JOHANNES KEPLER
UNIVERSITÄT LINZ

Eingereicht von
Katharina Kaiser, BSc

Angefertigt am
Institut für Wirtschaftsinformatik – Data & Knowledge Engineering

Betreuer
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Mitbetreuung
Dr. Christoph Schütz

Dezember 2016

Automatisierung von mehrstufigen Geschäftsprozessen mittels SCXML, XQuery und RestXQ

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

JOHANNES KEPLER UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 1. Dezember 2016

Katharina Kaiser

Abstract

Modern business process management systems (BPMS) support the flow of business processes through their partial automation. Multilevel business processes using Multilevel Business Artifact (MBA) allow a compact representation of business processes at the different levels of a company as well as connections between processes of different levels. The use of MBAs for the automation of multi-stage business processes requires them to be in a machine-readable form. State Chart XML (SCXML) is a XML-based notation for state charts which provides a way to represent multilevel business processes. This thesis describes the implementation of an SCXML interpreter for the automation of multilevel business processes. It is implemented using XQuery, a standardized language for querying XML documents. Furthermore, the interface for RESTful web services in XQuery, RESTXQ, was used in order to implement iterative execution of tasks. This was needed because of the declarative nature of XQuery which does not allow to order update operations in a timely manner.

Kurzfassung

Moderne Geschäftsprozessverwaltungssysteme (Business Process Management Systems, BPMS) unterstützen den Ablauf betrieblicher Geschäftsprozesse durch deren teilweise Automatisierung. Mehrstufige Geschäftsprozesse, mittels Multilevel-Business Artifact (MBA) modelliert, erlauben die kompakte Darstellung von Geschäftsprozessen auf den unterschiedlichen Ebenen eines Unternehmens sowie Zusammenhänge zwischen den Abläufen auf den einzelnen Ebenen. Der Einsatz von MBAs zur Automatisierung mehrstufiger Geschäftsprozesse bedarf einer maschinenlesbaren Darstellungsform. State Chart XML (SCXML), eine XML-basierte Notation für Zustandsautomaten, bietet eine Möglichkeit, mehrstufige Geschäftsprozesse für die Automatisierung abzubilden. Die vorliegende Arbeit präsentiert eine Implementierung eines SCXML-Interpreters zur Automatisierung mehrstufiger Geschäftsprozesse. Die Implementierung des SCXML-Interpreters erfolgt in XQuery, die standardisierte Abfrage- und Programmiersprache für XML-Dokumente, unter Verwendung von RESTXQ, einer Schnittstelle für die Erstellung von RESTful Web Services mittels XQuery. Die Verwendung von RESTXQ ermöglicht die Umsetzung iterativer Abläufe trotz deklarativer Natur von XQuery, die keine Reihenfolge von Update-Operationen vorsieht.

Inhaltsverzeichnis

1 Einführung **1**

2 Hintergrund **3**
2.1 Geschäftsprozessautomatisierung 3
2.2 Mehrstufige Geschäftsprozesse 4
2.3 State Chart XML 8
2.3.1 Sprachelemente 8
2.3.2 SCXML-Interpreter 10
2.4 XQuery und RESTXQ 12

3 Process Engine und Process Environment **14**
3.1 Systemüberblick 14
3.2 Ablauf der SCXML-Interpretation (Process Engine) 16
3.3 Interne Module zur SCXML-Interpretation 25
3.3.1 Verarbeitung von SCXML-Dokumenten 27
3.3.2 Verwaltung von Multilevel Business Artifacts 51
3.3.3 Sonstige Module 65
3.4 Logging 65
3.5 Process Environment 67
3.6 Einschränkungen 71

4 Worklist Handler **73**
4.1 Funktionen der REST-Schnittstelle 73
4.2 Grafische Benutzungsschnittstelle 76

5 Fazit und Ausblick **86**

6 Literatur **87**

Testfälle **90**

Abbildungsverzeichnis

1 Business Process Management System nach [Dumas et al., 2013, Weske, 2007] 4
2 Aufbau der XML-Repräsentation von MBAs [Schuetz, 2015b] 5
3 Beispiel für das Problem aufgrund der Pending Update List 14
4 Aufbau der Process Engine 16
5 RESTXQ Steuerungsfunktion 19
6 RESTXQ Grundsätzlicher Aufbau 20
7 Aufbau der Funktion sc:addDescendantStatesToEnter 45
8 Speicherung des MBA in der Datenbank 57
9 Verknüpfung zwischen Process Environment und Process Engine . . 69
10 Benutzungsschnittstelle: enterMBA 77
11 Ablauf der Benutzungsschnittstelle 78
12 Benutzungsschnittstelle: getEvents 79
13 Benutzungsschnittstelle: getDataElements 81
14 Benutzungsschnittstelle: addEvent 82
15 Benutzungsschnittstelle: queryMBA 83
16 Benutzungsschnittstelle: getResult 84

Tabellenverzeichnis

1 RESTXQ-Schnittstelle: Steuerungsfunktion 21
2 RESTXQ-Schnittstelle: rekursive Funktionen 23
3 RESTXQ-Schnittstelle: sonstige Funktionen 26
4 RESTXQ-Schnittstelle: Initiierungsfunktion 27
5 Modul SXCML: Funktionen zum Erhalten von internen und externen Events 27
6 Modul SCXML: Funktionen für ausführbaren Inhalt 28
7 Modul SCXML: Funktionen welche das MBA Modul aufrufen . . . 31
8 Modul SCXML: Funktionen zur Verarbeitung von Zustandswechseln 33
9 Modul SCXML: Funktionen zum Ausführen von `invoke`-Elementen 34
10 Modul SCXML: Hilfsfunktionen 35
11 Modul SCXML: Evaluierungsfunktionen 36
12 Modul SCXML: Funktionen zur Berechnung von Transitionen . . . 38
13 Modul SXCML: Funktionen zum Berechnen des Entry-/ExitSets . 39
14 Modul SCXML: Funktionen zum Ausführen von ausführbarem Inhalt 41
15 Modul SCXML: Zustandsfunktionen 43
16 Modul SCXML: Funktionen zur Berechnung von Nachfahren und Vorfahren 44
17 Modul SCXML: Funktionen zum Erhalt von Informationen über Transitionen 48
18 Modul SCXML: Funktionen zum Erhalt von Informationen über Zustände 49
19 Modul MBA: Funktionen für externe Events 51
20 Modul MBA: Funktionen für interne Events 52
21 Modul MBA: Funktionen für das aktuelle Event 52
22 Modul MBA: Funktionen für systemspezifische Variablen 1 53
23 Modul MBA: Funktionen zur Verknüpfung von aufgerufenen Interpretern 55
24 Modul MBA: Funktionen zur Speicherung von `invoke`-Elementen . 56
25 Modul MBA: Funktionen zur Markierung von MBAs 58
26 Modul MBA: Funktion zum Initialisieren des MBAs 59
27 Modul MBA: Funktionen für systemspezifische Variablen 2 62
28 Modul MBA: Funktionen zum Erhalten von Datenmodellen 63
29 Modul MBA: Hilfsfunktionen 64
30 Funktion zum Loggen 66
31 WorklistHandler : RESTXQ-REST-Funktionen 74
32 WorklistHandler : insertEvent-Funktion 74
33 WorklistHandler : RESTXQ-Funktionen 76

34 WorklistHandler: Benutzungsschnittstelle 81

Quellcodeverzeichnis

1 Beispiel für MBA 7
2 Beispiel für SCXML-Modell, welches Test355.txml im Implementation Report [Barnett, 2015] entspricht 11
3 Modul SCXML: Verwendung von Copy-Modify 15
4 Modul RESTXQ: Weiterleiten innerhalb der RESTXQ-Schnittstelle 17
5 Modul RESTXQ: Funktion in der RESTXQ-Schnittstelle am Beispiel von invoke 18
6 Modul RESTXQ: Rekursiver Funktionsaufruf 24
7 Modul SCXML: Regular Expression zum Überprüfen von Events . . 36
8 Modul SCXML: Funktion sc:getMBAFromText 37
9 Modul SCXML: Berechnung der zu verlassenden Zustände 40
10 Modul SCXML: Zusammenstellen des Funktionsheaders 42
11 Modul SCXML: Funktionsaufruf in sc:addDescendantStatesToEnter 46
12 Modul SCXML: Rekursiver Aufruf der Funktion sc:foldAncestorStatesToEnter 50
13 Modul MBA: Funktionsweise der Funktion mba:addCurrentState . . 54
14 Modul MBA: Speicherung der Collection in der Datenbank 59
15 Modul SCXML-Extension: Erweiterung inState 65
16 Modul SCXML-Extension: Verwendung von inState 65
17 Modul SCXML: Logging 67
18 Benutzungsschnittstelle: Rückgabewert der Funktion getEvents . . . 73
19 Benutzungsschnittstelle: Hinzufügen eines Events 75
20 Benutzungsschnittstelle: Berechnung der Eventdaten 75
21 Benutzungsschnittstelle: Rückgabewert der Funktion getDataElements 76
22 Benutzungsschnittstelle: XSL Stylesheet für die Funktion getEvents 80

1 Einführung

Geschäftsprozessmodelle (kurz: Prozessmodelle) werden dazu verwendet gewisse Merkmale von Prozessen in Organisationen zu beschreiben. Ein Prozess ist eine zeitliche Abfolge von Funktionen und Tätigkeiten, welche bestimmte betriebliche Objekte bearbeiten und von Personen mit bestimmten Rollen durchgeführt werden müssen (siehe [Becker, 2012]). Es gibt eine Vielzahl von Gründen, um einen Prozess zu modellieren. Neben einem besseren Verständnis der Abläufe in einem Unternehmen erleichtern es Prozessmodelle, Vorgänge für am Prozess beteiligte Personen verständlich darzustellen [Dumas et al., 2013]. Des Weiteren ermöglichen Prozessmodelle die systematische Analyse von gesamten Prozessen bzw. Teilabschnitten mithilfe qualitativer bzw. quantitativer Analysemethoden. Durch sie können Maßzahlen bestimmt werden, mit denen beispielsweise Aussagen über die Leistungsfähigkeit eines Prozesses getroffen werden können. Diese Metriken vereinfachen die Prozesskontrolle, darüber hinaus stellen sie eine wichtige Grundlage für die Verbesserung von Prozessen dar [Gadatsch, 2008].

Prozessmodelle sind Voraussetzung für die Automatisierung von Geschäftsprozessen mittels Geschäftsprozessverwaltungssystemen (Business Process Management Systems, BPMS). Ein BPMS stellt bei der Ausführung von Prozessen sicher, dass die einzelnen Aktivitäten eines Prozesses in der richtigen Reihenfolge durchgeführt und stets die benötigten Ressourcen bereitgestellt werden. Zu den Hauptkompenenten eines BPMS zählen unter anderen Process Environment, Process Engine und Worklist Handler [Dumas et al., 2013, Weske, 2007], welche für die Verarbeitung von Ereignissen und Durchführung von Aktionen sorgen.

Geschäftsprozesse laufen auf verschiedenen Hierarchieebenen innerhalb eines Unternehmens ab. Die Geschäftsprozesse auf den verschiedenen Ebenen laufen dabei nicht getrennt voneinander, sondern sind über verschiedene Beziehungen miteinander verknüpft. Das Konzept des Multilevel Business Artifact (MBA [Schuetz, 2015b]) ermöglicht die Abbildung von mehrstufigen Prozessmodellen. Eine maschinenlesbare Notation in XML unter Verwendung von State Chart XML dient als Grundlage für die Prozessautomatisierung.

Für die Automatisierung mehrstufiger Geschäftsprozesse, die mittels State Chart XML (SCXML) modelliert werden, präsentiert die vorliegende Arbeit einen SCXML-Interpreter als Grundlage für die Process Engine. SCXML ermöglicht die Beschreibung komplexer Zustandsautomaten und bietet dabei neben parallelen Zuständen die Möglichkeit, Zustände in Subzustände zu unterteilen. SCXML ist ereignisbasiert und erlaubt zusätzlich zur Modellierung von Prozessmodellen die Verwendung von Datenelementen [Moura und Guedes, 2007]. Diese W3C-Empfehlung wird in Zusammenhang mit der Darstellung von MBAs verwendet. Als Grundlage wird die von Schütz et al. [Schuetz et al., 2015] vorgeschlagene, an den SCXML-Standard angelehnte, logische Darstellung von MBAs verwendet.

In der vorliegenden Arbeit wird eine prototypische Implementierung von Komponenten eines BPMS für die Automatisierung mehrstufiger Geschäftsprozesse präsentiert. Für die Implementierung der Process Engine wurde die BaseX-Datenbank [BaseX Team, 2016] in Verbindung mit den beiden Technologien XQuery und RESTXQ verwendet. Dabei wurde auf die Umsetzung von Schütz [Schuetz, 2015b] aufgebaut. Die Technologieentscheidung zu Gunsten von XQuery wurde wegen der XML-basierten Modellierung von MBAs getroffen. Um sequenzielle, aufeinander aufbauende XQuery-Änderungen durchführen zu können, wurde eine Umsetzungsvariante mit RESTXQ gewählt. Außerdem erfolgen die Interaktionen zwischen Benutzern und Benutzerinnen mit dem BPMS über RESTXQ.

Die vorliegende Arbeit ist wie folgt gegliedert. Eine Beschreibung der relevanten Konzepte und verwendeten Technologien findet sich in Abschnitt 2. Eine Beschreibung der Umsetzung von Process Engine und Process Environment für die Automatisierung mehrstufiger Geschäftsprozesse, inklusive relevanter Entwicklungsentscheidungen, findet sich in Abschnitt 3. Eine Beschreibung der Benutzungsschnittstelle (Worklist Handler) findet sich in Abschnitt 4.

2 Hintergrund

In diesem Kapitel werden kurz die Grundlagen der Geschäftsprozessautomatisierung erläutert. Ebenfalls vorgestellt werden die verwendeten Technologien sowie ähnliche Arbeiten, welche sich ebenfalls mit der Automatisierung von Geschäftsprozessmodellen beschäftigen, insbesondere SCXML-Interpreter.

2.1 Geschäftsprozessautomatisierung

Für die Automatisierung von Geschäftsprozessen werden *Geschäftsprozessverwaltungssysteme* (Business Process Management Systems, BPMS) eingesetzt. BPMS verwenden Prozessbeschreibungen – die Prozessmodelle – um Prozesse zu koordinieren [Dumas et al., 2013, S. 298]. Zu den Aufgaben eines BPMS zählt die Sicherung der Einhaltung der Reihenfolge der einzelnen Prozessschritte sowie deren Ressourcenzuordnung. BPMS werden dazu verwendet, die Erreichung von Geschäftszielen zu erleichtern. [Weske, 2007]

Ein BPMS besteht aus mehreren Teilen (siehe Abbildung 1), die für jeweils einen bestimmten Aspekt des BPMS verantwortlich sind, nämlich Modellierungswerkzeug, Process Repository, Process Engine, Process Environment, Worklist Handler, und Werkzeuge für die Administration bzw. das Überwachen der Geschäftsprozesse. Die vorliegende Arbeit implementiert Process Engine, Process Environment und Worklist Handler eines BPMS für mehrstufige Geschäftsprozesse.

