesem Template beginnt die Transformation des Any-Operators. Das Template erzeugt den Automaten
     und berücksichtigt dabei den Parameter 'number-of-occurrences'. Im Anschluß wird für jeden inneren
     Operator das entsprechende Template aufgerufen. Abschließend werden ein oder mehrere initialen Zustände
     erzeugt.
     @param AUTOMATON_ID Als Id des Automaten wird dieser Parameter verwendet. -->
<xsl:template match= "//snoop:Any" >
  <xsl:param name= "AUTOMATON_ID" />
  <!-- Variable die die Operanden, angereichert um eine Positionsnummer, enthält. -->
  <xsl:variable name= "EVENTS" >
    <!-- Reichert die Operanden um ihre Position an. -->
    <xsl:call-template name= "create_event_element" >
      <xsl:with-param name= "EVENTSX" select= "./child::*" />
      <xsl:with-param name= "EVENTNR" select= "'1'" />
    </xsl:call-template>
  </xsl:variable>
  <!-- Erzeugt den Automaten mit all seinen Zuständen und Transitionen. -->
  <ceda:automaton time-constraint="equalorless">
    <!-- Setzt die ID des Automaten und eine lesbare Form des (Teil-)Ausdrucks. -->
    <xsl:attribute name= "id" select= "$AUTOMATON_ID" />
    <ceda:eventspecification>Any(<xsl:value-of select= "./@number-of-occurrences" />, )</
        ceda:eventspecification>
    <!-- Erzeugt rekursiv die Zustände und ihre Transitionen untereinander. Dazu wird die Anzahl der
         Operanden, die Automatenid des Any-Automaten, die Operanden und den namen des ersten Zustands.
         -->
    <xsl:call-template name= "create_any_zustand" >
      <xsl:with-param name= "NUMBER_OF_STEPS" ><xsl:value-of select= "./@number-of-occurrences"
          /></xsl:with-param>
      <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
      <xsl:with-param name= "EVENTS" ><xsl:copy-of select= "$EVENTS" /></xsl:with-param>
      <xsl:with-param name= "ZUSTANDNAME" ><xsl:value-of select= "$AUTOMATON_ID" />
          _zustand1</xsl:with-param>
    </xsl:call-template>
    <!-- Erzeugt den Endzustand. Er ist auch gleichzeitig der Auslöser eines Evnets. -->
    <ceda:state detect= "yes" typ= "ende" >
      <xsl:attribute name= "id" ><xsl:value-of select= "$AUTOMATON_ID" />_zustandE</xsl:attribute>
    </ceda:state>
  </ceda:automaton>
  <!-- Erzeugt die Automaten für die untergeordneten Operatoren. Jeder Automat der hier erzeugt wird erhält
       eine Id. Diese setzt sich aus dem bisherigen Automatennamen und dem Zusatz '_sub' gefolgt von der
       Positionsnummer des Operanden zusammen. -->
  <xsl:for-each select= "$EVENTS/*">
    <xsl:apply-templates select= ".[not(name(.) = 'snoop:Atomic-Event')]">
      <xsl:with-param name= "AUTOMATON_ID"><xsl:value-of select= "$AUTOMATON_ID" /><
          xsl:text>_sub</xsl:text><xsl:value-of select= "./@position" /></xsl:with-param>
    </xsl:apply-templates>
  </xsl:for-each>
  <!-- Erzeugen der initialen Instanzen. -->
  <xsl:call-template name="createInitialInstance">
    <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
  </xsl:call-template>
</xsl:template>
<!-- Dieses Template erzeugt einen Zustand mit all seinen Transitionen. Danach wird rekursiv die Erzeugung der
     Zustände gestartet auf die die Transitionen des gerade generierten Zustands verweisen.
     @param COUNTER Gibt an wieviele Rekursionsschritte bereits durchlaufen wurden.
     @param NUMBER_OF_STEPS Anzahl der Rekursionsschritte die durchgeführt werden sollen. Entspricht der
          Zahl der Operanden die Eintretten müssen.
     @param AUTOAMTEN_ID Die ID des Automaten zu dem der Zustand erzeugt wird.
     @param EVENTS Die Events zu denen in diesem Zustand Transitionen erzeugt werden sollen.
     @param ZUSTANDSNAME Der Name des Zustands der erzeugt werden soll.-->
<xsl:template name= "create_any_zustand" >
  <xsl:param name= "COUNTER" select= "'0'" />
  <xsl:param name= "NUMBER_OF_STEPS" />
  <xsl:param name= "AUTOMATON_ID" />
  <xsl:param name= "EVENTS" />
  <xsl:param name= "ZUSTANDNAME" />
  <!-- Erzeugen des Zustands. -->
  <ceda:state detect= "no" duplicate= "yes" >
    <xsl:attribute name= "typ">
      <xsl:choose><xsl:when test= "$COUNTER = 0" >initial</xsl:when>
      <xsl:when test= "$COUNTER &gt; 0 and $COUNTER &lt; $NUMBER_OF_STEPS" >normal</
          xsl:when></xsl:choose>
    </xsl:attribute>
    <xsl:attribute name= "id"><xsl:value-of select= "$ZUSTANDNAME" /></xsl:attribute>
```