Das Modellierungswerkzeug ermöglicht es Benutzerinnen und Benutzern, Geschäftsprozessmodelle zu erstellen und zu verändern, wodurch es möglich ist, ein Geschäftsprozessmodell mit Zuständen, Operationen und Verbindungen zwischen den Zuständen zu erzeugen [Weske, 2007, S. 128]. Das Modellierungswerkzeug unterstützt die Speicherung der erzeugten Geschäftsprozessmodelle im Process Repository. Im Repository kann das Modell sowohl von dem Modellierungstool als auch von der Process Engine aufgerufen werden. Die Process Environment ist für den Aufruf der Prozess Engine zuständig, was aufgrund des im Repository enthaltenen Prozessmodells erfolgt. Die Process Engine ist für die Interpretation des Geschäftsprozesses verantwortlich, weshalb die Process Engine auch das Kernelement eines BPMS ist. Im Worklist Handler können Elemente (Daten, Ereignisse) von Prozessbeteiligten hinzugefügt werden. Diese Elemente werden anschließend von der Prozess Environment aufgegriffen und haben Einfluss auf die Interpretation des Geschäftsprozessmodells. Für weiterführende Informationen zu BPMS sei an dieser Stelle auf die Literatur verwiesen [Dumas et al., 2013, Weske, 2007, Schuetz, 2015b].

In dieser Arbeit werden bestimmte Komponenten eines BPMS implementiert. Für die Speicherung von Prozessmodellen dient eine XML-Datenbank in BaseX als Repository, welches die Verwaltung der XML-Dokumente übernimmt. Um Über-

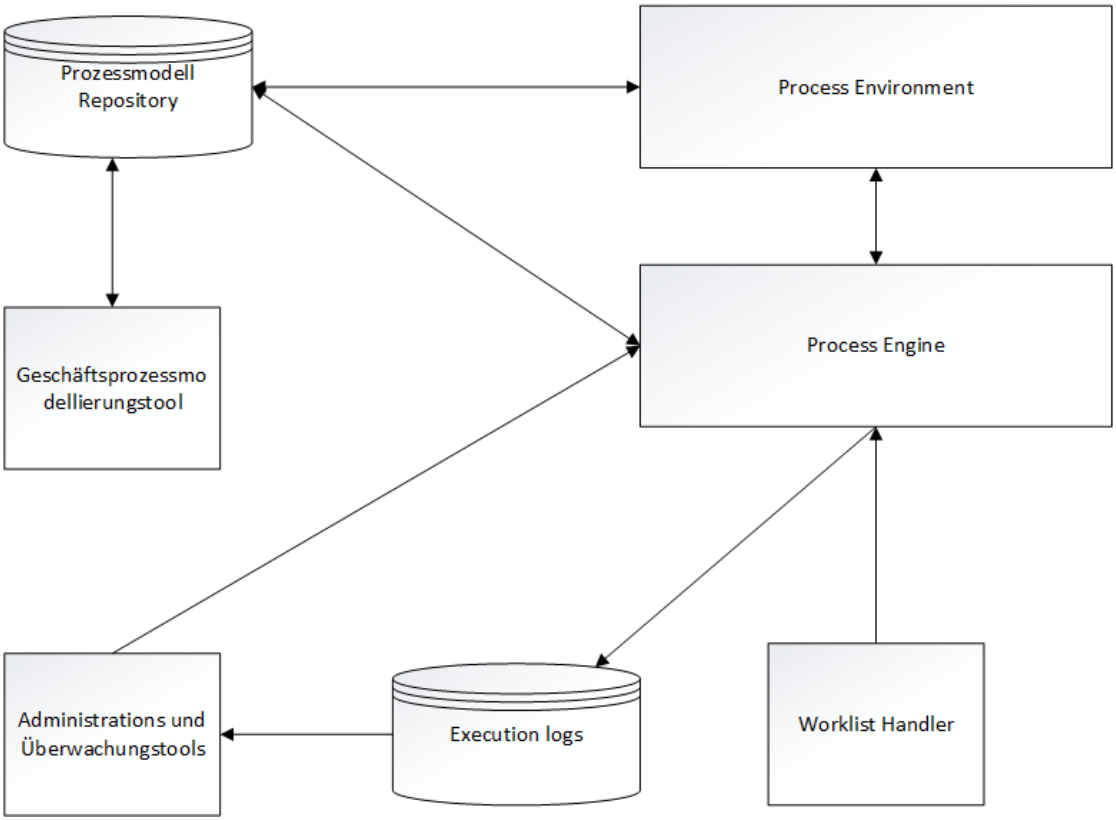

Abbildung 1: Business Process Management System nach [Dumas et al., 2013, Weske, 2007]

wachung und Analyse von mehrstufigen Prozessen in Zukunft zu ermöglichen, wird in dieser Arbeit eine Logging-Funktion implementiert. Das Modellierungswerkzeug wird in dieser Arbeit nicht umgesetzt.

2.2 Mehrstufige Geschäftsprozesse

Unter mehrstufigen Geschäftsprozessen versteht man die Abbildung von Geschäftsprozessen auf den verschiedenen hierarchischen Ebenen eines Unternehmens [Schuetz, 2015a]. Üblicherweise wird bei Unternehmen zwischen drei generischen Ebenen unterschieden, nämlich strategische, taktische und operative Ebene, wobei die genaue Ausgestaltung und Anzahl der Hierarchieebenen je nach Unternehmen variiert. Die operative Ebene beschäftigt sich bei dieser Aufteilung mit der Erreichung der in der strategischen und taktischen Ebenen festgelegten Ziele und damit mit den täglichen Aufgaben eines Unternehmens. Im Gegensatz dazu werden in der höchsten Ebene die wichtigsten Ziele einer Organisation festgelegt. In der

strategischen Ebene werden also die langfristigen Entscheidungen eines Unternehmens getroffen. In der mittleren, der taktischen Ebene werden dementsprechend mittelfristige Ziele, welche das Erreichen der strategischen Ziele als Vision besitzen festgelegt. Für weiterführende Informationen zur hierarchischen Gliederung von Unternehmen sei an dieser Stelle auf die Literatur verwiesen [Steinmann und Schreyögg, 2000, Steiner, 1971].

Das Konzept des Multilevel Business Artifact (MBA) bietet die Möglichkeit, mehrstufige Geschäftsprozesse abzubilden. Die Modellierung orientiert sich dabei an den Datenobjekten auf unterschiedlichen Abstraktionsebenen, die den Hierarchieebenen des Unternehmens folgen. Ein MBA verbindet einzelne Geschäftsprozesse auf den unterschiedlichen Ebenen mitsamt ihren jeweiligen Datenelementen und kapselt diese in einem Objekt. Die Ebenen eines MBA orientieren sich dabei an den Datenobjekten in einem Unternehmen. Die Zusammensetzung der Datenobjekte spiegelt grob die hierarchische Organisation des Unternehmens wieder.

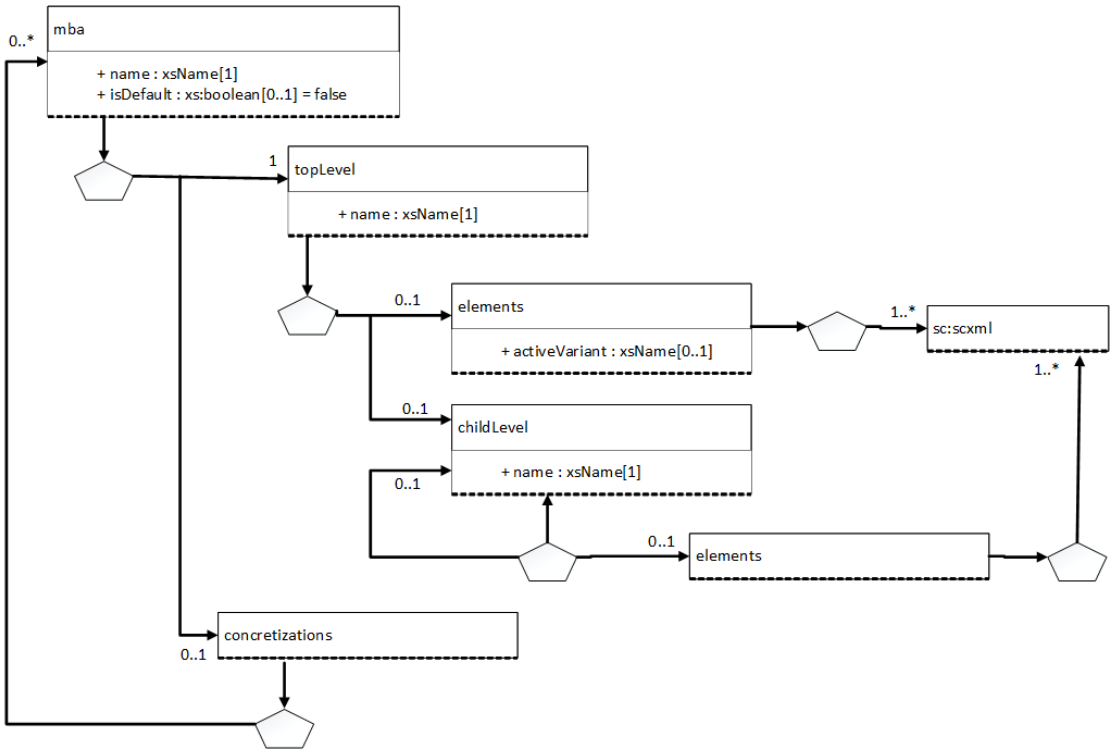

Abbildung 2: Aufbau der XML-Repräsentation von MBAs [Schuetz, 2015b]

Innerhalb eines MBA sind die einzelnen Prozesse hierarchisch geordnet und ein MBA beinhaltet neben dem Prozess- und Datenmodell für jede Hierarchieebene auch eine Instanziierung von Prozess- und Datenmodell auf der obersten

Ebene (Top-Level) [Schuetz, 2015a, Schuetz et al., 2015]. Jede Hierarchieebene beinhaltet also ein Zustandsautomatendiagramm (State Machine Diagram) sowie ein Datenmodell. Durch die Angabe des Zustandsautomatendiagramms sind auch die Methoden definiert, welche Änderungen am Zustand eines Objekts verursachen können. Dadurch werden auch die Kommunikationsmöglichkeiten zwischen den einzelnen Hierarchiestufen bestimmt.

Zur Speicherung von MBAs wird eine XML-basierte Notation verwendet. Dies ermöglicht auch die Nutzung des SCXML-Standards zur Beschreibung der darin beinhalteten Prozessmodelle. Weil in der vorliegenden Arbeit nur mit einfachen Hierarchien (Simple Hierarchies) gearbeitet wird, wird an diesere Stelle auf die Beschreibung von parallelen Hierarchien verzichtet (Parallel Hierachies). Der Aufbau einer XML-Abbildung eines MBA ist in Unterabschnitt 2.2 dargestellt. Die rechteckigen Elemente stehen in der Grafik dabei für XML-Elemente, ein Pfeil, der auf ein Pentagon zeigt, zeigt ein Kindelement des aktuellen Elements an. Das Top-Level-Element ist dabei immer ein MBA-Element welches auch gleichzeitig den höchsten Level des MBA beschreibt. Dieser höchste Level ist jedoch nur eine abstrakte Konkretisierung eines MBA und dient daher auch als Wurzelelement.

Ein MBA-Element besitzt immer ein `name`-Attribut und kann ein `isDefault`-Attribut enthalten. Unterhalb dieses Elements befinden sich die beiden Elemente `topLevel` und `concretizations`. Das im `topLevel`-Element enthaltene Element `elements` enthaltet alle Datenelemente und Zustände, die für die Beschreibung des Lebenszyklus des Top-Level notwendig sind. Dafür wird der SCXML-Standard verwendet, weshalb unterhalb des `elements`-Element immer ein `SCXML`-Element zu finden ist.

Das `childLevel`-Element beschreibt die Ebenen eines MBA unterhalb des Top-Levels. Die Schachtelung der `childLevel`-Elemente legt die hierarchische Ordnung der Ebenen eines MBA fest. Wie das `topLevel`-Element werden die unteren Ebenen des MBA ebenfalls über ein `element`-Element beschreiben. Unter dem `concretizations`-Element befinden sich die Konkretisierungen eines MBA. Die Konkretisierungen eines MBA sind wieder vollständige MBAs, mit allen Elementen eines MBA. Eine Konkretisierung prägt dabei immer den zweiten Level des konkretisierten MBA aus, d.h. das erste `childLevel`-Element unterhalb des Top-Levels des konkretisierten MBA. Ein Beispiel für Konkretisierung ist in Quellcode 1 dargestellt. Das hier beschriebene MBA `JohannesKeplerUniversity` beginnt wie notwendig mit einem `mba`-Element. Unterhalb dieses Elements liegt das `topLevel`-Element, das das MBA auf der obersten Hierarchieebene genauer beschreibt. Das SCXML-Modell, welches unterhalb des `element`-Elements angegeben ist, beschreibt das Zustandsautomatendiagramm des aktuellen MBA auf der `university`-Ebene. Das Element `childLevel` beschreibt nun den zweiten Level des MBA, in diesem Fall `school`, das ebenfall eine weitere Kindebene besitzt, nämlich `program`.

```
<?xml version="1.0" encoding="UTF-8"?>
<mba xmlns="http://www.dke.jku.at/MBA"
     name="JohannesKeplerUniversity"
     hierarchy="simple">
   <topLevel name="university">
      <elements>
        <scxml>
           ...
        </scxml>
      </elements>
      <childLevel name="school">
         <elements>
            <scxml>
               ...
            </scxml>
         </elements>
         <childLevel name="program">
         </childLevel>
      </childLevel>
   </topLevel>
   <concretizations>
      <mba name="SocialAndEconomicSciences" hierarchy="simple">
         <topLevel name="school">
            <elements>
               <scxml>
                  ...
               </scxml>
            </elements>
            <childLevel name="program">
               <elements>
                  <scxml>
                    ...
                  </scxml>
               </elements>
              </childLevel>
         </topLevel>
       <concretizations>
           <mba>
         ...
           </mba>
         </concretizations>
      </mba>
   </concretizations>
</mba>
```

Quellcode 1: Beispiel für MBA

Die Konkretisierung des MBA `JohannesKeplerUniversity` erfolgt schließlich im Element `concretizations`. Das MBA `SocialAndEconomicSciences` verfügt über die bereits angegebene Hierarchie mit den Ebenen `school` und `program`. Das MBA `SocialAndEconomicSciences` besitzt als zweite Hierarchieebene die Ebene `program`. Diese Hierarchiebene wird möglicherweise von MBAs in dem darauffolgenden `concretizations`-Element ausgeprägt und genauer beschrieben.

2.3 State Chart XML

State Chart XML (SCXML), seit September 2015 eine W3C-Empfehlung, ist eine XML-basierte Sprache zur Beschreibung von Zustandsautomaten/Zustandsdiagrammen [Barnett et al., 2015]. SCXML baut dabei auf CCXML (*Call Control eXtensible Markup Language*, [Auburn et al., 2011] und Harels Statecharts [Harel, 1987] auf.

2.3.1 Sprachelemente

SCXML besteht aus unterschiedlichen XML-Elementen, die dazu verwendet werden können, einen objektorientierten Geschäftsprozess zu beschreiben. Die wichtigsten dieser Sprachelemente werden nachfolgend kurz erklärt. Die Erklärung folgt dabei dem Standard [Barnett et al., 2015].

SCXML. Das SCXML-Element ist das übergeordnete Element, welches das gesamte Zustandsdiagramm beinhaltet. Alle Elemente unterhalb dieses Elements beschreiben das Zustandsdiagramm. Neben den einzelnen Zuständen ist hier auch das Datenmodell des SCXML-Modells angegeben. Datenelemente können im SCXML-Modell global oder auch lokal in den einzelnen Zuständen angegeben sein.

state. Dieses Element beinhaltet die Beschreibung eines einzelnen Zustands. Ein Zustand besteht dabei aus den Transitionen, den Subzuständen, den Datenelementen und eventuell aus den ausführbaren Inhalten sowie `invoke`- Elementen. Jeder Zustand hat eine eindeutige Id, welche auch zum identifizieren des Zustandes verwendet wird. Ausführbare Inhalte können innerhalb eines Zustandes an zwei Orten vorkommen, dem `onEntry`-Block sowie dem `onExit`-Block. Wie bereits aus diesen Bezeichnungen abgeleitet werden kann, wird der `onEntry`-Block beim Betreten des Zustandes ausgeführt, während der `onExit`-Block beim Verlassen des Zustandes ausgeführt wird. Für Verbindungen zwischen den einzelnen Zuständen gibt es Transitionen, die innerhalb des Zustandes definiert sind. Der Zustand, in dem eine Transition definiert ist, markiert ihren Startzustand. Zur Angabe des Initialzustands eines zusammengesetzten Zustands gibt es drei Möglichkeiten. Die erste

Möglichkeit ist die Verwendung des `initial`-Attributes, in dem die Id des initialen Zustands angegeben wird. Die zweite Möglichkeit ist die Verwendung eines `initial`-Elementes. Die dritte Möglichkeit ist das Fehlen des `initial`-Elements oder des `initial`-Attributs. In diesem Fall wird der erste Zustand im Dokument als initialer Zustand verwendet (*document order*).

parallel. Das `parallel`-Element wird verwendet, um parallele Zustände zu modellieren. Alle Subzustände dieses Elements sind gleichzeitig aktiv.

transition. Transitionen sind die Verbindungen zwischen den einzelnen Zuständen. Sie sind innerhalb von Zuständen angegeben, und haben bei Bedarf ein `target`-Attribut , welches auf einen anderen Zustand verweist. Sollte kein `target`-Attribut angegeben sein, verweist die Transition auf den Startzustand. Transitionen können über Bedingungen eingeschränkt sein, welche in einem `cond`-Attribut angegeben werden. Außerdem können sie über Events getriggert werden. Die Angabe dieser Trigger erfolgt im `event`-Attribut. Innerhalb dieses Attributs können mehrere Events eine Transition triggern, diese sind dabei mit Leerzeichen getrennt. Innerhalb von Transitionen können ebenfalls ausführbare Inhalte angegeben sein, diese werden beim Ausführen der Transition ausgeführt.

intial. Das `initial`-Element ist eine Möglichkeit, innerhalb von komplexen Zuständen den initialen Zustand anzugeben. Dabei kann innerhalb dieses Elements sowohl ein Zustand als auch eine Transition angegeben sein, welche zum initialen Zustand führt.

history. Historyzustände erlauben es dem SCXML-Zustand die aktuelle Konfiguration zu speichern. Der SCXML-Standard kennt und unterstützt dabei die beiden Historyarten `shallow` und `deep`.

invoke. Das `invoke`-Element wird dazu verwendet, um ein neues SCXML-Modell als externen Service aufzurufen. Das SCXML kann dabei direkt unterhalb des `invoke`-Elements angegeben sein, aber auch in einer externen Datei.

Neben den Elementen, welche das Zustandsdiagramm beschreiben, gibt es die darin inkludierten ausführbaren Inhalte. Ausführbare Inhalte erlauben das Verändern von Daten bzw. das Interagieren eines SCXML-Modelles mit externen Funktionen. Wie bereits erwähnt kann ausführbarer Inhalt sowohl in den `onEntry`- und `onExit`-Blöcken von Zuständen vorkommen als auch in Transitionen. Der Standard unterstützt dabei nachfolgende ausführbare Inhalte.

raise. Fügt ein neues Event zur Liste der internen Events hinzu.

if. Dieses Element erlaubt über das `cond`-Attribut eine Einschränkung des ausführbaren Inhaltes durchzuführen. Ausführbare Inhalte innerhalb dieses Elements werden nur durchgeführt, wenn diese Einschränkung sich positiv evaluieren lässt. Dieses Event ermöglicht auch das Darstellen von Verzweigungen mit der Verwendung von `elseif` und `else`-Elementen. Wenn die Bedingungen im `if`-Element nicht positiv evaluiert werden, wird an diese Elemente weitergeleitet.

foreach. Dieses Element erlaubt es über eine Liste von Elementen im Datenmodell zu iterieren und die ausführbaren Inhalte für jedes der darin beinhalteten Datenelemente auszuführen. Allerdings wird dieses Element nicht vom vorliegenden Interpreter unterstützt.

log. Dieses Item wird dazu verwendet, um etwas im Log zu speichern. Es kann unter anderem für Debugmeldungen verwendet werden

assign. Dieses Element wird für das Verändern des Datenmodells verwendet. Der zu verändernde Wert wird im Attribut `location` angegeben, der Wert, mit dem dieser Wert verändert wird, wird in dem Attribut `expression` beschrieben. Anstatt dem `expression`-Attribut kann auch ein Subelement diesen Wert beschreiben.

send. Das `send`-Element wird dazu verwendet, um Events an externe Systeme, wie z.B. andere SCXML-Interpreter zu versenden. Mit dem `send`-Event erzeugte Events werden dem SCXML-Modell immer als externe Events hinzugefügt.

Ein kurzes Beispiel eines SCXML-Modells ist in Quellcode 2 dargestellt. Dieses Modell beinhaltet die beiden Zustände `s0` und `s1` sowie die beiden finalen Zustände `pass` und `fail`. Die Transition im Zustand `s0` führt zum Zustand `pass`, in welchem anschließend der ausführbare Inhalt `log` ausgeführt wird.

2.3.2 SCXML-Interpreter

Eine der vollständigsten Umsetzungen des SCXML-Standards ist Commons SCXML, dass von der Apache Software Foundation entwickelt wird [Apache Commons, 2015]. Der Algorithmus ist in Java umgesetzt und befindet sich noch in Entwicklung. Der Interpreter ist Open Source und ermöglicht auch Außenstehenden zu diesem Projekt beizutragen. Das XPath-Datenmodell wird zwar unterstützt, jedoch gibt es noch einige Probleme mit diesem [Apache Commons, 2016]. Der Commons-SCXML-Interpreter orientiert sich bei der Implementierung an den

```
1 <scxml xmlns="http://www.w3.org/2005/07/scxml"
2 xmlns:conf="http://www.w3.org/2005/scxml-conformance"
3 datamodel="xpath" version="1.0">
4    <state id="s0">
5       <transition target="pass" />
6    </state>
7    <state id="s1">
8       <transition target="fail" />
9    </state>
10   <final id="pass">
11      <onentry>
12         <log expr="'pass'" label="Outcome" />
13      </onentry>
14   </final>
15   <final id="fail">
16      <onentry>
17         <log expr="'fail'" label="Outcome" />
18      </onentry>
19   </final>
20 </scxml>
```

Quellcode 2: Beispiel für SCXML-Modell, welches Test355.txml im Implementation Report [Barnett, 2015] entspricht

Testfällen des Implementation Reports von SCXML [Barnett, 2015]. Der enthaltene Parser analysiert nicht nur die Elemente des SCXML-Dokuments, sondern validiert auch die Syntax bzw. die die Struktur des Elementes. Dadurch können nur valide Dokumente verwendet werden [Mahmood et al., 2009].

Eine weitere Umsetzung des Standards ist JSSCxml [Touffy, 2015], welche mit Javascript implementiert wurde. Dieser Algorithmus ist ähnlich aufgebaut wie der im SCXML-Standard angegebene Algorithmus. Der JSSCxml-Interpreter unterstützt jedoch nur das ECMAScript-Datenmodel. Das XPath-Datenmodell wird nicht unterstützt. Dieser Interpreter unterstützt ebenfalls benutzerspezifischen ausführbaren Inhalt. Die JSSCxml-Umsetzung hat noch Probleme mit dem ausführbaren Inhalt `assign`, bei dem es bei nicht näher definierten Beispielen zu Problemen kommen könnte.

Ein weiteres Problem von JSSCxml ist die nur teilweise Unterstützung von I/O-Prozessoren. Dies funktioniert nicht mit dem standardisierten Element `send`, sondern mit Hilfe eines benutzerspezifischen ausführbaren Inhaltes. Auch einige Arten von Fehler die bei den beiden Elementen `param` und `donedate` auftreten können, sind noch nicht implementiert. Neben dem nicht unterstützen Datenmodel, gibt es noch zusätzliche Probleme bei den beiden ausführbaren Inhalten `send`

und `assign`. Die letzte Änderung der Implementierung wurde im August 2015 durchgeführt.

Gavin Kistner hat mit dem SCXML-Interpreter names LXSC (*Lua XML StateChart Interpreter*) eine Umsetzung in Lua [Ierusalimschy et al., 1996] erstellt [Kistner, 2014]. Der Vorteil dieser Implementierung ist, dass auch das Datenmodell mit Lua evaluiert wird, die als Scriptsprache einige weiter Elemente für die Evaluierung bietet. Dies hat daher auch den Nachteil, dass andere Datenmodelle wie ECMAScript oder XPath nicht unterstützt werden. Auch der LXSC-Interpreter unterstützt benutzerspezifische ausführbare Inhalte, welche jedoch zuerst implementiert werden müssen. LXSC basiert auf dem Working Draft der SCXML-Spezifikation von Februar 2013 und unterstützt die wichtigsten Elemente dieser Spezifikation. Es gibt jedoch Probleme bezüglich dem Verzögern von Events (Element `delay`) und das Aufrufen von externen Services mittels `invoke`-Element wird nicht unterstützt. Aufgrund dessen ist auch keine andere Interpreter-Interpreter-Kommunikation implementiert und auch das Element `send` wird nicht unterstützt [Kistner, 2014]. Die letzte Aktualisierung dieser Implementierung erfolgte jedoch bereits vor Änderung des SCXML-Modells zur W3C-Empfehlung im Juli 2015.

Ein weiterer SCXML-Interpreter ist die von Barbier entwickelte Pauware engine. Dies ist eine Java-Implementierung des SCXML-Standards, welche an der Unversität Pau entwickelt wurde [Barbier, 2016b, Romeo et al., 2016]. Neben dem Interpreter wurde auch eine Erweiterung der Pauware Engine implementiert, welche die grafische Simulation von SCXML-Dokumenten erlaubt [Barbier, 2016a]. Das XPath-Datenmodell wird von dieser Implementierung nicht unterstützt, es sind jedoch alle sonstigen Elemente wie auch Timer und Historyzustände korrekt implementiert. Zur Unterstüzung für Anwender ist auch eine Liste bereits umgesetzer SCXML-Dokumente zum besseren Verständnis vorhanden [Barbier et al., 2015].

2.4 XQuery und RESTXQ

Für diese Arbeit wurden die XML-Technologien XQuery und RESTXQ verwendet. Die Verwendung von XQuery und RESTXQ war aufgrund der Speicherung der mehrstufigen Geschäftsprozessmodelle in einer XML-Datenbank vorteilhaft gegenüber einer Implementierung in einer nicht XML-basierten Technologie.

Die XML Query Language (XQuery) ist eine W3C-Empfehlung zum Selektieren und Bearbeiten von Daten aus Dokumenten und Datenbanken, die in XML-Notation gespeichert sind [Robie et al., 2014]. XQuery ist Turing-vollständig und ist der Nachfolger von Quilt [Zhang et al., 2012, BaseX Team, 2016]. Seit Dezember 2015 ist Version 3.1 ein W3C Candidate Recommendation, allerdings wurde sie aufgrund der bereits existierenden Umsetzung in BaseX und den inkludierten

Funktionen zur Verarbeitung von Maps bereits verwendet. Die XQuery-Version 3.1 baut auf die XPath Version 3.1 [Dyck und Robie, 2014] und erweitert diese, daher ist jeder gültige XPath-Ausdruck auch ein gültiger XQuery-Ausdruck. XQuery wird von mehreren unterschiedlichen Datenbanken unterstützt, z.B. eXistdb und BaseX. Für weiterführende Informationen wird auf die XQuery-Spezifikation verwiesen [Robie et al., 2014].

RESTXQ ist eine von Adam Retter entwickelte Schnittstelle die XQuery-Funktionen um Anmerkungen für REST-Services erweitert. Diese Erweiterung erlaubt es mit XQuery-Webapplikationen zu erstellen. Die hinzugefügten Anmerkungen ermöglichen es einer XQuery-Funktion einen URI zuzuweisen. Dadurch kann diese Funktion auch von einem REST-Service aufgerufen werden. Zusätzlich kann die HTTP-Methode, z.B. `GET` oder `POST`, der Methode spezifiziert werden und die Verwendung dadurch eingeschränkt werden. Über weitere Anmerkungen können die Funktionen darüber hinaus noch weiter eingeschränkt werden, dazu zählt auch der erlaubte Rückgabewert. Sollte keine der oben genannten Einschränkungen angegeben sein, wird die Funktion für alle HTTP-Methoden und auch für alle Rückgabewerte ausgeführt.

Auch REST-Parameter werden über die in der Schnittstelle definierten Anmerkungen definiert, dabei ist es auch möglich Standardwerte zu setzen [Retter, 2016, BaseX Team, 2016]. Das Ergebnis eines Aufrufes der RESTXQ-Schnittstelle kann einer von drei Rückgabewerten sein. Die erste Möglichkeit ist eine Ressource, d.h. einfacher Inhalt, der zweite Typ ein HTTP-Header und schlussendlich eine Kombination der beiden anderen Rückgabewerte. Die in BaseX verwendete RESTXQ-Schnittstelle hat noch einige zusätzliche Erweiterungen im Gegensatz zu der von Adam Retter entwickelten Schnittstelle.

3 Process Engine und Process Environment

Im nachfolgenden Kapitel wird die Implementierung des SCXML-Interpreters beschrieben. Dabei wird zwischen den beiden Elementen Process Engine und Process Environment unterschieden. Die beiden Kapitel 3.2 und 3.3 behandeln dabei die Process Engine, das Kapitel 3.5 behandelt die Process Environment. Im Kapitel 3.4 wird das von der Process Engine erstellt Logging beschrieben.