```
      <!-- Aufruf zum rekursiven erzeugen der Transitionen. -->
      <xsl:call-template name= "create_any_transition">
        <xsl:with-param name= "COUNTER" select= "'1'" />
        <xsl:with-param name= "EVENTS" select= "$EVENTS" />
        <xsl:with-param name= "IF_LAST_STEP" select= "$NUMBER_OF_STEPS - $COUNTER" />
        <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
        <xsl:with-param name= "BASETARGET" select= "$ZUSTANDNAME" />
      </xsl:call-template>
    </ceda:state>
    <!-- Rekursiver Aufruf. Für jede erzeugte Transition wird ein weiterer Zustand erzeugt. Die Rekursion wird
        abgebrochen wenn die maximale Rekusionstiefe aus NUMBER_OF_STEPS ereicht ist.-->
    <!-- Rekursionsabruch -->
    <xsl:if test= "$COUNTER &lt; $NUMBER_OF_STEPS - 1" >
      <!-- Für jede erzeugte Transition den Ziel-Zustand erstellen. -->
      <xsl:for-each select= "$EVENTS/*" >
        <xsl:variable name= "EVENT" select= "." />
        <xsl:call-template name= "create_any_zustand" >
          <xsl:with-param name= "COUNTER" select= "$COUNTER + 1" />
          <xsl:with-param name= "NUMBER_OF_STEPS" select= "$NUMBER_OF_STEPS" />
          <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
          <xsl:with-param name= "EVENTS" ><xsl:copy-of select= "$EVENTS/*[not(./@position = $
              EVENT/@position)]" /></xsl:with-param>
          <xsl:with-param name= "ZUSTANDNAME" ><xsl:value-of select= "$ZUSTANDNAME" />-<
              xsl:value-of select= "$EVENT/@position" /></xsl:with-param>
        </xsl:call-template>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>
  <!-- Dieses Template wird aufgerufen um Transitionen zu erzeugen. Dabei wird jedes Rekursiv für jedes Element
      aus dem Parameter EVENTS eine Transition mit einem solchen Element als Ziel erzeugt.
      @param COUNTER Enthält die Rekursionstiefe in der sich das Template befindet.
      @param EVENTS Menge der Events zu denen eine Transition erzeugt werden soll. Jede ist das Ziel einer
          Transition.
      @param IF_LAST_STEP Zahl die angibt ob die erzeugten Automaten auf den Endzustand verweisen sollen
          oder zu neuen Zuständen. 1 bedeutet einen Verweis auf den Endzustand. Ein Wert größer 1 sybolisiert
          einen Verweis auf normale Zustände.
      @param AUTOMATON_ID Die ID des Automaten zu dem die Transitionen erzeugt werden sollen.
      @param BASETARGET Mit diesem Parameter wird der Basisname der Zustände übergeben die als Ziele der
          Transitionen verwendet werden. Durch hinzufügen der Position des Events das als Auslöser dient wird das
          jeweilige Ziel angegeben.-->
  <xsl:template name= "create_any_transition">
    <xsl:param name= "COUNTER" select= "'1'" />
    <xsl:param name= "EVENTS" />
    <xsl:param name= "IF_LAST_STEP" />
    <xsl:param name= "AUTOMATON_ID" />
    <xsl:param name= "BASETARGET" />
    <!-- Erzeugen einer Transition -->
    <ceda:transition>
      <!-- Erzeugen des Attributes target. Wenn IF_LAST_STEP gleich 1 ist, wird als Wert die ID des
          Endzustandes angegeben. Anderenfalls wird '-x' an den BASETARGET angehängt und als Ziel
          verwendet. x ist dabei die Position des Events, das als Auslöser der Transition verwendet wird, in
          EVENTS. -->
      <xsl:attribute name= "target">
        <xsl:choose>
        <xsl:when test= "$IF_LAST_STEP &gt; 1" >
          <xsl:value-of select= "$BASETARGET" />-<xsl:value-of select= "$EVENTS/*[position() = $
              COUNTER]/@position" />
        </xsl:when>
        <xsl:when test= "$IF_LAST_STEP = 1" >
          <xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_zustandE</xsl:text>
        </xsl:when>
        </xsl:choose>
      </xsl:attribute>
      <!-- Fallunterscheidung ob das Event ein atomares oder zusammengestztes Event ist. In erstem Fall wird
          eine primitive Transition erzeugt. In zweitem Fall wird eine Transition erzeugt mit einer Referenz auf
          einen anderen Automaten. -->
      <xsl:choose>
      <xsl:when test= "$EVENTS/child::*[position() = $COUNTER][name(.) = 'snoop:Atomic-Event']">
        <xsl:attribute name= "typ">primitiv</xsl:attribute>
        <xsl:copy-of select= "$EVENTS/*[position() = $COUNTER]/*[1]" />
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name= "typ">composite</xsl:attribute>
        <xsl:attribute name= "subautomaton"><xsl:value-of select= "$AUTOMATON_ID" /><xsl:text>_sub
            </xsl:text><xsl:value-of select= "$EVENTS/*[position() = $COUNTER]/@position" /></
            xsl:attribute>
      </xsl:otherwise>
      </xsl:choose>
    </ceda:transition>
    <!-- Rekursiver Aufruf. Solange noch nicht alle Elemente aus EVENTS bearbeitet wurden wird das Template
```

```
        wieder aufgerufen und der COUNTER inkrementiert. -->
    <xsl:if test= "$COUNTER &lt; count($EVENTS/child::*)">
        <xsl:call-template name= "create_any_transition" >
            <xsl:with-param name= "COUNTER" select= "$COUNTER + 1" />
            <xsl:with-param name= "EVENTS" select= "$EVENTS" />
            <xsl:with-param name= "IF_LAST_STEP" select= "$IF_LAST_STEP" />
            <xsl:with-param name= "AUTOMATON_ID" select= "$AUTOMATON_ID" />
            <xsl:with-param name= "BASETARGET" select= "$BASETARGET" />
        </xsl:call-template>
    </xsl:if>
</xsl:template>
<!-- Dieses Template fügt jedem Operanden des bearbeiteten Any-Operators eine Nummer hinzu die ihre Position
     angibt. Dazu ruft sich das Template rekursiv selbst auf.
     @param EVENTNR Die Position des Operanden der mit seiner Positionsnummer versehen werden soll.
          Beginnt standartmäßig mit Eins.
     @param EVENTSX Die Liste der Operanden des Any-Operators. -->
<xsl:template name= "create_event_element" >
    <xsl:param name= "EVENTNR" select= "'1'" />
    <xsl:param name= "EVENTSX" />
    <!-- Fallunterscheidung für alle möglichen Tags die als Operanden auftretten können. -->
    <xsl:choose>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Atomic-Event'" >
        <xsl:element name= "snoop:Atomic-Event" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:And'" >
        <xsl:element name= "snoop:And" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Or'" >
        <xsl:element name= "snoop:Or" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Sequence'" >
        <xsl:element name= "snoop:Sequence" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
    </xsl:element>
    </xsl:when>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Any'" >
        <xsl:element name= "snoop:Any" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when>
    <xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Multi_Occurrences'" >
        <xsl:element name= "snoop:Multi_Occurrences" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when><xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Aperiodic'" >
        <xsl:element name= "snoop:Aperiodic" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when><xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Cumulative_Aperiodic'"
        >
        <xsl:element name= "snoop:Cumulative_Aperiodic" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
        </xsl:element>
    </xsl:when><xsl:when test= "name($EVENTSX[position() = $EVENTNR]) = 'snoop:Periodic'" >
        <xsl:element name= "snoop:Periodic" >
            <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/@*" />
            <xsl:copy-of select= "$EVENTSX[position() = $EVENTNR]/*" />
```

```
      </xsl:element>
   </xsl:when><xsl:when test= "name($EVENTSX[position()␣=␣$EVENTNR])␣=␣'snoop:Cumulative_Periodic'" >
      <xsl:element name= "snoop:Cumulative_Periodic" >
         <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
         <xsl:copy-of select= "$EVENTSX[position()␣=␣$EVENTNR]/@*" />
         <xsl:copy-of select= "$EVENTSX[position()␣=␣$EVENTNR]/*" />
      </xsl:element>
   </xsl:when><xsl:when test= "name($EVENTSX[position()␣=␣$EVENTNR])␣=␣'snoop:Not'" >
      <xsl:element name= "snoop:Not" >
         <xsl:attribute name= "position" ><xsl:value-of select= "$EVENTNR" /></xsl:attribute>
         <xsl:copy-of select= "$EVENTSX[position()␣=␣$EVENTNR]/@*" />
         <xsl:copy-of select= "$EVENTSX[position()␣=␣$EVENTNR]/*" />
      </xsl:element>
   </xsl:when>
   </xsl:choose>
   <!-- Rekursiver Aufruf des Templates sollte nicht der letzte Operator bereits verarbeitet sein. -->
   <xsl:if test= "$EVENTNR␣&lt;␣count($EVENTSX/*)" >
      <xsl:call-template name= "create_event_element" >
         <xsl:with-param name= "EVENTSX" select= "$EVENTSX" />
         <xsl:with-param name= "EVENTNR" select= "$EVENTNR␣+␣1" />
      </xsl:call-template>
   </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