3.1 Systemüberblick

Bei der Untersuchung der bereits existierenden SCXML-Interpreter (vgl. Unterunterabschnitt 2.3.2) wurde festgestellt, dass das XPath-Datenmodell nicht unterstützt wird. Ein Grund dafür war, dass das XPath-Datenmodell aufgrund nicht existierender Implementierungen teilweise aus der Beschreibung des Standards herausgefallen ist [Barnett et al., 2014, Barnett et al., 2015]. Daher war die Unterstützung dieses Datenmodells eines der Hauptziele der Implementierung.

Ein weiterer wichtiger Faktor war die Unterstützung von MBAs. Dies ermöglicht die Kommunikation zwischen den Hierarchiestufen eines MBAs sowie zwischen den unterschiedlichen MBAs. Aufgrund der Tatsache das MBAs in XML gespeichert werden, wurde bereits sehr früh die Entscheidung getroffen soweit als möglich die Umsetzung mit XQuery durchzuführen.

Zuerst wurde versucht eine rekursive Lösung des Interpreters zu erstellen. Das bedeutete, dass sich der Algorithmus erneut aufruft, solange der Interpreter noch abzuarbeitende Punkte besitzt. Hier wurde jedoch sehr bald ein weiteres Problem festgestellt.

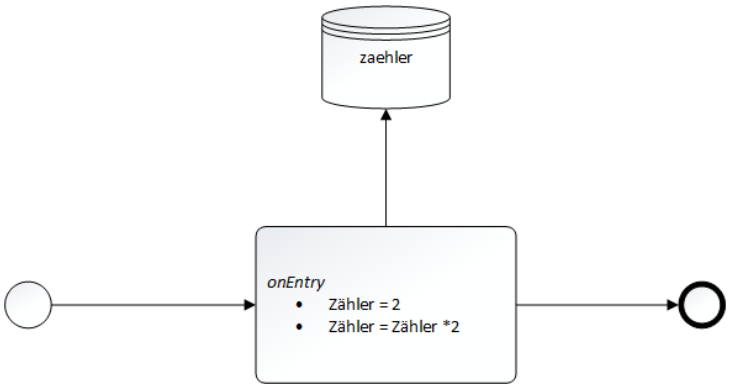

Abbildung 3: Beispiel für das Problem aufgrund der Pending Update List

Eines der Grundprinzipien von XQuery ist, dass die Update Funktionen, d.h. Funktion welche Änderungen in der Datenbank verursachen können, nicht sofort

ausgeführt werden, sondern zuerst in einer Pending Update List gesammelt werden. Erst am Ende einer Abfrage wird diese Liste optimiert und anschließend in einer festgelegten Reihenfolge ausgeführt [BaseX Team, 2016]. Da sich z.B. `db:create` am Ende der festgelegten Reihenfolge befindet, kann demzufolge auf eine neu erstellte Datenbank nicht in der gleichen Abfrage zugegriffen werden. Im Allgemeinen gilt, dass Änderungen erst am Schluss einer Abfrage nach außen sichtbar sind. Das führt dazu, dass beim Ändern von Datenelementen gewisse Änderungen nicht durchgeführt werden können. Bei dem in Abbildung 3 dargestellten Problem würde es daher dazu kommen, dass der `zaehler` zwar auf zwei gesetzt werden würde, die zweite Änderung, den `zaehler` auf den doppelten Wert des `zaehlers` zu setzen, jedoch nicht mehr. Je nach Implementierung führt dies außerdem zu einem Abbruch des Interpreters oder zum Verlust von einer der beiden Änderungen.

Ein weiteres Problem ist, dass Funktionen welche Daten verändern, keine Werte an die aufrufende Funktion zurückgeben können. Dabei ist es wichtig zu wissen, dass dabei auch die aufrufende Funktion als Funktion markiert werden muss, welche Daten verändert.

Um das oben beschriebene Problem zu umgehen wurde im zweiten Versuch die Möglichkeit einer NON-Updating-Expression untersucht. Dies ist eine Möglichkeit die Daten zu kopieren und anschließend diese Kopie der Daten zu bearbeiten.

```
1  let $insertMBA :=
2          copy $insertMBA :=
3     $insertMBA
4          modify
5          (mba:init($insertMBA))
6          return $insertMBA
```

Quellcode 3: Modul SCXML: Verwendung von Copy-Modify

Bei diesem Konzept wird nur eine Kopie mit veränderten Daten erzeugt und keine Datenbankänderung durchgeführt. Dies erlaubt einer Funktion das geänderte MBA an die aufrufende Funktion zurück zu liefern. Nach Durchführung aller Änderungen wird am Schluss die Datenbankänderung durchgeführt. Das Problem bei dieser Lösung ist der hohe Speicherverbrauch. Bei der Bearbeitung müsste das gesamte MBA kopiert werden, da ausführbare Inhalte aus dem `sync`-Modul das komplette MBA verändern können. Dies führt bei komplexen MBAs mit vielen Datenbankänderungen dazu, dass sehr viele Kopien eines einzelnen MBA erzeugt und im Speicher gehalten werden müssten. Deswegen wurde eine Umsetzung mit RESTXQ entwickelt, da RESTXQ mehrere XQuery-Updates mit dem Befehl `rest:forward` und der Benutzung von REST-URLs erlaubt. Diese Lösung wird im nachfolgenden Kapitel genauer erklärt. Das in Quellcode 3 beschriebene Bei-

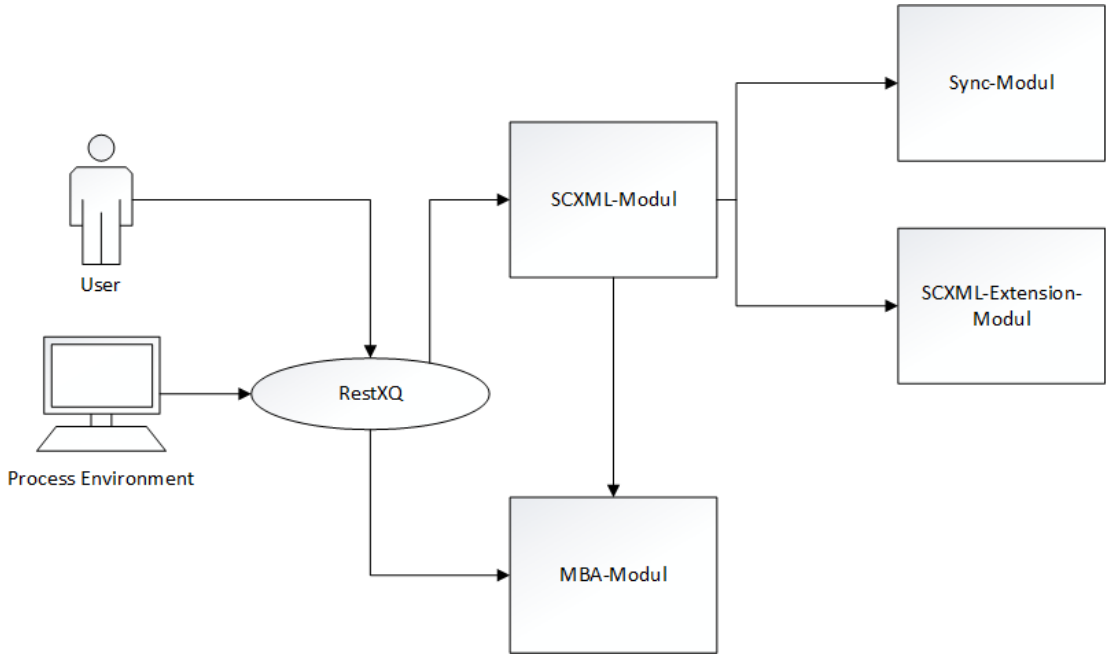

Abbildung 4: Aufbau der Process Engine

spiel entstammt der Funktion `sc:invokeStatesSingle` bei dem das übergebene SCXML-Modell vor den Aufruf modifiziert werden muss.

3.2 Ablauf der SCXML-Interpretation (Process Engine)

Die Process Engine erlaubt das Ausführen eines Zustandsdiagramms vom Anfang bis zum Ende [Dumas et al., 2013]. Sie arbeitet mit dem gesamten MBA, allerdings ist bis auf spezielle ausführbare Inhalte nur das darin enthaltene SCXML-Modell von Bedeutung. Durch die Process Engine werden das Wechseln von Zuständen bzw. die Modifikation von Datenelementen der SCXML-Modelle ermöglicht.

Wie bereits in Unterabschnitt 2.4 erwähnt, wurden für die Entwicklung der Process Engine die beiden Technologien XQuery sowie RESTXQ verwendet. Die RESTXQ-Schnittstelle ist dabei für den grundsätzlichen Ablauf des Interpreters zuständig.

Der Aufbau der SCXML-Interpretation ist dabei in Abbildung 4 dargestellt, die Process Environment sowie der User greifen auf die RESTXQ-Schnittstelle zu. Diese führt gemeinsam mit den internen Modulen die Interpretation durch. Der Zugriff auf die RESTXQ-Schnittstelle erfolgt über eine URL, der hierbei sowohl im Browser als auch über die in Unterabschnitt 3.5 beschriebene Process Environment erfolgen kann. Ein beispielhafter Aufruf der RESTXQ-Schnittstelle lautet z.B: `/initMBA/dbName/collectionName/mbaName`.

Wie bei dem beispielhaften Aufruf angegeben, sind bei allen Funktionsaufrufen innerhalb der RESTXQ-Schnittstelle immer mindestens drei Parameter angegeben. Die nachfolgend aufgelisteten Parameter müssen dabei immer angegeben werden. Der Grund ist, dass diese drei Parameter ein MBA identifizieren und damit das darin enthaltene SCXML-Modell eindeutig bestimmt werden kann.

- dbName
  - Der Name der Datenbank, in dem das MBA gespeichert ist.
- collectionName
  - Der Name der Collection, dem das MBA zugeordnet ist.
- mbaName
  - Der Name des MBAs, um es innerhalb der Collection zu identifizieren.

Innerhalb der RESTXQ-Schnittstelle wird auf andere Funktionen der RESTXQ-Schnittstellen ebenfalls via Link zugegriffen. Zusätzliche Parameter der Funktion, neben den MBA-Daten, werden am Ende der URL hinzugefügt.

```
1  db:output(
2     <rest:forward>
3          {fn:concat('/processPendingTransitions/',
4         string-join(($dbName,$collectionName,$mbaName), '/' ))}
5     </rest:forward>) )
```

Quellcode 4: Modul RESTXQ: Weiterleiten innerhalb der RESTXQ-Schnittstelle

Eine der Einschränkungen von XQuery ist, dass Funktionen, die Daten aktualisieren, keine Werte zurückgeben können. Um dieses Problem zu umgehen wird die BaseX-spezifische Funktion `db:output` benutzt. Sie ermöglicht die Rückgabe von Daten in Kombination mit Funktionen welche Daten aktualisieren. In Quellcode 4 ist die Kombination der Funktionen `db:output` sowie der Weiterleitung zu der Folgefunktion `processPendingTransitions` dargestellt.

Quellcode 5 zeigt den typischen Aufbau einer Funktion der RESTXQ-Schnittstelle. Nach der Angabe des REST-Pfades bzw. der Übergabemethode an die Funktion folgt der übliche XQuery-Funktionskopf. Mit den übergebenen Parametern wird nun das MBA mit der Funktion `mba:getMBA` aus der Datenbank geholt. Anschließend ruft die Funktion zwei Funktionen auf. Die erste Funktion kümmert sich um die tatsächliche Änderung in der Datenbank. Die zweite Funktion `db:output` sorgt dafür, dass sich wie in Quellcode 4 dargestellt , eine Weiterleitung zur nächsten Funktion `processPendingTransitions` erfolgt.

```
1 declare
2 %rest:path("/invokeStates/{$dbName}/{$collectionName}/{$mbaName}")
3 %rest:GET
4 updating function page:invoke(
5     $dbName xs:string, $collectionName xs:string,
6     $mbaName xs:string)
7 {
8   let $mba  := mba:getMBA($dbName, $collectionName, $mbaName)
9   return
10    (sc:invokeStateswithNewDb($mba),
11    db:output(
12      <rest:forward>
13        {
14            fn:concat('/processPendingTransitions/',
15            string-join(($dbName,$collectionName,$mbaName), '/' ))
16        }
17      </rest:forward>) )
18 };
```

Quellcode 5: Modul RESTXQ: Funktion in der RESTXQ-Schnittstelle am Beispiel von invoke

Die RESTXQ-Schnittstelle ist für den grundsätzlichen Ablauf des Interpreters zuständig. Als Grundlage wurde der im SCXML-Standard [Barnett et al., 2015] bereitgestellte Algorithmus verwendet.

In Abbildung 6 ist die Umsetzung des Algorithmus dargestellt. Die Abbildung zeigt die RESTXQ-Funktionen für den Algorithmus sowie die Zusammenhänge zwischen den Funktionen. Der Einstiegspunkt für den Interpreter ist die Funktion `initSCXML`. In dieser wird überprüft ob dies der erste Aufruf des Interpreters ist. Ist dies der Fall müssen zuerst die Initialzustände durch die Funktion `enterStates` betreten werden. Sollte der Interpreter dies bereits erledigt haben, wird über die Steuerungsfunktion `controller` der Ablauf des Interpreters gesteuert. Diese Funktion ist auch in der Tabelle 1 kurz beschrieben und in Abbildung 5 dargestellt.

Innerhalb der Steuerungsfunktion wird geprüft welche Schritte der Interpreter als nächstes durchlaufen soll. Dies ist von mehreren Punkten abhängig, welche nachfolgend aufgelistet sind.

1. Dem Transitionstyp der aktuell zu verarbeitenden Transitionen
2. Dem Status der Variable `running` welche überprüft, ob der Interpreter weiterhin Events abarbeiten soll.
3. Dem Inhalt der Liste der aufzurufenden Zustände.

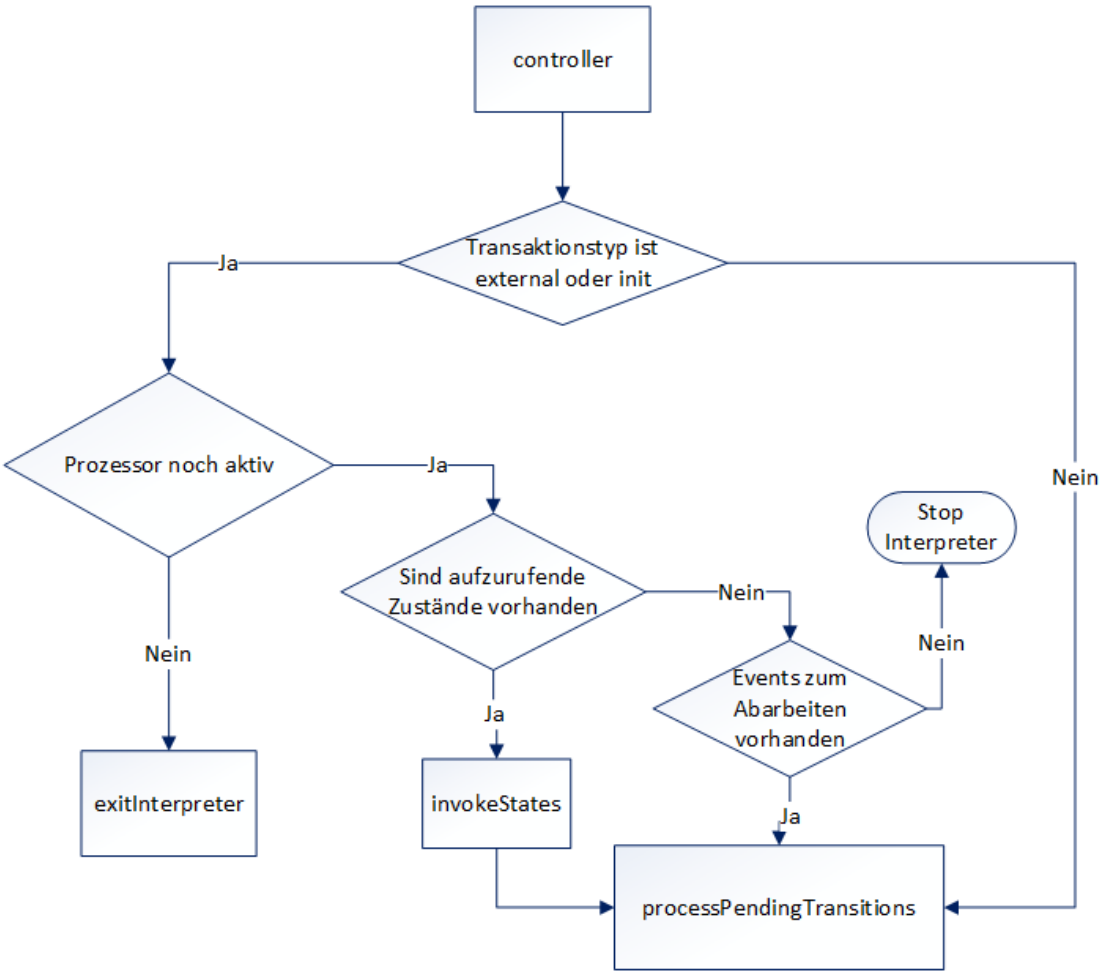

Abbildung 5: RESTXQ Steuerungsfunktion

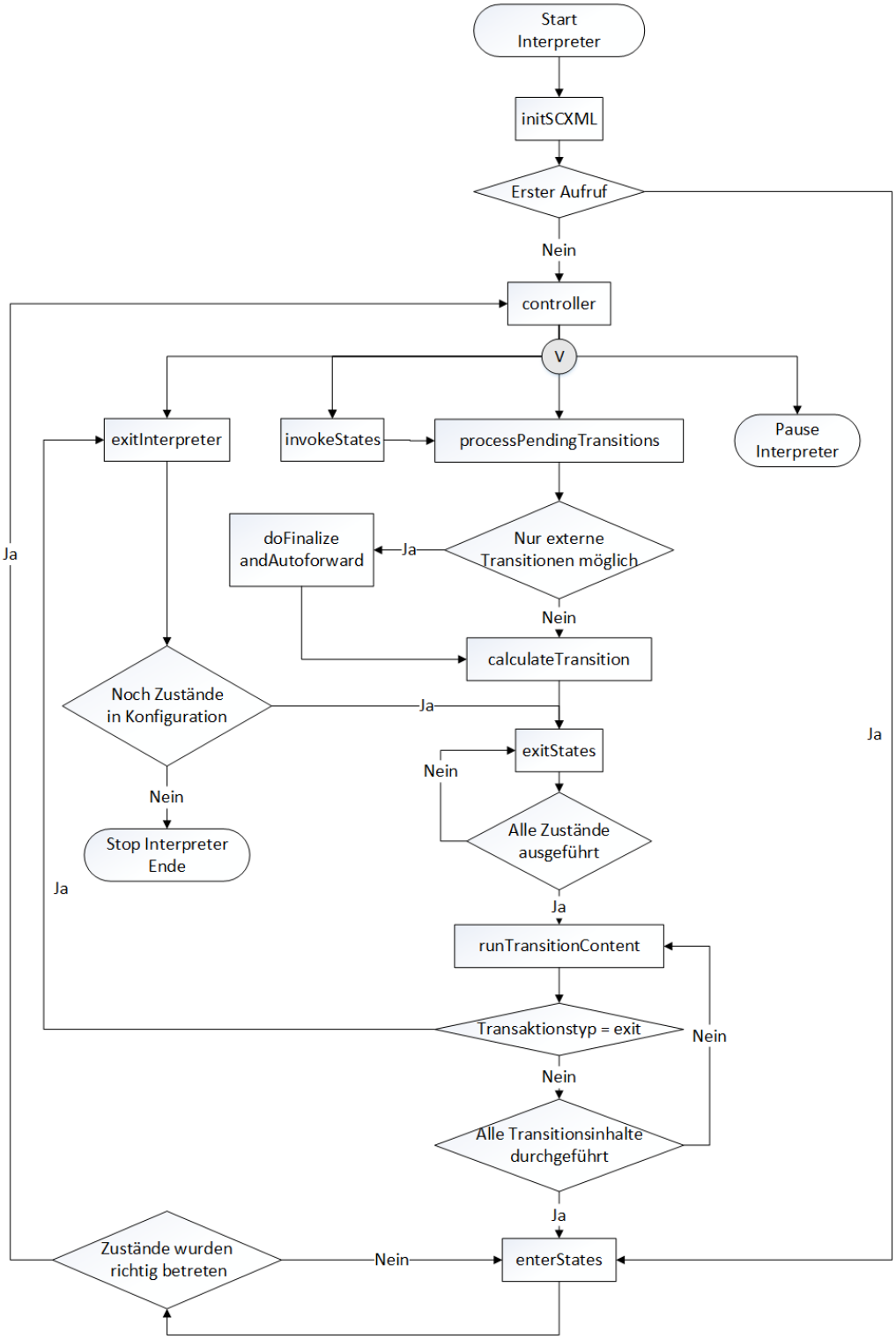

Abbildung 6: RESTXQ Grundsätzlicher Aufbau

Funktion	Parameter	Beschreibung
controller	$dbName xs:string $collectionName xs:string $mbaName xs:string $transType xs:string	Diese Funktion steuert den allgemeinen Ablauf des Interpreters. Wie in Abbildung 5 dargestellt wird überprüft, ob die aktuellen Transitionen vom Typ `inital` oder `external` ist. Sollte dies nicht zutreffen, wird die Funktion `processPendingTransitions` aufgerufen. Ansonsten wird überprüft ob der Interpreter noch aktiv ist. Wenn der Interpreter nicht mehr aktiv ist, wird die Funktion `exitInterpreter` aufgerufen. Schlussendlich wird noch überprüft ob die Liste der noch aufzurufenden Zustände leer ist. Beinhaltet diese Liste noch Werte, wird die Funktion `invokeStates` aufgerufen. Trifft keine der bereits genannten Bedingungen zu, wird der Interpreter unterbrochen und erst wieder durch einen externen Aufruf aktiv.

Tabelle 1: RESTXQ-Schnittstelle: Steuerungsfunktion

4. Die Liste der Transitionen welche aktuell ausführbar sind.

Die erste Entscheidung wird aufgrund des Transitionstyps getroffen. Dies ist insofern wichtig, da die Transitionstypen die unterschiedlichen Prioritäten der Transitionen abbilden. Der Interpreter arbeitet mit den nachfolgenden Transitionstypen:

- init
  - Dieser Typ wird beim ersten (initialen) Betreten des Interpreters verwendet.
  - Die initialen Zustände des SCXML-Modells müssen zuerst betreten werden.
- eventless
  - Dieser Typ wird für Transitionen verwendet, welche nicht durch ein Event ausgelöst werden.
  - Transitionen, welche kein Event benötigen, sind am höchsten gereiht. Sie werden immer als erstes durchgeführt.
- internal
  - Wird für Transitionen verwendet, die das aktuelle interne Event benötigen.
  - Interne Transitionen haben die zweithöchste Priorität und werden nach den ereignislosen Transitionen durchgeführt.
- external
  - Dieser Typ wir für Transitionen verwendet, die das aktuelle externe Event benötigen.
  - Externe Transitionen haben die niedrigste Priorität.
- exit
  - Dieser Transitionstyp wird verwendet, wenn der Interpreter beendet wird.
  - Ist dieser Transitionstyp aktiv, werden die aktuell aktiven Zustände verlassen und deren ausführbare Inhalte durchgeführt.

Funktion	Parameter	Beschreibung
exitContent	$dbName xs:string $collectionName xs:string $mbaName xs:string $counterContent xs:string $counterExit xs:string $transType xs:string	Löscht die zu verlassenden Zustände aus der Konfiguration sowie aus der Liste der aufzurufenden Zustände. Mögliche Historyzustände werden befüllt sowie die aufzurufenden Inhalte werden durchgeführt.
runTransition-Content	$dbName xs:string $collectionName xs:string $mbaName xs:string $counter xs:string $transType xs:string $transType xs:string	Führt den auszuführenden Inhalt der aktuell gültigen Transitionen aus.
enterStates	$dbName xs:string $collectionName xs:string $mbaName xs:string $counterContent xs:string $counterEntry xs:string $transType xs:string	Fügt die zu betretenden Zustände zur Konfiguration und der Liste der aufzurufenden Zustände hinzu. Auch die Initialisierung des Datenmodells sowie die Berechnung und Ausführung von finalen Zuständen wird durchgeführt.
doFinalizeAnd-Autoforward	$dbName xs:string $collectionName xs:string $mbaName xs:string $counterFinalize xs:string	Es wird überprüft ob das Aktuelle Event den Interpreter abbricht. Sollte das Event von einem aufgerufenen Unterprozess geschickt worden sein, wird der jeweilige `finalize` Block aufgerufen. Schlussendlich werden Events, wenn notwendig, an die Unterprozesse weitergeleitet.

Tabelle 2: RESTXQ-Schnittstelle: rekursive Funktionen

Sollte der aktuelle Transitionstyp weder `init` noch `external` sein, wird die Funktion `processPendingTransitions` aufgerufen. Anschließend wird überprüft, ob der Interpreter noch aktiv sein sollte. Ist dies nicht der Fall wird die Funktion `exitInterpreter` aufgerufen. Sollten noch aktive Transitionen vorhanden sein, wird die Funktion `processPendingTransitions` aufgerufen. Ist die Liste der aufzurufenden Zustände nicht leer, wird davor die Funktion `invokeStates` aufgerufen.

Wenn die Steuerungsfunktion feststellt, dass keine Verarbeitung zum aktuellen Zeitpunkt mehr möglich ist, wird die Interpretation des MBAs abgebrochen. Wenn der Interpreter für dieses MBA wieder aufgerufen werden soll, ist ein Aufruf von außerhalb notwendig. Dieser Aufruf kann unter anderem von der in Unterabschnitt 3.5 beschriebenen Process Environment erfolgen.

Im Anschluss an die Funktion `processPendingTransitions` wird die Funktion `calculateTransition` aufgerufen. Sind beim Abschluss der Funktion `processPendingTransitions` nur externe Transitionen möglich, wird davor die Funktion `doFinalizeAndAutoforward` aufgerufen.
`doFinalizeAndAutoforward` kümmert sich um die Kommunikation zwischen den aufgerufenen Interpretern und den aufrufenden Interpretern. Die Parameter zum Aufrufen der bereits genannten Funktionen sind unter anderem in Tabelle 2 aufgezählt. Nachdem die aktuell zu bearbeitenden Transitionen in der Funktion `calculateTransition` festgestellt wurden, werden die zu verlassenden Zustände in der Funktion `exitContents` verlassen. Alle Funktionen, die sich mit ausführbaren Inhalten beschäftigen sind rekursiv. Das heißt sie rufen sich, wie in Quellcode 6 angegeben, mit einem Zähler selbst wieder auf. Solange nicht alle Zustände und Transitionen abgearbeitet wurden, rufen sich die Funktionen selbst wieder auf.

```
1  db:output(
2     <rest:forward>
3         {fn:concat('/exitStates/',
4         string-join(($dbName,$collectionName,$mbaName,
5         $counterContent+1, $counterExit, $transType), '/' ))}
6     </rest:forward>))
```

Quellcode 6: Modul RESTXQ: Rekursiver Funktionsaufruf

Die beiden Zählervariablen `counterContent` bzw. `counterExit` in Quellcode 6 stehen dabei jeweils für den ausführbaren Inhalt bzw. den aktuell zu verlassenden Zustand. Diesen Funktionsaufbau haben sowohl die Funktionen `exitStates`, `runTransitionContent`, `enterStates` als auch `doFinalizeAndAutoforward`. Alle drei Funktionen können dabei sowohl den ausführbaren Inhalt aus dem Standard, als auch benutzerspezifische ausführbare Inhalte ausführen. Durch den rekursiven Aufruf der Funktionen über die RESTXQ-Schnittstelle wird sichergestellt, dass das Problem wie in Abbildung 3 dargestellt, nicht auftritt.

Im nächsten Schritt wird der ausführbare Inhalt der Transition in der Funktion `runTransitionContent` ausgeführt. Zu Beginn der gerade genannten Funktion wird überprüft, ob der Transitionstyp `exit` ist. Sollte dies der Fall sein, wird die Funktion `exitInterpreter` erneut aufgerufen und das Beenden des Interpreters fertiggestellt. Üblicherweise wird jedoch die Funktion `enterStates` aufgerufen, welche die Zustände der Transitionsziele betritt. Am Ende der Funktion `enterStates` wird wieder die Steuerungsfunktion `controller` aufgerufen. Durch diesen Aufruf wird der Kreislauf geschlossen und der Interpreter fängt von vorne an.

In der Tabelle 3 werden die RESTXQ-Funktionen sowie deren Funktionsparameter genauer beschrieben. Der Funktionsaufruf der nachfolgenden Funktionen wird dabei immer mit `rest:forward`-Elementen durchgeführt, wie in Quellcode 4 dargestellt.

Die Funktion `calculateTransitions` berechnet die aktiven Zustände und speichert diese. Anschließend werden auch die, aufgrund der gültigen Transitionen, zu betretenden Zustände gespeichert. Das ist notwendig, da sich durch das Verlassen der Zustände und dem Ausführen der darin enthaltenen ausführbaren Inhalte Daten verändern können. Dadurch könnten Transitionsbedingungen anders evaluierten werden und bei einer erneuten Berechnung würden sich die zu betretenden Zustände verändern. Das Speichern erlaubt das Ändern der jeweiligen Daten bereits in der Funktion `exitStates` und ermöglicht, dass in der Funktion `enterStates` trotzdem die richtigen Zustände betreten werden.