C.3 Weitere XSLT-Stylsheets

C.3.1 Stylesheet „ced-organisation“

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" xmlns:ceda= "http://rewerse.net/
    event-automaton-ml" version= "1.0">
<!-- Über diesen Parameter wird gesteuert welche Aufgabe das Stylesheet ausführen soll. Entweder es löscht mit '
    delete' alle Endzustände oder es gibt die Composite Events mit 'get' aus. Eine dritte Möglichkeit ist das mit '
    deregister' ein gesamter Eventausdruck, bzw. dessen Automaten und Instanzen gelöscht werden.-->
<xsl:param name= "MODE">get</xsl:param>
<!-- Über diesen Parameter kann angegeben werden welcher Eventausdruck entfernt werden soll.-->
<xsl:param name= "ID" />
<!-- Output übernimmt die Spezifizierung der zu erzeugenden Ausgabe. In diesem Fall ein XML-Dokument der
    Version 1.0 in UTF-8-Kodierung mit Einrückungen, einer XML-Deklaration und dem Medien-Typ "text/
    xml".-->
<xsl:output method= "xml" version= "1.0"
    encoding= "UTF-8" indent= "yes"
    cdata-section-elements= ""
    omit-xml-declaration= "no"
    standalone= "yes" media-type= "text/xml" />
<!-- Das Template für die Wurzel. Hier wird der Parameter MODE ausgewertet und das Template aufgerufen das
    sich daraus ergibt. -->
<xsl:template match= "/">
  <xsl:choose>
  <xsl:when test= "$MODE = 'get'">
    <xsl:call-template name="getEvents" />
  </xsl:when>
  <xsl:when test= "$MODE = 'delete'">
    <xsl:call-template name="deleteFinalInstances" />
  </xsl:when>
  <xsl:when test= "$MODE = 'deregister'">
    <xsl:call-template name="deregisterExpression">
      <xsl:with-param name="ID" select="$ID" />
    </xsl:call-template>
  </xsl:when>
  </xsl:choose>
</xsl:template>
<!-- Dieses Template wird im Modus 'get' ausgeführt. Es liest alle Instanzen aus die in einem Detekt-Zustand
    sind. Danach werden sie in ein eca:result umgewandelt.
    ACHTUNG: Der Name der Regel zu der das Event gehört liegt im XML-Dokument nicht vor und muss
        ausserhalb dieses Stylesheet hinzugefügt werden! -->
<xsl:template name= "getEvents">
  <xsl:for-each select= "/ceda:ceda/ceda:instance[./ceda:current-state/@ref = //ceda:state[./@detect='yes']/@id]"
      >
    <eca:answer xmlns:eca="http://www.eca.org/eca-ml" component="event">
      <xsl:attribute name= "internalid"><xsl:value-of select= "./@automaton" /></xsl:attribute>
      <eca:result>
        <!-- Kopiert die Abfolge der Events. -->
        <xsl:for-each select="./ceda:parameterstore/*[./@arrivaltime] | ./ceda:parameterstore//
            ceda:composite-event/*[./@arrivaltime]" >
          <xsl:sort select="./@arrivaltime" />
          <xsl:copy-of select= "." />
        </xsl:for-each>
      </eca:result>
      <!-- Überprüft, ob Variablenbindungen vorliegen. Wenn ja wird ein eca:variable-bindings Element erzeugt
          . -->
      <xsl:if test= "./ceda:variables/eca:variable">
        <eca:variable-bindings>
          <eca:tuple>
            <!-- Kopiert die Variablenbindungen -->
            <xsl:copy-of select= "./ceda:variables/eca:variable" />
          </eca:tuple>
        </eca:variable-bindings>
      </xsl:if>
    </eca:answer>
  </xsl:for-each>
</xsl:template>
<!-- Dieses Template wird im Modus 'delete' ausgeführt. Es Kopiert das Dokument bis auf die Instanzen die in
    einem Endzustand liegen. -->
<xsl:template name= "deleteFinalInstances">
  <ceda:ceda>
    <!-- Kopieren der ursprünglichen Attribute in das neue ceda-Element.-->
    <xsl:copy-of select= "ceda:ceda/@*" />
    <!-- Kopieren der Automatenspezifikationen, da sie unveränderlich sind.-->
    <xsl:copy-of select= "ceda:ceda/ceda:automaton" />
    <!-- Kopiert all die Instanzen, die nicht in einem Endzustand sind. -->
    <xsl:copy-of select= "/ceda:ceda/ceda:instance[not(./ceda:current-state/@ref = //ceda:state[./@typ='ende' or
        ./@typ='abort']/@id)]" />
```

```
    </ceda:ceda>
</xsl:template>
<!-- Dieses Template wird im Modus 'deregister' ausgeführt. Es entfernt den Automaten zu der angegebenen
     Automaten-ID aus dem ceda-Dokument. Dabei werden auch alle Subautomaten und alle Instanzen zu diesen
     Automaten entfernt. -->
<xsl:template name= "deregisterExpression">
    <xsl:param name="ID"/>
    <ceda:ceda>
        <!-- Kopieren der ursprünglichen Attribute in das neue ceda-Element.-->
        <xsl:copy-of select= "ceda:ceda/@*" />
        <!-- Kopieren der Automaten, die nicht zur gewählten ID passen. -->
        <xsl:copy-of select= "ceda:ceda/ceda:automaton[not(starts-with(@id,$ID))]" />
        <!-- Kopiert alle Instanzen, die nicht zur gewälten ID passen. -->
        <xsl:copy-of select= "/ceda:ceda/ceda:instance[not(starts-with(@automaton,$ID))]" />
    </ceda:ceda>
</xsl:template>
</xsl:stylesheet>
```

Anhang D

Java-Quellcodes

D.1 Basisklassen

D.1.1 Klasse Automatons

```
package rewerse.ced;
import java.util.List;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.jdom.Document;
import org.jdom.Content;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.output.Format;
import org.jdom.transform.JDOMSource;
import org.jdom.transform.JDOMResult;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Templates;
import javax.xml.transform.Transformer;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.Result;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;
public class Automatons{
    private File automatonsfile;
    private Document automatonsdom;
    private Templates templates;
    private final String organisationstylsheetfilename = "/WEB-INF/XSLT/ced-organisation.xsl";
    private Templates organisationtemplates;
    private String basepath = "";
    /**
      * @param automatonsfilename
      * @param xsltstylesheet
      * @throws TransformerConfigurationException
      * @throws IOException
      * @throws JDOMException
    */
    public Automatons(String automatonsfilename, String xsltstylesheet, String basepath)throws
          TransformerConfigurationException, IOException, JDOMException{
        java.io.FileInputStream automatonsstream = null;
        this.basepath = basepath;
        try{
            this.automatonsfile = new File(automatonsfilename);
            automatonsstream = new java.io.FileInputStream(this.automatonsfile);
            SAXBuilder saxbuilder = new SAXBuilder(false);
            this.automatonsdom = saxbuilder.build(automatonsstream, this.automatonsfile.getParentFile().toURL().
                  toExternalForm());
            //this.automatonsdom.setBaseURI(this.automatonsfile.getParentFile().toURL().toExternalForm());
```