Eine weitere Funktion die zur Benutzung des Interpreters notwendig ist, ist die Funktion `initMBA` in Tabelle 4. Sie ist notwendig, da für die Benutzung des Interpreters einige Elemente eingefügt werden müssen. Dazu gehören neben den Systemvariablen auch die vom Standard festgelegten Parameter. Die Funktion `initMBA` fügt diese Parameter zum SCXML-Modell hinzu. Erst danach kann der Interpreter mit dem MBA arbeiten. Des Weiteren wird das MBA als geändert markiert und aus der Liste der neuen MBAs entfernt. Durch diese Änderung des Status wird sichergestellt, dass ein MBA nicht doppelt initialisiert wird.

Ein Teil der Benutzungsschnittstelle ist ebenfalls im RESTXQ-Modul umgesetzt. Da diese Funktionen jedoch nicht in Beziehung mit den restlichen Funktionen der RESTXQ-Schnittstelle stehen, wird die Benutzungsschnittstelle erst in Abschnitt 4 erläutert.

3.3 Interne Module zur SCXML-Interpretation

Die Funktionen der in Unterabschnitt 3.2 beschriebenen RESTXQ-Schnittstelle rufen XQuery-Funktionen auf, die auf mehrere Module verteilt sind. Das Modul MBA ist für die Verwaltung des MBAs zuständig. Neben den Zugriffsfunktionen für das MBA ist auch die Erstellung des MBAs in diesem Modul enthalten. Außerdem ist die Verwaltung der Variablen des Interpreters, welche im SCXML gespeichert

Funktion	Parameter	Beschreibung
initSCXML	$dbName xs:string $collectionName xs:string $mbaName xs:string $counter xs:string	Erstellt das Datenmodell und entfernt das MBA von der Liste der geänderten MBAs. Auch die Initialisierung des SCXML-Modelles wird durchgeführt.
calculateTransitions	$dbName xs:string $collectionName xs:string $mbaName xs:string $transType xs:string	Berechnet und speichert die aktuell gültigen Transitionen sowie die Zustände, welche durch diese Transitionen verlassen werden.
processPending-Transitions	$dbName xs:string $collectionName xs:string $mbaName xs:string	Stellt fest welche Transitionen aktuell verarbeitet werden müssen und leitet je nach gültigem Transitionstyp an die dementsprechende Funktion weiter.
exitInterpreter	$dbName xs:string $collectionName xs:string $mbaName xs:string $transType xs:string $counter xs:string	Diese rekursive Funktion beendet den aktuellen SCXML-Interpreter indem alle aktiven Zustände verlassen werden. Sollte sich der Interpreter in einem finalen Zustand befinden, werden die `Done`-Events an den aufrufenden Interpreter gesendet.
invokeStates	$dbName xs:string $collectionName xs:string $mbaName xs:string $transType xs:string	Diese Funktion nimmt die in der Liste der aufzurufenden Zustände und führt die aufzurufende Interpretation aus.

Tabelle 3: RESTXQ-Schnittstelle: sonstige Funktionen

Funktion	Parameter	Beschreibung
initMBA	$dbName xs:string $collectionName xs:string $mbaName xs:string	Die für den Interpreter benötigten Elemente werden dem MBA hinzugefügt. Außerdem wird der Status des MBAs aktualisiert.

Tabelle 4: RESTXQ-Schnittstelle: Initiierungsfunktion

sind, in diesem Modul zu finden. Das zweite Hauptmodul ist das Modul SCXML, das die Funktionen zum Interpretieren des SCXML-Modells beinhaltet. Wie in Abbildung 4 dargestellt greift die RESTXQ-Schnittstelle auf die beiden Hauptmodule zu, welche die verbleibenden Module verwenden. Die sonstigen eingesetzten Module `SCXML-Extension` sowie `SYNC` sind ebenfalls im Unterunterabschnitt 3.3.3 beschrieben.

3.3.1 Verarbeitung von SCXML-Dokumenten

Das SCXML-Modul beinhaltet die wichtigsten Funktionen zur Verarbeitung der Prozessmodelle eines MBAs. Dazu zählen die Berechnung der zu betretenden und zu verlassenden Zustände, die Verarbeitung von Events sowie alle sonstigen Funktionen, die zur Interpretation eines SCXML-Modells notwendig sind.

Funktion	Parameter	Beschreibung
sc:getNextExternalEvent	$dbName xs:string $collectionName xs:string $mbaName xs:string	Holt sich das MBA und lädt das nächste externe Event als aktuell aktives Event.
sc:getNextInternalEvent	$dbName xs:string $collectionName xs:string $mbaName xs:string	Holt sich das MBA und lädt das nächste interne Event als aktuell aktives Event.

Tabelle 5: Modul SXCML: Funktionen zum Erhalten von internen und externen Events

Wie bereits im vorangegangenen Kapitel beschrieben, ist es notwendig eine Trennung zwischen externen und internen Events zu treffen. In Tabelle 5 sind zwei Funktionen zum Bestimmen des nächsten gültigen Events vorhanden. Je nachdem, welcher Transitionstyp gerade aktiv ist, wird die zugehörige Funktion aufgerufen. In Tabelle 6 sind die Funktionen zum Ausführen der ausführbaren Inhalte aufgezählt.

Funktion	Parameter	Beschreibung
sc:getExecutable-ContentsExit	$dbName xs:string $collectionName xs:string $mbaName xs:string $states element()*	Gibt die ausführbaren Inhalte der zu verlassenden Zustände zurück.
sc:getExecutable-ContentsTransition	$dbName xs:string $collectionName xs:string $mbaName xs:string	Gibt die ausführbaren Inhalte der aktuellen Transitionen zurück.
sc:getExecutable-ContentsEnter	$dbName xs:string $collectionName xs:string $mbaName xs:string $state xs:string $historyContent xs:string	Gibt die ausführbaren Inhalte der zu betretenden Zustände zurück.
sc:runExecutable-Content	$dbName xs:string $collectionName xs:string $mbaName xs:string $content	Führt den ausführbaren Inhalt aus.

Tabelle 6: Modul SCXML: Funktionen für ausführbaren Inhalt

Die Funktion `sc:runExecutable` wird zum Kategorisieren von ausführbaren Inhalten verwendet. Sie ruft dazu entweder die zuständige Funktion auf oder führt den Inhalt innerhalb der Funktion aus. In SCXML-Modellen können ausführbare Inhalte in `transition`-, `onExit`-, `onEntry`- sowie `finalize`-Blöcken vorkommen. Dabei ist besonders wichtig, dass die Inhalte innerhalb eines Blockes in der angegebenen Reihenfolge (*document order*) aufgerufen werden. Dies wird mit dem in Quellcode 4 beschriebenen Befehl `restforward` erreicht.

Der SCXML-Interpreter unterstützt verschiedene Arten von ausführbaren Inhalten. So werden neben den im Standard [Barnett et al., 2015] genannten Inhalten zusätzlich weitere, benutzerspezifische Funktionen unterstützt. Letztere sind nachfolgend aufgezählt und kurz beschrieben.

- sync:assignAncestor
  - Fügt dem angegeben MBA eine Konkretisierung hinzu. Dazu verwendet sie die im Standard definierte Funktion `sc:assign`.
- sync:sendAncestor
  - Sendet ein Event an alle übergeordneten Hierarchielevel dieses MBAs. Solche Events werden den MBAs als externe Events hinzugefügt.
- sync:assignDescendants
  - Fügt dem angegeben MBA neue Konkretisierungen hinzu. Dabei wird die im Standard definierte Funktion `sc:assign` verwendet.
- sync:sendDescendants
  - Sendet ein Event an alle Konkretisierungen des angegebenen MBAs und fügt ihnen das Event als externes Event hinzu.
- sync:newDescendant
  - Erstellt eine neue Konkretisierung, die als Nachfolger des aktuellen SCXML-Modells eingefügt wird.
- sc:getValue
  - Speichert den aktuellen Wert einer angegebenen Variable im Log für den User. Anschließend kann mit der Funktion `getLog` darauf zurückgegriffen werden.

Eine Art ausführbaren Inhalts ist `sc:send`. Dieses Element ermöglicht das Senden von Events sowohl an das eigene als auch an fremde MBAs. Dabei wird zuerst das zu sendende Event erstellt, indem alle übergebenen Parameter ausgewertet werden. Sollte dabei ein Fehler auftreten, wird kein Event versendet, sondern ein Fehlerevent generiert. Dieses wird in die interne Eventliste des aktuellen SCXML-Modells hinzugefügt. Falls das Event fehlerfrei erstellt werden konnte, wird das MBA ermittelt, dem das Event hinzugefügt werden soll. Dazu wird die Angabe des `location`-Attributs verwendet. Der SCXML-Interpreter unterstützt die nachstehenden Angaben bei der Auswertung dieses Attributs.

1. Keine Angabe des `location`-Attributs
   - Ist kein Wert angegeben, wird das Event dem aktuellen MBA als externes Event hinzugefügt.
2. #_internal
   - Das aktuelle Event wird dem aktuellen MBA als internes Event hinzugefügt.
3. #_parent
   - Sollte das aktuelle MBA ein aufgerufenes MBA sein, wird das Event dem aufrufenden MBA als externes Event hinzugefügt.
4. #_scxml_
   - Im Anschluss an diese Lokationsangabe erfolgt die Angabe des MBAs in der folgenden Form: `mbaName:dbName,collectionName,mbaName`.
   - Dem angegebenen MBA wird ein externes Event hinzugefügt.
5. #_
   - Im Anschluss an diese Lokationsangabe erfolgt die Angabe des MBAs in der Form `mbaName:dbName,collectionName,mbaName`.
   - Dem angegebenen MBA wird ein externes Event hinzugefügt.

Für die letzten beiden Lokationsangaben wird die Funktion `sc:getMBAFromText`, welche in Tabelle 10 beschrieben ist, verwendet. Diese wandelt das in der `location`-Variable angegebene MBA in das tatsächliche MBA um. Kann die `location`-Angabe nicht korrekt ausgewertet werden, wird ein Fehlerevent zur aktuellen Liste der externen Events hinzugefügt.

Die Funktion `sc:getExecutableContentExit` gibt die in den `onExit`-Blöcken der übergebenen Zustände vorhandenen ausführbaren Inhalte in

Funktion	Parameter	Beschreibung
sc:initMBARest	$dbName xs:string $collectionName xs:string $mbaName xs:string	Initialisierung des MBAs sowie Entfernung des MBAs aus der Liste der geänderten MBAs.
sc:initSCXMLRest	$dbName xs:string $collectionName xs:string $mbaName xs:string	Setzt die zu betretenden Zustände bei dem ersten Durchlauf des Interpreters.
sc:removeFrom-InsertLog	$dbName xs:string $collectionName xs:string $mbaName xs:string	Entfernt das MBA von der Liste der neu hinzugefügten MBAs.
sc:markAsUpdated	$dbName xs:string $collectionName xs:string $mbaName xs:string	Markiert das MBA als geändert, indem sie das MBA zur Liste der geänderten MBAs hinzufügt.
sc:removeFrom-UpdateLog	$dbName xs:string $collectionName xs:string $mbaName xs:string	Entfernt das MBA von der Liste der noch zu ändernden MBAs.
sc:initDatamodel	$dbName xs:string $collectionName xs:string $mbaName xs:string	Ist das Binding des SCXML-Modells eine späte Bindung, wird mit dieser Funktion das Datenmodell initialisiert. Sollte dies nicht der Fall sein, wird das Initialisieren des Datenelements bereits beim allgemeinen Initialisieren des SCXML-Modells erledigt.
mba:create-Datamodel	$mba element()	Initialisiert das Datenmodell bei der Initialisierung des MBAs für den Interpreter und beachtet dabei den Bindungszeitpunkt der Variable.

Tabelle 7: Modul SCXML: Funktionen welche das MBA Modul aufrufen

umgekehrter Reihenfolge an die aufrufende RESTXQ-Funktion zurück. Beim Berechnen der ausführbaren Inhalte der zu betretenden Zustände (`sc:getExecutableContentEnter`) müssen neben den Inhalten der `onEntry`-Blöcke auch jene der `initial`-Blöcke beachtet werden.

Tabelle 7 enthält die Funktionen, welche für Änderungen am MBA direkt notwendig sind. Dazu gehören die Funktionen zum Initialisieren des MBAs sowie des SCXML-Modells. Des Weiteren sind jene Funktionen beschrieben, die MBAs als geändert oder neu markieren. Diese Funktionen sind sehr ähnlich aufgebaut. Zuerst wird das zu bearbeitende MBA aufgrund der Parameter `dbName`, `collectionName` sowie `mbaName` ermittelt. Anschließend wird es an die betreffende Funktion im MBA-Modul weitergeleitet.

Die Funktion `sc:initSCXML` dient dazu, die aktuell aktiven Zustände beim ersten Aufruf des Interpreters in der dazugehörigen Variable zu speichern. Wenn die Liste der aktiven Zustände bereits befüllt wurde, wird dieser Schritt übersprungen. Die Funktion `sc:initDatamodel` dient zur Initialisierung von einzelnen Datenelementen in Zuständen. Dies ist nur notwendig, wenn späte Bindung (*late binding*) aktiv ist. Da bei später Bindung die Datenelemente eines Zustandes erst beim erstmaligen Betreten des Zustandes initialisiert werden, wird diese Funktion beim Betreten der Zustände in der Funktion `sc:enterStatesSingle` aufgerufen.

Die in der Tabelle 7 ebenfalls angegebene Funktion `mba:createDatamodel` dient der Initialisierung der Datenelemente eines MBAs. Auch diese Funktion beachtet den Zeitpunkt der Bindung, da bei später Bindung nur die Datenelemente auf SCXML-Ebene initialisiert werden müssen.

Bei der Initialisierung werden die Datenelemente initialisiert, jedoch muss darauf geachtet werden, dass Datenelemente, die innerhalb eines `invoke`-Elements angegeben sind, nicht initialisiert werden. Diese Elemente werden erst bei der Erzeugung des MBAs initialisiert. Bei der Initialisierung eines Datenelements werden die drei nachstehenden Angaben zur Berechnung des gültigen Wertes unterstützt. Die Angabe erfolgt über Attribute in den Datenelementen.

- expr
  - Wie auch bei der Angabe der Eventbedingungen kann auch bei Datenelementen ein Ausdruck zum Evaluieren angegeben sein. Zur Berechnung wird die in Tabelle 11 angegebene Funktion `sc:eval` verwendet.
- src
  - Nach diesem Attribut erfolgt die Angabe einer Datei inklusive ihres Verzeichnispfades. Die Angabe der Datei muss dabei mit `file:` beginnen.
- systemsrc

Funktion	Parameter	Beschreibung
sc:updateRunning	$dbName xs:string $collectionName xs:string $mbaName xs:string	Überprüft das aktuelle Event und setzt den Status des Interpreters. Sollte das aktuelle Event ein abbrechendes Event sein, wird die Variable `running` auf falsch gesetzt.
sc:exitStatesSingle	$dbName xs:string $collectionName xs:string $mbaName xs:string $stateToExit $type	Führt die notwendigen Änderungen durch, die vor dem Verlassen eines Zustandes erforderlich sind.
sc:enterStatesSingle	$dbName xs:string $collectionName xs:string $mbaName xs:string $state	Führt die beim Betreten eines Zustands notwendigen Änderungen durch.
sc:exitInterpreter	$dbName xs:string $collectionName xs:string $mbaName xs:string	Ist der Interpreter durchgelaufen, wird ein `done`-Event an den aufrufenden Interpreter übergeben, wodurch dieser den Abschluss des aufgerufenen Zustandes feststellen kann.