```
        }//ende try
        finally{
            automatonsstream.close();
        }//ende finally
        File stylesheetfile = new File(xsltstylesheet);
        Source stylesheet = new StreamSource(new FileInputStream(stylesheetfile), stylesheetfile.getParentFile().toURL
            ().toExternalForm()); //File with the XSLT-Template Definitions
        TransformerFactory transformerfactory = TransformerFactory.newInstance(); //Factory for creating Templates-
            Objects
        this.templates = transformerfactory.newTemplates(stylesheet);//The Compiled XSLT-Templates from File "
            stylesheet"
        File organisationstylsheetfile = new File(basepath + organisationstylsheetfilename);
        stylesheet = new StreamSource(new FileInputStream(organisationstylsheetfile), stylesheetfile.getParentFile().
            toURL().toExternalForm()); //File with the XSLT-Template Definitions
        this.organisationtemplates = transformerfactory.newTemplates(stylesheet);//The Compiled XSLT-Templates
            from File "stylesheet"
    }//ende Konstruktor Automatons
    /**
     * @return
    */
    public Document getDOMTree(){
        return this.automatonsdom;
    }//ende Method getDOMTree()
    /**
     * @param doc
    */
    public void setDOMTree(Document doc){
        this.automatonsdom = doc;
    }//end Method setDOMTree
    /**
     * @param automaton
    */
    public void addAutomatons( Document newautomatons ){
        List nodes = newautomatons.getRootElement().cloneContent();
        for (int i = 0; i < nodes.size(); i++){
            this.automatonsdom.getRootElement().addContent((Content)nodes.get(i));
        }//ende for
    }//ende Method addAutomaton(Element)
    /**
     * @throws IOException
    */
    public void serialize() throws IOException{
        java.io.FileOutputStream out = null;
        try{
            XMLOutputter jdomoutputter = new XMLOutputter(Format.getPrettyFormat());
            out = new java.io.FileOutputStream(this.automatonsfile);
            jdomoutputter.output(this.automatonsdom, out);
        }//end try serialize
        finally{
            out.close();
        }//end finally
    }//ende serialize()
    /**
      * @param ausdruck
      * @return
      * @throws TransformerException
    */
    public Document createAutomaton(Element ausdruck, String id)throws TransformerException{
        Source source = new JDOMSource(ausdruck);
        JDOMResult result = new JDOMResult();
        Transformer transformer = this.templates.newTransformer();   //creats a new Transformer
        transformer.setParameter("AUTOMATON_NAME", id);
        transformer.transform(source, result);          //transforms the source
        return result.getDocument();
    }//ende Method createAutomaton(Element)
    /**
    */
    public List getCompositeEvents() throws JDOMException, TransformerException{
        Source source = new JDOMSource(this.automatonsdom);
        JDOMResult result = new JDOMResult();
        Transformer transformer = this.organisationtemplates.newTransformer();
        transformer.setParameter("MODE", "get");
        transformer.transform(source, result);
        return result.getResult();
    }
/**
*/
public void removeCompositeEvents()throws TransformerException{
    Source source = new JDOMSource(this.automatonsdom);
    JDOMResult result = new JDOMResult();
```

```
    Transformer transformer = this.organisationtemplates.newTransformer();//creats a new Transformer
    transformer.setParameter("MODE", "delete");
    transformer.transform(source, result); //transforms the source
    this.automatonsdom = result.getDocument();
}//end Method removeCompositeEvents();
/***/
public void removeEventExpression(String internalid) throws TransformerException{
    //Ich nehme das Root-Element weil er einfach den Pfad zur DTD nicht findet. So wird sie erst gar nicht
        gewsucht.
    Source source = new JDOMSource(this.automatonsdom.getRootElement());
    JDOMResult result = new JDOMResult();
    Transformer transformer = this.organisationtemplates.newTransformer();//creats a new Transformer
    transformer.setParameter("MODE", "deregister");
    transformer.setParameter("ID", internalid);
    transformer.transform(source, result); //transforms the source
    this.automatonsdom = result.getDocument();
}//end Method removeEventExptession(String)
}//end Class Automatons
```

D.1.2 Klasse EventConsumer

```
package rewerse.ced;
import java.io.File;
import java.io.FileInputStream;
import java.net.MalformedURLException;
import java.io.IOException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Templates;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.Source;
import javax.xml.transform.Result;
import javax.xml.transform.stream.StreamSource;
import org.jdom.transform.JDOMSource;
import org.jdom.transform.JDOMResult;
import rewerse.ced.Automatons;
/**
  * @author Sebastian Spautz (sebastian.spautz@freenet.de)
  * @version 0.0.1
*/
public class EventConsumer{
   /* ------------------------------------------------- */
   /* --------------- Class Variables ---------------- */
   /* ------------------------------------------------- */
   /***/
   private Automatons automatons;
   /***/
   private Templates templates;
   /***/
   private File basedirectory;
   /* ------------------------------------------------- */
   /* --------------- Constructors ------------------- */
   /* ------------------------------------------------- */
   /**
      @param automatons
      @param stylesheetPath The Path to the XSLT-Document witch would use in this Consumer to transform the
           Automatons.
      @throws TransformerConfigurationException
   */
   public EventConsumer(Automatons automatons, String basepath) throws TransformerConfigurationException,
        IOException{
      this.automatons = automatons;
      this.basedirectory = new File(basepath);
      /*This Code creats the Templates-Object "this.templates". With this Object the Transformations will do.</p
           >*/
      FileInputStream in = null;
      try{
         File stylesheetfile = new File(this.basedirectory, "WEB-INF/XSLT/ceda-io.xsl");
         in = new FileInputStream(stylesheetfile);
         Source stylesheet = new StreamSource(in, stylesheetfile.getParentFile().toURL().toExternalForm()); //File
              with the XSLT-Template Definitions
         TransformerFactory transformerfactory = TransformerFactory.newInstance(); //Factory for creating
              Templates-Objects
         templates = transformerfactory.newTemplates(stylesheet);//The Compiled XSLT-Templates from File "
              stylesheet"
      }finally{
         in.close();
      }
   }//ende Constructor(ServletContext)
   /* ------------------------------------------------- */
   /* --------------- Public Methods ----------------- */
   /* ------------------------------------------------- */
   /**
      @param primitivevent
      @throws TransformerException
   */
   public synchronized void transformAutomaton() throws TransformerException/*, MalformedURLException*/{
        Source old_automaton_status = new JDOMSource(this.automatons.getDOMTree().getRootElement());
        JDOMResult new_automaton_status = new JDOMResult();
        Transformer transformer = this.templates.newTransformer(); //creats a new Transformer
        transformer.transform(old_automaton_status, new_automaton_status);//transforms the source
        this.automatons.setDOMTree(new_automaton_status.getDocument());
   }//ende transformAutomaton()
}//ende Class Consumer
```

D.1.3 Klasse ApplicationData

```
package rewerse.ced;
import java.util.Properties;
import java.io.*;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerConfigurationException;
import org.xml.sax.SAXException;
import rewerse.ced.Automatons;
public class ApplicationData{
   /* ------------------------------------------------ */
   /* ------------- Fields and Variables ------------- */
   /* ------------------------------------------------ */
   private static volatile Automatons automatons = null;
   private static volatile Properties keyanswerurlmap = null;
   private static volatile Properties keyrulemap = null;
   private static String basepath;
   /* ------------------------------------------------ */
   /* ----------------- Methods -------------------- */
   /* ------------------------------------------------ */
   /** */
   public static void initData(String basepath, String automatonfilename, String xsltstylesheet)throws
         TransformerConfigurationException, org.jdom.JDOMException, IOException{
      automatons = new Automatons(basepath + "/" + automatonfilename, basepath + "/" + xsltstylesheet, basepath
            );
      ApplicationData.basepath = basepath;
      keyanswerurlmap = new Properties();
      keyrulemap = new Properties();
      keyanswerurlmap.loadFromXML(new FileInputStream(basepath + "/WEB-INF/conf/urlmap.xml"));
      keyrulemap.loadFromXML(new FileInputStream(basepath + "/WEB-INF/conf/rulemap.xml"));
   }//ende initData()
   /** */
   public static Automatons getAutomatons(){
      return automatons;
   }//ende getAutomatons()
   /** */
   public static boolean debug(){
      return true;
   }//end debug()
   public static String loadAnswerURL(String internalid){
      return keyanswerurlmap.getProperty(internalid);
   }
   public static void storeAnswerURL(String internalid, String url)throws IOException{
      keyanswerurlmap.setProperty(internalid, url);
      FileOutputStream out = null;
      try{
         out = new FileOutputStream(basepath + "/WEB-INF/conf/urlmap.xml");
         keyanswerurlmap.storeToXML(out,
            "Mapping_from_internal_Automaton-ID's_to_the_URL_where_the_detected_Events_will_send.", "UTF-8"
               );
      }finally{ out.close(); }
   }
   public static String loadRuleID(String internalid){
      return keyrulemap.getProperty(internalid);
   }
   public static void storeRuleID(String internalid, String ruleid) throws IOException{
      keyrulemap.setProperty(internalid, ruleid);
      FileOutputStream out = null;
      try{
         out = new FileOutputStream(basepath + "/WEB-INF/conf/rulemap.xml");
         keyrulemap.storeToXML(out,
            "Mapping_from_internal_Automaton-ID's_to_the_external_ECA-Rule-Id's", "UTF-8");
      }finally{ out.close(); }
   }
}//end Class ApplicationData
```

D.2 Servlets

D.2.1 Klasse CEDServlet