Tabelle 8: Modul SCXML: Funktionen zur Verarbeitung von Zustandswechseln

- Diese Angabe erlaubt den Zugriff auf die Systemvariablen wie z.B. $_ioprocessors. Die Auswertung erfolgt mithilfe von `sc:evalWithError`.

Sollte bei der Evaluierung der einzelnen Attribute ein Fehler passieren wird ein internes Fehlerevent erzeugt. Das Datenelement wird gelöscht und ein neues leeres Datenelement ohne Zuweisung angelegt.

Die Funktionen zum Verarbeiten von Zustandswechseln sind in Tabelle 8 beschrieben. Die Funktion `exitStatesSingle` führt die Datenbankänderungen durch, die beim Verlassen zusätzlich zum Ausführen der ausführbaren Inhalte sowie dem Entfernen des Zustandes aus der Liste der aktiven Zustände durchzuführen sind. Dazu gehört das Befüllen der Historyzustände, das Abbrechen von aufgerufenen SCXML-Interpretern und das Entfernen von Zuständen von der Liste der noch auszuführenden Zustände. Der Interpreter unterscheidet dabei zwischen zwei unterschiedlichen Historyzuständen. Während bei `deep`-Historzuständen alle

Funktion	Parameter	Beschreibung
sc:invokeStates	$mba element()	Erstellt eine neue Datenbank mit den im aufzurufenden Zustand angegebenen SCXML-Text. Dazu wird ein leeres MBA mit den Daten, Parametern und weiteren Informationen befüllt. Der eigentliche Aufruf dieses neuen MBAs erfolgt wieder über die Process Environment.
sc:autoForward	$dbName xs:string $collectionName xs:string $mbaName xs:string	Nimmt das aktuelle Event und baut es als neues Event zusammen. Anschließend wird es an den aufgerufenen Prozess weitergeleitet.

Tabelle 9: Modul SCXML: Funktionen zum Ausführen von `invoke`-Elementen

Zustände gespeichert werden, in denen sich der Interpreter in diesem Moment aktuell befindet, werden bei der `shallow`-History nur die oberflächlichen Zustände gespeichert. Die Funktion `sc:enterStatesSingle` führt alle Änderungen durch, die beim Betreten eines Zustands notwendig sind. Wenn der zu betretende Zustand ein finaler Zustand ist, fällt darunter auch das Erstellen des `done`-Events. Sollte dabei festgestellt werden, dass sich das gesamt SCXML-Modul in einem finalen Zustand befindet, wird der Status des SCXML-Moduls über die Funktion `sc:updateRunning` neu gesetzt. Des Weiteren fügt sie den Zustand sowohl zur Liste der aktuellen Zustände als auch zur Liste der neu aufzurufenden Zustände hinzu. Um bei später Bindung die Initialisierung des Datenmodells zu ermöglichen, wird zusätzlich die Funktion `sc:initDatenmodell` aufgerufen. Schlussendlich wird der Zustand in dieser Funktion noch zu der Liste der aktiven Zustände hinzugefügt.

Die Funktionen `sc:updateRunning` und `sc:exitInterpreter` sind eng miteinander verknüpft. Nach einer Änderung des Status in der Funktion `sc:updateRunning` wird das MBA in der Funktion `sc:exitInterpreter` korrekt verlassen.

Die Funktion `sc:invokeStates` dient zum Aufruf des Inhalts von `invoke`-Elementen. Diese Elemente werden dazu genutzt, um eine neue Instanz des Interpreters aufzurufen. Dazu wird ein neues MBA erzeugt, welches die Erstellung einer neuen Datenbank voraussetzt. Damit die Process Environment auf das neu erstellte MBA aufgreifen kann, ist die Namensgebung der Datenbank vorgegeben. Der Name der Datenbank besteht aus der Zeichenkette `invoke`, der eine zufällig gene-

Funktion	Parameter	Beschreibung
sc:getMBAFromText	$src xs:string	Dies ist eine Hilfsfunktion, die einen Ausdruck in das dazugehörige MBA übersetzt. Der Übergabeparameter ist wie in Quellcode 8 aufgebaut.
sc:matchesEvent-Descriptors	$eventName xs:string $eventDescriptors as xs:string*	Überprüft, ob das übergebene Event mit dem der Transaktion entsprechend der vorgegebenen Regeln übereinstimmt. Die dazu verwendete Regular Expression ist in Quellcode 7 dargestellt.
sc:isInFinalState	$state element() $configuration element()* $enterState element()	Überprüft, ob sich das MBA in einem finalen Zustand befindet.

Tabelle 10: Modul SCXML: Hilfsfunktionen

rierte Nummer angehängt wird. In dieser Datenbank wird nun eine neue Collection mit dem Namen `invoke` sowie anschließend ein einfaches MBA ebenfalls mit dem Namen `invoke` eingefügt. Diesem MBA wird das im `invoke`-Element referenzierte bzw. beinhaltete SCXML eingefügt. Der Aufbau des MBA in der Datenbank ist der der Abbildung 8 dargestellt.

Um die Kommunikation zwischen dem neuen und dem aufrufenden MBA zu ermöglichen, wird eine Referenz auf das neue MBA im aufrufenden gespeichert. Für die Kommunikation in die andere Richtung wird in dem neu erzeugten MBA ebenfalls das aufrufende MBA referenziert.

Das `invoke`-Element erlaubt unter anderem das Übergeben von Parametern an das neu erstellte MBA. Die Funktion `invokeStates` übergibt diese Daten an das neue MBA. Sollte bei diesen Parametern oder bei der Erstellung und Kontrolle des neuen MBAs Probleme auftreten, wird kein MBA erstellt. Die Funktion `sc:autoForward` erlaubt dass Events, die an das aufrufende MBA gesendet wurden, auch an das aufgerufene MBA weitergeleitet werden. Dazu wird das aktuelle Event in alle Einzelteile aufgeteilt und als neues Event an das aufgerufene MBA versendet.

Für die in Tabelle 10 aufgezählten Funktionen `sc:matchesEventDescriptors` wird die Regular Expression von Quellcode 7 verwendet. Sie ermöglicht es, die in den Transitionen genannten Eventbedingungen mit den aktuell gültigen Events zu

Funktion	Parameter	Beschreibung
sc:evaluateCond	$cond $dataModels	Überprüft, ob die übergebene Bedingung wahr oder falsch ist. Sollte die Bedingung nicht evaluiert werden können, wird sie als falsch interpretiert.
sc:eval	$expr xs:string $dataModels element()*	Überprüft, ob sich die übergebene Bedingung als wahr oder falsch auswerten lässt. Sollte die Bedingung nicht evaluiert werden können, wird ein Fehler geworfen.
sc:evalWithError	$expr xs:string $dataModels	Überprüft, ob sich die übergebene Bedingung als wahr oder falsch auswerten lässt. Sollte die Bedingung nicht evaluiert werden können, wird der Fehlercode zurückgegeben.

Tabelle 11: Modul SCXML: Evaluierungsfunktionen

vergleichen. Insbesondere die Beachtung der Suffixe bzw. Platzhalter ist bei der Evaluierung wichtig.

```
1 fn:matches(
2     $descriptor || '|' || $eventName,
3     '^((([a-zA-Z0-9\.]+)\.\*\|\3\.[a-zA-Z0-9\.]+)|
4     (\*\|[a-zA-Z0-9\.]+)|((([a-zA-Z0-9\.]|\.)+)\|(\6))|
5     (([a-zA-Z0-9\.]+)\|\10\.[a-zA-Z0-9\.]+))$')
```

Quellcode 7: Modul SCXML: Regular Expression zum Überprüfen von Events

Die ebenfalls in Tabelle 10 beschriebene Funktion `sc:getMBAFromText` wird zur Auswertung von Lokationsangaben verwendet. Ihr einziger Parameter ist eine Zeichenkette, welche die Bezeichnung des MBAs sowie der Collection und den Datenbanknamen beinhaltet. Sie werden mit Beistrichen getrennt, des Weiteren muss der Name der Datenbank mit dem Prefix `mba:` versehen werden. Ein beispielhafter Aufruf der Funktion ist in Quellcode 8 zu sehen. Der Wert `myMBAse` steht dabei für die Datenbank, `JohannesKeplerUniversity` für den Namen der Collection und `InformationSystems` für den Namen des MBAs. Diese Funktion wird sowohl beim Senden von Events an andere MBAs als auch beim Senden von Nachrich-

ten zwischen aufgerufenen und aufrufenden SCXML verwendet. Sie ermöglicht es, fremde MBAs einfach zu referenzieren und wieder aufzurufen.

```
1 $mba := sc:getMBAFromText(
2         "mba:myMBAse,JohannesKeplerUniversity,InformationSystems")
```

Quellcode 8: Modul SCXML: Funktion sc:getMBAFromText

Funktionen zur Auswertung von Bedingungen bzw. Daten sind in Tabelle 11 beschrieben. Sie werden beim Berechnen der aktuell gültigen Transitionen, beim Befüllen von Parametern und bei der Initialisierung von Datenmodellen benötigt. Die drei Evaluierungsfunktionen sind grundsätzlich gleich aufgebaut, unterscheiden sich jedoch bei der Auswertung und insbesondere bei den Rückgabewerten im Fehlerfall.

Zum Auswerten wird die BaseX-Funktion `xquery:eval` verwendet, welche die Modulverknüpfungen und Bedingungskürzel aus dem Modul `scxml`-Extension erhält. Zusätzlich werden das aktuelle Datenmodell und die zu überprüfende Bedingung übergeben.

In Tabelle 12 sind die Funktionen beschrieben, welche sich mit Transitionen beschäftigen. Neben den Funktionen für die Auswahl der aktuell gültigen Transitionen (`sc:selectEventlessTransitions` und `sc:selectTransitions`) sind auch die Funktionen zur Berechnung der Zustände, die aufgrund von Transitionen betreten bzw. verlassen werden müssen, enthalten. Zur Berechnung der Transitionen werden die jeweiligen Bedingungen der aktuell möglichen Transitionen evaluiert. Bei der Funktion `sc:selectEventlessTransitions` wird zusätzlich überprüft, ob die Eventbedingung nicht existiert bzw. ob sie leer ist. Im Gegensatz dazu wird bei der Funktion `sc:selectTransitions` auch die Eventbedingung mit dem aktuell aktiven Event überprüft. Dazu wird die Regular Expression aus Quellcode 7 verwendet. Beide Funktionen rufen anschließend noch die Funktion `sc:removeConflictionTransitions` auf, da ansonsten mehrere Transitionen zurückgegeben werden könnten, jedoch außerhalb von parallelen Zuständen immer nur eine Transition aktiv sein darf.

Die Funktionen zur Berechnung der zu verlassenden bzw. zu betretenden Zustände sind in Tabelle 13 angegeben. Die Bestimmung der zu verlassenden Zustände (*ExitSet*) ist im Gegensatz zu den zu betretenden Zuständen relativ einfach, da hier lediglich all jene Zustände verlassen werden müssen, in denen die Transition ihren Ursprung hat, sowie deren Vorfahren. Dabei ist jedoch auch zu beachten, dass nur Zustände verlassen werden, die nicht durch das Ziel der Transition wieder betreten werden würden. Die Funktion, die diese Zustände berechnet, ist in Quellcode 9 dargestellt.

Funktion	Parameter	Beschreibung
sc:select-EventlessTransitions	$configuration element()* $dataModels element()*	Berechnet jene Transitionen für ein MBA, welche auch ohne Event aktiv werden. Diese Transitionen haben die höchste Priorität.
sc:selectTransitions	$configuration element()* $dataModels element()* $event	Berechnet für ein MBA diejenigen Transitionen, die aufgrund des aktuellen Events und ihrer Bedingung aktiv sind.
sc:select-TransitionsoC	$configuration element()* $dataModels element()* $event	Selektiert die möglichen Transitionen, die aufgrund der manuellen Eingabe eines Events aktiv sein können.
sc:remove-Conflicting-Transitions	$configuration element()* $transitions element()*	Mehrere Transitionen werden an diese Funktion übergeben um danach nur diejenigen Transitionen zu erhalten, welche wirklich genommen werden. Nur bei parallelen Zuständen kann mehr als eine Transition aktiv sein.

Tabelle 12: Modul SCXML: Funktionen zur Berechnung von Transitionen

Funktion	Parameter	Beschreibung
sc:computeExitSet	$configuration element()* $transitions element()*	Berechnet die Zustände, die aufgrund der gewählten Transitionen bzw. der aktuell aktiven Zustände verlassen werden.
sc:computeExitSet2	$configuration element()* $transitions element()*	Berechnet die Zustände, die aufgrund der gewählten Transitionen bzw. der aktuell aktiven Zustände verlassen werden.
sc:computeEntry	$transitions element()*	Berechnet die Zustände, die durch die ausgewählten Transitionen betreten werden.
sc:computeEntryOc	$transitions element()*	Berechnet für die Benutzeroberfläche diejenigen Zustände, die aufgrund der manuellen Eingabe eines Events betreten werden können.
sc:computeEntryInit	$scxml	Berechnet die zu betretenden Zustände bei dem initialen Aufruf des Interpreters.

Tabelle 13: Modul SXCML: Funktionen zum Berechnen des Entry-/ExitSets

```
1 declare function sc:computeExitSet(
2     $configuration element()*,
3     $transitions element()*)
4     as element()*{
5     let $statesToExit :=
6       for $t in $transitions
7       let $domain := sc:getTransitionDomain($t)
8           for $s in $configuration
9               return
10                  if (sc:isDescendant($s, $domain)) then $s
11                  else ()
12    return $statesToExit
13 };
```

Quellcode 9: Modul SCXML: Berechnung der zu verlassenden Zustände

Die Berechnung der zu betretenden Zustände ist komplexer und wird bei der nachfolgenden Funktion `sc:addDescendantStatesToEnter` genauer erklärt. Der Unterschied zwischen den beiden Funktionen `sc:computeEntry` und `sc:computeEntryInit` liegt bei der Berechnung des Startwertes und der Übergabeparameter. Während bei der zuletzt genannten Funktion das gesamte SCXML-Element als Startelement und davon insbesondere die initialen Elemente verwendet werden, erhält die andere Funktion die aktuell gültigen Transitionen und berechnet dadurch das Startset.

In Tabelle 14 sind die drei Funktionen aufgeführt, die für die Durchführung von ausführbaren Inhalten zuständig sind. Die Funktion `sc:assign` ermöglicht das Ändern von Daten. Dazu wird der Wert des Ausdrucks an der Stelle des angegebenen Ortes (`location`) gespeichert. Über die Angabe des Typs ist es möglich, den Ablauf des Assigns genauer zu steuern. Damit können auch Einfüge- bzw. Löschaktionen mit diesem Element durchgeführt werden.