```
package rewerse.ced.servlets;
import java.text.SimpleDateFormat;
import java.io.IOException;
import javax.xml.messaging.JAXMServlet;
import javax.xml.messaging.ReqRespListener;
import javax.xml.transform.TransformerConfigurationException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPException;
import rewerse.ced.ApplicationData;
import rewerse.ced.debug.*;
/** Dieses Servlet bildet die Basis für die SOAP-Kommunikation. Dazu wird die JAXM-API importiert. Diese
    stellt eine Servlet-Klasse bereit das die Kommunikation abwickelt. Darüber hinaus werden einige Aufgaben
    implementiert die allen Servlets gemein sind.
  * @author Sebastian Spautz (sebastian.spautz@freenet.de)
  * @version 0.0.1 */
public abstract class CEDServlet extends javax.xml.messaging.JAXMServlet implements javax.xml.messaging.
    ReqRespListener {
  /***/
  protected SimpleDateFormat dateformat;
  /***/
  protected ServletContext servletcontext;
  /* ---------------------------- */
  /* ---- Servlet Methods ---- */
  /* ---------------------------- */
  /** */
  public void init(ServletConfig servletconfig) throws ServletException{
    super.init(servletconfig); //do the default things from JAXMServlet, HttpServlet and Servlet.
    this.dateformat = new SimpleDateFormat("MM/dd/yyyy_kk/mm");
    //includes the Context-Path for all Fileoperations.
    this.servletcontext = servletconfig.getServletContext();
    try{//try to load the Application Data
      if (ApplicationData.getAutomatons() == null) ApplicationData.initData(this.servletcontext.getRealPath("."),
          "XML/automatons.xml", "WEB-INF/XSLT/create_automaton.xsl");
    }//end try
    catch (IOException ioe){
      if (ApplicationData.debug())ioe.printStackTrace(System.err);
      log ("CED_Composite_Event_Servlet:_A_Exception_by_init()_is_occured_(Files_not_Found):\n_" + ioe);
      throw new ServletException(ioe);
    }//end catch IOException
    catch (org.jdom.JDOMException jdome){
      if (ApplicationData.debug())jdome.printStackTrace(System.err);
      log ("CED_Composite_Event_Servlet:_A_Exception_by_init()_is_occured_(Can_not_parse_the_XML-File):\n_
          ____" + jdome);
      throw new ServletException(jdome);
    }//end catch JDOMException
    catch (TransformerConfigurationException tce){
      if (ApplicationData.debug())tce.printStackTrace(System.err);
      log ("CED_Composite_Event_Servlet:_A_Exception_by_init()_is_occured_(Transformer_can_not_be_initalised)
          :\n_" + tce);
      throw new ServletException(tce);
    }//end catch TransformerConfigurationException
  }//end of Method init
  /* Diese Methode verarbeitet Get-Anfragen indem ein kurzer Hinweis im HTML-Format zurück gegeben wird.*/
  /**This Method reacts on the incoming of a Http-Get-Request. A short HTML-Message whould send.
    @param req The incoming Request.
    @param resp The chanel for the outgoing Response. */
  public void doGet(HttpServletRequest req, HttpServletResponse resp){
    try{
      resp.setContentType("text/html");
      resp.getWriter().println("<html><body><h1>SOAP-Fault</h1><p>This_URL_is_reserved_vor_SOAP-
          Messages_using_POST-Requests.</p></body></html>");
      resp.getWriter().println("<html><body><h1>SOAP-Fault</h1><p>Diese_URL_ist_reserviert_f?P-
          Nachrichten_mittels_Post-Anfragen.</p></body></html>");
    }catch(Exception e){
      System.err.println("Error_while_processing_a_GET-Request");
    }//end Exception
  }//ende Method doGet(HTTPServletRequelst, HTTPServletResponse)
  /* ------------------------------- */
```

```
    /* ----- New and other Methods ----- */
    /* ------------------------------------ */
    /** This Method creates a simple SOAP-Fault-Message.
        @return A SOAP-Fault-Message. */
    protected SOAPMessage createFaultMessage(){
        SOAPMessage faultresult = null;
        try{
            MessageFactory factory = MessageFactory.newInstance();
            faultresult = factory.createMessage();
            SOAPBody resultbody = faultresult.getSOAPPart().getEnvelope().getBody();
            resultbody.addFault();
        }catch(SOAPException soape){
            log("CED Primitiv Event Servlet: A SOAP-Exception has occured: \n" + soape);
            return null;
        }//ende catch SOAPException
        return faultresult;
    }//end Method createFaultMessage()
}//end Class CEDServlet
```

D.2.2 Klasse RegisterEventExpressionServlet

```
package rewerse.ced.servlets;
//Imports aus J2SE
import java.util.Date;
import java.util.Iterator;
import java.io.IOException;
//Imports aus Servlet-API
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
//Imports aus SAAJ-API
import javax.xml.soap.*;
//Imports aus JAXP-API
import javax.xml.transform.TransformerException;
//Imports aus DOM
import org.w3c.dom.Node;
import org.w3c.dom.Element;
//Interne Imports
import rewerse.ced.servlets.CEDServlet;
import rewerse.ced.ApplicationData;
import rewerse.ced.debug.*;
/** This Servlet Contains the Registration for new SNOOP-Event-Expressions. When starts it initialise the
      Application Data with the Class ApplicationData. On input of a SOAP-Message it transform the Expression
      into Automatons and add this to the older ones.
  * @author Sebastian Spautz (sebastian.spautz@freenet.de)
  * @version 0.1
*/
public class RegisterEventExpressionServlet extends CEDServlet {
   /* ---------------------------------- */
   /* --------- Class Variables --------- */
   /* ---------------------------------- */
   /***/
   private volatile int automatoncounter = 1;
   /* ---------------------------------- */
   /* ---------- Servlet Methods -------- */
   /* ---------------------------------- */
   /** This Method describes the Funktion of this Servlet. This overrides the generic Method in JAXMServlet
      repectivly HttpServlet.
      @return A describing String.
   */
   public String getServletInfo(){
      return "Servlet_for_registration_Complex_Event_Spezifications.";
   }//end Method getServletInfo()
   /** This Method from Class Servlet initialise this Servlet.
      @param servletconfig The Configuration of the Servlet. This Parameter is set from the Servlet-Container.
      @throws ServletException This Excpetion will thrown when the Initialisation not works.
   */
   public void init(ServletConfig servletconfig) throws ServletException{
      super.init(servletconfig); //do the default things from CEDServlet, JAXMServlet, HttpServlet and Servlet.
   }//end Method init(ServletConfig)
/* ------------------------------------------------ */
/* --- Methods from Interface ReqRespListener --- */
/* ------------------------------------------------ */
/** This Method from the Interface ReqRespListener work on the received SOAP-Message. It starts the detection of
      composite Events for each XML-Fragment, as a primitiv Event, in the SOAP-Body. Also a Response will
      createt.
  @param message The Received SOAP-Message.
  @return The Response for the incoming Message.
*/
public SOAPMessage onMessage(SOAPMessage message){
   SOAPHeader messageheader = null;//Header of received SOAP-Message
   SOAPMessage result = null; //Answer for received SOAP-Message
   Iterator expressioniterator = null;
   //Try to get the SOAP-Body and initial the answer Message
   try{
      messageheader = message.getSOAPPart().getEnvelope().getHeader();//get SOAP-Header
      //create new SOAP-Message
      result = MessageFactory.newInstance().createMessage();
      //reads the Event Expressions from the SOAP-body
      expressioniterator = message.getSOAPPart().getEnvelope().getBody().getChildElements(message.getSOAPPart().
            getEnvelope().createName("Eventdeclaration", "snoop", "http://rewerse.net/snoop-expression-ml"));
   }
   catch (SOAPException soape){
      if (ApplicationData.debug())soape.printStackTrace(System.err);
      log("CED_Composite_Event_Servlet:_It_is_not_a_correct_SOAP-Message_" + soape);
      //Creats a SOAP-Fault-Message because the arrived Message is not correct
      return this.createFaultMessage();
   }//ende try-catch for SOAP-Exception
   //reads the Event Expressions from the SOAP-body
   for (;expressioniterator.hasNext();){
```

```
        Element expression = (Element) expressioniterator.next();
        try{
            Date arrivaltimestamp = new Date();
            //Creates a new ID for the Automaton
            String idstring = this.dateformat.format(arrivaltimestamp) + "_#" + this.automatoncounter++;
            String ruleid = ((Element)messageheader.getChildElements(message.getSOAPPart().getEnvelope().createName
                ("rule-id")).next()).getTextContent();
            String answerurl = ((Element)messageheader.getChildElements(message.getSOAPPart().getEnvelope().
                createName("answer-url")).next()).getTextContent();
            //default Variable-Bindings
            Iterator variableIterator = messageheader.getChildElements(message.getSOAPPart().getEnvelope().createName
                ("variable-bindings", "eca", "http://www.eca.org/eca-ml"));
            //Aufruf der Registrierungsmethode
            this.register(expression, idstring, ruleid, answerurl, variableIterator);
            //create a registration-Element and put it to the Answer-Message
            SOAPElement registrationelement = SOAPFactory.newInstance().createElement("registration", "ced", "http://
                rewerse.net/ced-organisation-ml");
            registrationelement.addChildElement("rule-id", "ced", "http://rewerse.net/ced-organisation-ml").
                addTextNode( ((Element)messageheader.getChildElements(message.getSOAPPart().getEnvelope().
                createName("rule-id")).next()).getTextContent() );
            registrationelement.addChildElement("internal-id", "ced", "http://rewerse.net/ced-organisation-ml").
                addTextNode(idstring);
            registrationelement.addChildElement("timestamp", "ced", "http://rewerse.net/ced-organisation-ml").
                addTextNode(this.dateformat.format(arrivaltimestamp));
            result.getSOAPPart().getEnvelope().getBody().addChildElement(registrationelement);
        }//end try
        catch (IOException ioe){
            if (ApplicationData.debug())ioe.printStackTrace(System.err);
            log("CED_Composite_Event_Servlet:_Can_not_serialize_the_Automaton-XML-File_or_can_not_write_the_
                Configuration.\n" + ioe);
            continue;
        }//end catch IOException
        catch (TransformerException te){
            if (ApplicationData.debug())te.printStackTrace(System.err);
            log("CED_Composite_Event_Servlet:__\n" + te);
            continue;
        }//end catch TransformerException
        catch (SOAPException soape){//Creats a SOAP-Fault-Message because the arrived Message is not correct
            if (ApplicationData.debug())soape.printStackTrace(System.err);
            log("CED_Composite_Event_Servlet:_Can't_create_a_Answer:___\n" + soape);
            return this.createFaultMessage();
        }//ende try-catch for SOAP-Exception
    }//ende for all Events in the SOAP-Message
    //Returns a Result Message with the ID's of the registratet Expressions
    return result;
}//end Method onMessage(SOAPMessage)
/**This Method transforms the Eventexpression to an Automaton and register the Anserwer-URL and the Rule-ID
     from the ECA-Engine in the Config-Files.
    @param eventexpression The Eventexpression as an DOM-Element.
    @param id The internal ID for the Eventexpression.
    @param ruleid The ID for the ECA-Rule.
    @param answerurl The URL to send the detected composite Events.
    @param variabletuples The Tuples with Variablebindings as List of DOM-Elements.*/
private void register(Element eventexpression, String id, String ruleid, String answerurl,Iterator variabletuples)
        throws IOException, TransformerException{
    //stores the Id of the Rule from wher the Eventexpression are.
    ApplicationData.storeRuleID(id, ruleid);
    //stores the Answer URL for the new Event Expression
    ApplicationData.storeAnswerURL(id, answerurl);
    if ( variabletuples.hasNext() ){
        Element variables = (Element) variabletuples.next();
        eventexpression.appendChild(variables);
    }
    //creates the JDOM-Representation of the Element
    org.jdom.input.DOMBuilder builder = new org.jdom.input.DOMBuilder();
    org.jdom.Element jdomexpression = builder.build((Element)eventexpression);
    //Transform the Event Expression to an Automaton
    org.jdom.Document newautomatons = ApplicationData.getAutomatons().createAutomaton(jdomexpression, id);
    //add the new Automatons into the ceda-Document
    ApplicationData.getAutomatons().addAutomatons(newautomatons);
    //save the new Situation
    ApplicationData.getAutomatons().serialize();
}//ende Method register
}//end Class ReceiveComplexEventServlet
```

D.2.3 Klasse ReceiveAtomicEventServlet

```
package rewerse.ced.servlets;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import org.w3c.dom.Node;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.soap.*;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;
import rewerse.ced.servlets.CEDServlet;
import rewerse.ced.ApplicationData;
import rewerse.ced.EventConsumer;
import rewerse.ced.debug.*;
/**
  * @author Sebastian Spautz (sebastian.spautz@freenet.de)
  * @version 0.0.4
*/
public class ReceiveAtomicEventServlet extends CEDServlet {
   /* ------------------------------------------------ */
   /* ---------------- Class Variables --------------- */
   /* ------------------------------------------------ */
   /***/
   private EventConsumer consumer;
   /* ------------------------------------------------ */
   /* ---------------- Servlet Methods --------------- */
   /* ------------------------------------------------ */
   /** This Method describes the Funktion of this Servlet. This overrides the generic Method in JAXMServlet
        respectivly HttpServlet.
      @return A describing String.
   */
   public String getServletInfo(){
      return "Servlet_for_receiving_primitiv_Events_and_give_them_to_the_detection_Unit_for_compsite_Events.";
   }//end Method getServletInfo()
   /**
      @param servletconfig
      @throws ServletException
   */
   public void init(ServletConfig servletconfig) throws ServletException{
      super.init(servletconfig);
      try{//try to create the Event Consumer
         this.consumer = new EventConsumer(ApplicationData.getAutomatons(), this.servletcontext.getRealPath(".")
              );
      }//end try
      catch (IOException ioe){
         if (ApplicationData.debug())ioe.printStackTrace(System.err);
         log ("CED_Primitiv_Event_Servlet:_A_Exception_by_init()_is_occured_(Files_not_Found):\n_" + ioe);
         throw new ServletException(ioe);
      }//end catch IOException
      catch (TransformerConfigurationException tce){
         if (ApplicationData.debug())tce.printStackTrace(System.err);
         log ("CED_Primitiv_Event_Servlet:_A_Exception_by_init()_is_occured_(Transformer_can_not_be_initalised):\n
              _" + tce);
         throw new ServletException(tce);
      }//end catch TransformerConfigurationException
   }//end Method init(ServletConfig)
/* ------------------------------------------------ */
/* --- Methods from Interface ReqRespListener --- */
/* ------------------------------------------------ */
/** This Method from the Interface ReqRespListener work on the received SOAP-Message. It starts the detection of
      composite Events for each XML-Fragment, as a primitiv Event, in the SOAP-Body. Also a Response will
      createt.
   @param message The Received SOAP-Message.
   @return The Response for the incoming Message.*/
public SOAPMessage onMessage(SOAPMessage message){
   SOAPBody messagebody = null; //Body of received SOAP-Message
```

```
    SOAPMessage result = null; //Answer for received SOAP-Message
    try{
        //Get the SOAP-Body Elements from the Message.
        messagebody = message.getSOAPPart().getEnvelope().getBody();
        //Creats the Answer Message.*/
        result = MessageFactory.newInstance().createMessage();
        SOAPElement responseelement = SOAPFactory.newInstance().createElement("response", "ced", "http://rewerse.
            net/ced-organisation-ml").addTextNode("Receive Atomar Events!");
        result.getSOAPPart().getEnvelope().getBody().addChildElement(responseelement);
    }
    catch (SOAPException soape){
        if (ApplicationData.debug())soape.printStackTrace(System.err);
        log("CED Primitiv Event Servlet: It is not a correct SOAP-Message " + soape);
        //Creats a SOAP-Fault-Message because the arrived Message is not correct SOAP
        return this.createFaultMessage();
    }//ende try-catch for SOAP-Exception
    //Reads the Events from the SOAP-Message (each XML-Fragment insite the SOAP-Body shout be one).
    Iterator eventiterator = messagebody.getChildElements();
    for (;eventiterator.hasNext();){
        Node event = (Node) eventiterator.next();
        if (event instanceof SOAPBodyElement){//only Element-Nodes are from Interesting
            //Creates the XML-File with the Event. With this File the XSLT-Stylesheet can load the Event.
            try{
                this.parseEvent((Element)event);
            }//ende try for serializing a Event
            catch (ParserConfigurationException pce){
                log("CED Primitiv Event Servlet: Can't generate the Event-XML-File!\n   " + pce);
                if (ApplicationData.debug()) System.out.println(pce);
                return this.createFaultMessage();
            }
            catch (java.io.IOException ioe){
                log("CED Primitiv Event Servlet: Can't save the Event to File!\n   " + ioe);
                if (ApplicationData.debug()) System.out.println(ioe);
                return this.createFaultMessage();
            }//end catch MalformedURLException
            //The transformation of the Automatons with the Event is start hier.
            try{
                this.consume();
            }//ende try transform Automaton
            catch (TransformerException te){
                if (ApplicationData.debug())te.printStackTrace(System.err);
                log("CED Primitiv Event Servlet: Transformation does not work!\n" + te);
                continue;
            }//ende catch TransformerException
            catch (IOException ioe){
                if (ApplicationData.debug())ioe.printStackTrace(System.err);
                log("CED Primitiv Event Servlet: Can't write the new Automatonfile!\n" + ioe);
                continue;
            }//end catch IOException
            catch (org.jdom.JDOMException jdome){
                if (ApplicationData.debug())jdome.printStackTrace(System.err);
                log("CED Primitiv Event Servlet: Can't read the Events from CEDA-Document" + jdome);
                continue;
            }
            catch (SOAPException soape){
                if (ApplicationData.debug())soape.printStackTrace(System.err);
                log("CED Primitiv Event Servlet: Can't send a detected Event" + soape);
                return this.createFaultMessage();
            }//ende try-catch for SOAP-Exception
        }//ende if Child is SOAPElement
    }//ende for all Events in the SOAP-Message
    /*Returns the Answer-Message*/
    return result;
}//end Method onMessage(SOAPMessage)
/** This Method creates a Eventfile for the XML-Element in the argument.
     @param event the DOM-Element with the atomic Event.
     @throws ParserConfigurationException
     @throws IOException
*/
private void parseEvent(Element event)throws ParserConfigurationException, IOException{
    FileOutputStream out = null;
    try{
        /*create the DOM-Document with on of the Events from the SOAP-Message.*/
        DocumentBuilderFactory domfactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = domfactory.newDocumentBuilder();
        Document eventdoc = builder.newDocument();
        //Creates an attribute an add it to the Event
        event.setAttribute("arrivaltime", this.dateformat.format(new Date()));
        //Add's the Event-Node to the new XML-Document
        eventdoc.appendChild(eventdoc.importNode(event, true));
```

```
        //Save the Event in File 'Event.xml' with JDOM.
        org.jdom.output.XMLOutputter jdomoutputter = new org.jdom.output.XMLOutputter();
        org.jdom.input.DOMBuilder jdombuilder = new org.jdom.input.DOMBuilder();
        out = new FileOutputStream(new File(this.servletcontext.getRealPath("WEB-INF/XSLT/Event.xml")));
        jdomoutputter.output(jdombuilder.build(eventdoc),out);
    }
    finally{
        try{
            out.close();
        }catch(Exception e){
            if (ApplicationData.debug())e.printStackTrace(System.out);
            log("CED_Primitiv_Event_Servlet:_Can't_Close_the_File_with_the_Event!\n" + e);
        }
    }
}//end Method parseEvent
/** This Method starts the Transformation of the Automatons. The detected Events where send to the ECA-Engine
    .
     @throws TransformerException
     @throws JDOMException
     @throws IOException
     @throws SOAPException
*/
private void consume() throws TransformerException, org.jdom.JDOMException, IOException, SOAPException{
    //Transforms the Automatons
    this.consumer.transformAutomaton();
    //send new Composite Events to ECA-Engines
    List compositeevents = ApplicationData.getAutomatons().getCompositeEvents();
    for (int i = 0; i < compositeevents.size(); i++){
        org.jdom.Element compositeevent = (org.jdom.Element) compositeevents.get(i);
        String id = compositeevent.getAttribute("internalid").getValue();
        String ruleid = ApplicationData.loadRuleID(id);
        if (ruleid != null){
            compositeevent.setAttribute("ref", ruleid);
            compositeevent.removeAttribute("internalid");
            java.net.URL endpoint = new java.net.URL(ApplicationData.loadAnswerURL(id));
            SOAPMessage eventdetectmessage = MessageFactory.newInstance().createMessage();
            Document eventdom = new org.jdom.output.DOMOutputter().output(new org.jdom.Document(
                    compositeevent));
            eventdetectmessage.getSOAPPart().getEnvelope().getBody().appendChild( eventdetectmessage.getSOAPPart().
                    importNode(eventdom.getDocumentElement(), true) );
            SOAPConnection soapconn = SOAPConnectionFactory.newInstance().createConnection();
            SOAPMessage response = soapconn.call(eventdetectmessage, endpoint);
            soapconn.close();
        }//end if no ruleid
    }//end for all Composite Events
    //remove States with detected Events
    ApplicationData.getAutomatons().removeCompositeEvents();
    //save the new Situation
    ApplicationData.getAutomatons().serialize();
}//end Method consume();
}//ende Class ReceiveEventServlet
```

D.2.4 Klasse DeregisterEventExpressionServlet

```
package rewerse.ced.servlets;
//Imports aus J2SE
import java.util.Date;
import java.util.Iterator;
import java.util.Vector;
import java.io.IOException;
//Imports aus Servlet-API
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
//Imports aus SAAJ-API
import javax.xml.soap.*;
//Imports aus JAXP-API
import javax.xml.transform.TransformerException;
//Imports aus DOM
import org.w3c.dom.Node;
import org.w3c.dom.Element;
//Interne Imports
import rewerse.ced.servlets.CEDServlet;
import rewerse.ced.ApplicationData;
import rewerse.ced.debug.*;
/** This Servlet Contains the Deregistration for new SNOOP-Event-Expressions. When starts it
    initialise the Application Data with the Class ApplicationData. On input of a SOAP-Message
    it reads the <rule-id>-Elements from the SOAP-Header and deregister this Evnetexpresseions.
  * @author Sebastian Spautz (sebastian.spautz@freenet.de)
  * @version 0.0.1
*/
public class DeregisterEventexpressionServlet extends CEDServlet {
/* ------------------------------------ */
/* ---------- Servlet Methods ---------- */
/* ------------------------------------ */
/**This Method describes the Funktion of this Servlet. This overrides the generic Method in JAXMServlet
   repectivly HttpServlet.
   @return A describing String.
*/
public String getServletInfo(){
   return "Servlet for deregistration of Complex Event Expressions";
}//end Method getServletInfo()
/**This Method from Class Servlet initialise this Servlet.
   @param servletconfig The Configuration of the Servlet. This Parameter is set from the Servlet-Container.
   @throws ServletException This Excpetion will thrown when the Initialisation not works.
*/
public void init(ServletConfig servletconfig) throws ServletException{
   super.init(servletconfig); //do the default things from CEDServlet, JAXMServlet, HttpServlet and Servlet.
}//end Method init(ServletConfig)
/* ------------------------------------------------ */
/* --- Methods from Interface ReqRespListener --- */
/* ------------------------------------------------ */
/**
   @param message The Received SOAP-Message.
   @return The Response for the incoming Message.
*/
public SOAPMessage onMessage(SOAPMessage message){
   SOAPHeader messageheader = null;//Header of received SOAP-Message
   SOAPMessage result = null; //Answer for received SOAP-Message
   Iterator messageelementiterator = null;
   //Try to get the SOAP-Body and initial the answer Message
   try{
      messageheader = message.getSOAPPart().getEnvelope().getHeader();//get SOAP-Header
      //create new SOAP-Message
      result = MessageFactory.newInstance().createMessage();
      SOAPElement responseelement = SOAPFactory.newInstance().createElement("response", "ced", "http://rewerse.
            net/ced-organisation-ml").addTextNode("Derigister Event Expressions!");
      result.getSOAPPart().getEnvelope().getBody().addChildElement(responseelement);
      //reads the Event Expressions from the SOAP-body
      messageelementiterator = messageheader.getChildElements(message.getSOAPPart().getEnvelope().createName("
            rule-id"));
   }
   catch (SOAPException soape){
      if (ApplicationData.debug())soape.printStackTrace(System.err);
      log("CED Composite Event Deregistration Servlet: It is not a correct SOAP-Message " + soape);
      //Creats a SOAP-Fault-Message because the arrived Message is not correct
      return this.createFaultMessage();
   }//ende try-catch for SOAP-Exception
   //Starts the Deregistration for each Element in the SOAP-Body
   for (;messageelementiterator.hasNext();){
      Element expression = (Element) messageelementiterator.next();
      String externalid = expression.getTextContent();
      try{
```

```
            this.deregistration(externalid);
        }catch(TransformerException te){
            if (ApplicationData.debug())te.printStackTrace(System.err);
            log("CED_Composite_Event_Deregistration_Servlet:_Can_not_deregistration_Event_Expression_" + te);
            //Creats a SOAP-Fault-Message because the arrived Message is not correct
            return this.createFaultMessage();
        }catch(IOException ioe){
            if (ApplicationData.debug())ioe.printStackTrace(System.err);
            log("CED_Composite_Event_Deregistration_Servlet:_Can_not_write_deregitration_to_File" + ioe);
            //Creats a SOAP-Fault-Message because the arrived Message is not correct
            return this.createFaultMessage();
        }
    }//ende for all Events in the SOAP-Message
    //Returns a Result Message with the ID's of the deregistratet Expressions
    return result;
}//end Method onMessage(SOAPMessage)
/* ------------------------------------------ */
/* -------------- New Methods --------------- */
/* ------------------------------------------ */
/**
    @param id Die ID der ECA-Regel deren Eventausdruck aus der Snoop-Engine entfernt werden soll.*/
private void deregistration(String ruleid) throws TransformerException, IOException{
    Vector<String> eventids = ApplicationData.loadEventIDs(ruleid);
    if( eventids != null){
        for (int i = 0; i < eventids.size(); i++){
            String eventid = eventids.elementAt(i);
            //Deleting the ID's from the Config-Files.
            ApplicationData.deleteRuleID(eventid);
            //Remove from Automatons and Instances.
            ApplicationData.getAutomatons().removeEventExpression(eventid);
            //Save the new Situation.
            ApplicationData.getAutomatons().serialize();
        }
    }
}//end Method deregistration(String)
}//end Class ReceiveComplexEventServlet
```

Literaturverzeichnis

[AAB+05] José Júlio Alferes, Ricardo Amador, Erik Behrends, Mikael Berndtsson, François Bry, Gihan Dawelbait, Andreas Doms, Michael Eckert, Oliver Fritzen, Wolfgang May, Paula Lavinia Pătrânjan, Loic Royer, Franz Schenk, and Michael Schröder. Specification of a model, language and architecture for evolution and reactivity. Technical Report I5-D4, REWERSE EU FP6 NoE, 2005. Available at http://www.rewerse.net.

[AAM05] José Júlio Alferes, Ricardo Amador, and Wolfgang May. A general language for evolution and reactivity in the semantic web. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, number 3703, pages 101–115. Springer, 2005.

[BFMS05] Erik Behrends, Oliver Fritzen, Wolfgang May, and Daniel Schubert. An eca engine for deploying heterogenous component languages in the semantic web. Technical report, Institut für Informatik, Universität Göttingen, 2005.

[CKAK94] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 606–617. Morgan Kaufmann, 1994.

[MAA05] Wolfgang May, José Júlio Alferes, and Ricardo Amador. An ontology- and resources-based approach to evolution and reactivity in the semantic web. In *Ontologies, Databases and Semantics (ODBASE)*, number 3761, pages 1553–1570. Springer, 2005.

[Mis91] Deepak Mishra. Snoop: An event specification language for active database systems. Master's thesis, University of Florida, August 1991.

[Suna] Sun Microsystems, Inc. *J2EE: Java Servlet Technology.* http://java.sun.com/products/servlet/.

[Sunb] Sun Microsystems, Inc. *Java 5 Standard Edition.* http://java.sun.com/j2se/.

[W3C99a] W3C. *XML Path Language (XPath)*, November 1999. http://www.w3.org/TR/xpath.

[W3C99b] W3C. *XSL Transformations (XSLT)*, November 1999. http://www.w3.org/TR/xslt.

[W3C04] W3C. *Extensible Markup Language (XML) 1.0 (Third Edition)*, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.

Listings