Die Funktion `sc:getValue` funktioniert sehr ähnlich, jedoch ist der Speicherort des Ergebnisses von der aktuell gültigen `EventId` abhängig. Auch in der Funktion `getLog` wird eine Datenbankänderung durchgeführt. Sie speichert Werte bzw. vorgegebene Texte in dem für die Benutzerinnen und Benutzer einsehbaren Log. Weitere Informationen gibt es im Abschnitt 4 Benutzungsschnittstelle.

Für die Ausführung wird auch auf das zusätzliche Modul `SCXML-Extension` zugegriffen, welches zwei Hilfsfunktionen für vereinfachte Aufrufe beinhaltet, genauere Informationen dazu sind in Unterunterabschnitt 3.3.3 zu finden. Dieser Aufruf ist auch im nachfolgenden Quellcode 10 zu sehen. Mittels der Funktion `xquery:update` wird ein neu erstellter XQuery-Ausdruck ausgeführt. Bei diesem Zusammenstellen wird auf die oben erwähnten Hilfsfunktionen zugegriffen, da diese den Aufruf der Fremdmodule bzw. eigenen Funktionsdeklarationen beinhaltet.

Funktion	Parameter	Beschreibung
sc:assign	$dataModels element()*, $location xs:string $expression xs:string? $type xs:string? $attribute xs:string? $nodelist node()* $dbName xs:string $collectionName xs:string $mbaName xs:string	Diese Funktion sorgt für die tatsächliche Ausführung des ausführbaren Inhalts des `assign`-Elements.
sc:getValue	$dataModels element()*, $location xs:string $expression xs:string? $type xs:string? $attribute xs:string? $nodelist node()* $id	Diese Funktion sorgt für die tatsächliche Ausführung des ausführbaren Inhalts des `getValue`-Elements. Der Wert wird dabei abhängig von der `EventId` gespeichert.
sc:l xsog	$dataModels element()*, $expression xs:string? $label xs:string? $id $dbName xs:string $collectionName xs:string $mbaName xs:string	Diese Funktion sorgt für die tatsächliche Ausführung des ausführbaren Inhalts des `log`-Elements. Der Wert wird dabei abhängig von der `EventId` gespeichert.

Tabelle 14: Modul SCXML: Funktionen zum Ausführen von ausführbarem Inhalt

Genauere Informationen sind in Unterunterabschnitt 3.3.3 angegeben. Da bei diesen Funktionen das jeweils aktuelle Datenmodell übergeben wird, können sie auch mit den aktuellen Werten arbeiten und mit diesen Werten das aktuelle MBA verändern.

```
1   xquery:update(
2         scx:importModules() ||
3         fn:string-join($declare) ||
4         $declareNodeList ||
5         scx:builtInFunctionDeclarations() ||
6         'let $locations := '
7         ...
```

Quellcode 10: Modul SCXML: Zusammenstellen des Funktionsheaders

Von den in Tabelle 15 beschriebenen Funktionen sind `sc:getInitalesStates` und `sc:findLCCA` hervorzuheben. Die Funktion `sc:findLCCA` erhält eine Liste von Zuständen und gibt den kleinsten Zustand zurück, der alle Zustände beinhaltet. Da in diesem Fall auch das SCXML-Modul als eine Art Zustand betrachtet wird, gibt sie das gesamte SCXML zurück, falls keine Übereinstimmung gefunden wird.

Die Funktion `sc:getInitalStates` berechnet den initialen Zustand für den übergebenen. Daher wird zuerst überprüft, ob ein initialer Zustand angegeben ist. Ist dies nicht der Fall, wird der erste Zustand, der im Dokument angegeben ist, zurückgegeben (*document order*).

Tabelle 16 beinhaltet jene Funktionen, die für die Berechnung der zu betretenden Zustände verwendet werden. Sie stellen fest, ob neben dem direkten Ziel der Transition auch noch Sub- bzw. Superzustände betreten werden müssen.

Wichtige Details der Implementierung sind dabei die beiden rekursiven Funktionen `sc:addAncestorStatesToEnter` bzw. `sc:addDescendantStatesToEnter`, die sich jeweils gegenseitig aufrufen. Dadurch wird garantiert, dass kein zu betretender Zustand ausgelassen wird.

Die Funktionsweise der Funktion `sc:addDescendantStatesToEnter` ist in Abbildung 7 skizziert. Die wichtigsten Parameter der Funktion sind `$states`, `$statesToEnter` und `$cont`. Der erste Parameter entspricht den Zuständen, die noch überprüft werden müssen und der zweite, `$states`, beinhaltet die Zustände die im Anschluss betreten werden müssen. Der Parameter `$cont` beinhaltet einen Funktionsaufruf. Beim ersten Aufruf beinhaltet dieser Funktionsaufruf nur das Einfügen der berechneten Dateien in eine Map um alle berechneten Werte an die aufrufende Funktion zurückgeben zu können. Die Funktion untersucht so lange die Zustände in der Liste `$states`, bis alle überprüft wurden.

Zu Beginn der Funktion wird überprüft, ob die Liste im Parameter `$states` noch Elemente beinhaltet. Ist dies nicht der Fall, ist die Berechnung abgeschlossen

Funktion	Parameter	Beschreibung
sc:getInitialStates	$state	Gibt für den übergebenen Zustand den initialen Zustand zurück.
sc:getHistoryStates	$state	Gibt für den aktuellen Zustand jene Zustände zurück, die in seiner History gespeichert sind.
sc:getChildStates	$state element()	Gibt alle direkten Zustände unterhalb des aktuellen Zustands zurück.
sc:findLCCA	$states element()*	Diese Funktion gibt den *Least Common Compound Ancestor*, also den kleinsten gemeinsamen Vorfahren von Zuständen, die in der Liste `$states` enthalten sind, zurück. Da alle Zustände eines Modells zumindest das SCXML-Element als gemeinsamen Vorfahren besitzen, ist garantiert, dass diese Funktion einen gültigen Wert zurückgibt.

Tabelle 15: Modul SCXML: Zustandsfunktionen

Funktion	Parameter	Beschreibung
sc:addDescendant-StatesToEnter	$state element()	Übergeordnete Funktion, die alle Nachfahren eines Zustands berechnet, die betreten werden müssen. Diese Funktion ruft anschließend nur die überladene Funktion auf.
sc:addDescendant-StatesToEnter	$states element()* $statesToEnter element()* $statesForDefaultEntry element()* $cont $historyContent	Diese Funktion bestimmt die Nachfahren eines Zustandes, die ebenfalls betreten werden müssen.
sc:addAncestor-StatesToEnter	$state element() $ancestor element()	Übergeordnete Funktion welche Vorfahren eines Zustands bestimmt, die ebenfalls betreten werden müssen. Sie beinhaltet lediglich den Aufruf der gleichnamigen, überladenen Funktion.
sc:addAncestor-StatesToEnter	$ancestor element() $statesToEnter element()* $statesForDefaultEntry element()* $cont $historyContent	Wenn die nachfolgende Funktion `foldAncestorStatesToEnter` fertig berechnet ist, wird in dieser Funktion das Zusammenfügen der Ergebnisse durchgeführt.
sc:foldAncestor-StatesToEnter	$states element()* $statesToEnter element()* $statesForDefaultEntry element()* $cont $historyContent	Diese rekursive Funktion führt die tatsächliche Berechnung der Zustände durch, die als Vorfahren des zu kontrollierenden Zustands ebenfalls betreten werden müssen.

Tabelle 16: Modul SCXML: Funktionen zur Berechnung von Nachfahren und Vorfahren

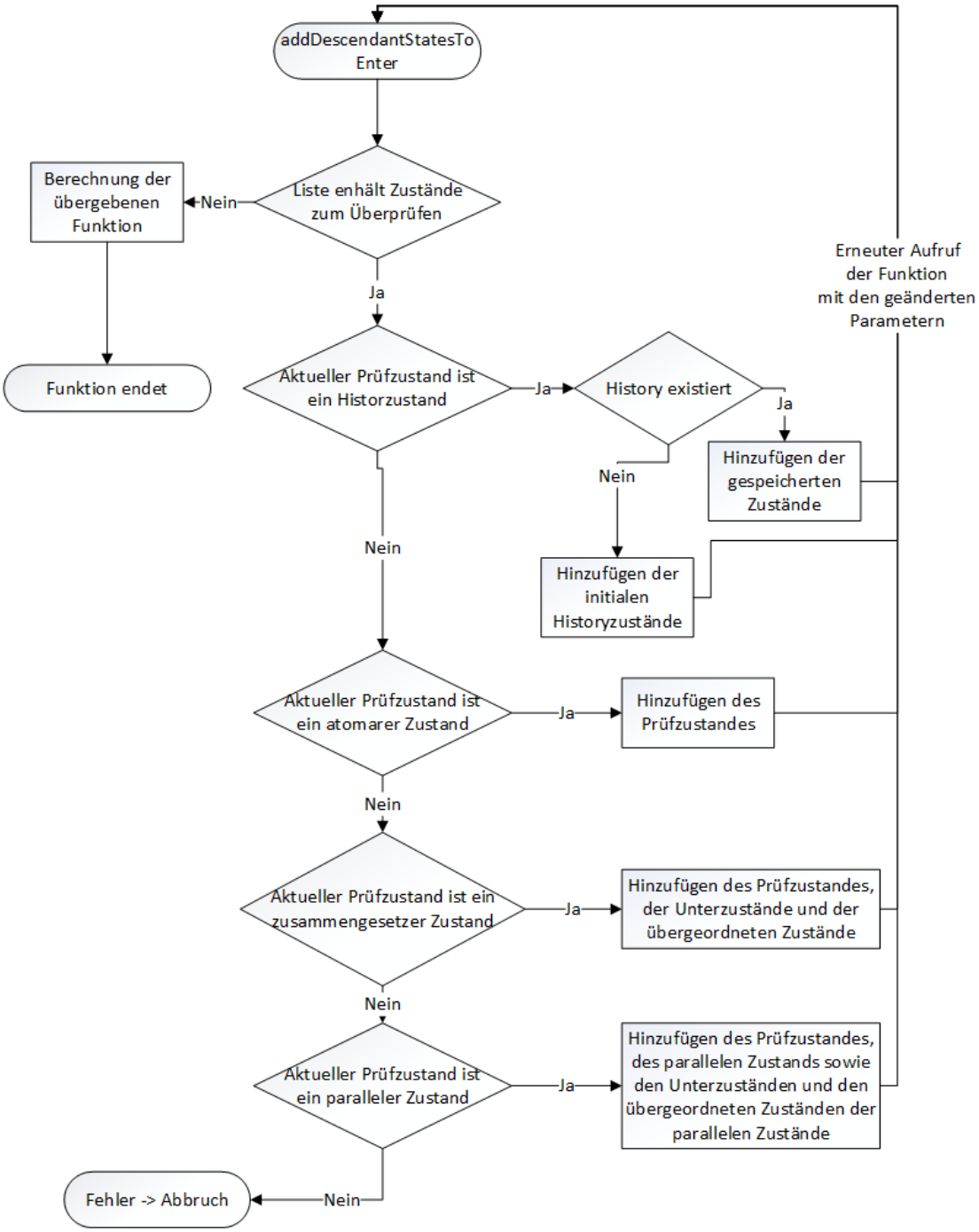

Abbildung 7: Aufbau der Funktion sc:addDescendantStatesToEnter

und die Funktion von `$cont` wird wie in Quellcode 11 aufgerufen. Anderenfalls wird im Anschluss der erste Zustand aus der Liste `$states` überprüft.

```
1 if (fn:empty($states)) then
2   $cont(
3     $statesToEnter, $statesForDefaultEntry, $historyContent)
```

Quellcode 11: Modul SCXML: Funktionsaufruf in sc:addDescendantStatesToEnter

Zuerst wird überprüft, ob der erste Zustand ein Historyzustand ist. Sollte dies der Fall sein wird weiters geprüft, ob bereits eine History zu diesem Zustand vorhanden ist. Trifft das nicht zu, wird die vom SCXML-Modell vorgegebene History herausgesucht.

Nun wird die Funktion `sc:addDescendantStatesToEnter` erneut aufgerufen, wobei sie nur für den ersten Zustand in der berechneten History ausgeführt wird. Über den Parameter `$cont` wird dabei anschließend der Aufruf der Funktion `sc:addAncestorStateToEnter` aufgerufen. Sie ist ebenfalls sehr ähnlich aufgebaut, daher erhält diese wiederum die Funktion `sc:addDescendantStatesToEnter` als Parameter. Bei diesem letzten Aufruf werden alle restlichen Zustände in mitgegeben. Dadurch wird sichergestellt, dass zuerst die Nachfolger (Subzustände) und die Vorfahren (Superzustände) des ersten Zustandes in der History, die betreten werden müssen, berechnet werden, und anschließend alle restlichen Zustände, die bereits in der Liste waren, sowie die restlichen Zustände der History.

Sollte der Interpreter feststellen, dass es zu einem Zustand bereits eine History gibt, werden die Funktionen wieder genauso aufgerufen wie zuvor beschrieben. Das bedeutet, dass mit dem ersten Zustand wieder `sc:addDescendantStatesToEnter` aufgerufen wird. Mithilfe von `$cont` wird dafür gesorgt, dass anschließend die Funktion `sc:addDescendantStatesToEnter` für den ersten Zustand aufgerufen wird und am Ende dieser Funktion `sc:addDescendantStatesToEnter` mit den restlichen Zuständen aufgerufen wird.

Sollte der zu überprüfende Zustand kein Historyzustand sein, wird als nächstes geprüft, ob er ein atomarer Zustand ist. Wenn dies zutrifft wird der aktuell Zustand zur Liste der zu betretenden Zustände hinzugefügt, und die Funktion mit den restlichen Zuständen der Variable `$states` aufgerufen.

Die nächste Prüfung stellt fest, ob es sich um einen zusammengesetzten Zustand handelt. Trifft dies zu, werden die initialen Zustände dieses Zustands bestimmt. Anschließend wird der aktuelle Zustand zur Liste der zu betretenden Zustände hinzugefügt. An die aufrufende Funktion wird wieder die Kombination der Funktionsaufrufe wie bei den Historyzuständen zurückgegeben. Die beiden ersten Aufrufe

beachten also nur den ersten Zustand der initialen Zustände, die Folgefunktion die übrigen.

Am Schluss der Funktion wird noch überprüft, ob der zu überprüfende Zustand ein `parallel`-Element ist. Sollte dies der Fall sein, werden die Kindzustände dieses Zustandes berechnet. Anschließend werden diejenigen Zustände weiterverarbeitet, welche sich noch nicht in der Liste der zu betretenden Zustände befinden.

Bei der Berechnung der in diesem Fall zu betretenden Zustände gibt es einen Unterschied zu den restlichen Aufrufen. Während zuvor jeweils nur ein direkter Subzustand beachtet werden musste, werden bei parallelen Zuständen stets alle Subzustände betreten. Daher ist es notwendig, die Funktion `sc:addDescendantStatesToEnter` für alle Subzustände aufzurufen und diese zur Liste der zu betretenden Zustände hinzuzufügen. Innerhalb dieser Funktion wird erneut die gleiche Funktion aufgerufen, welche die verbleibenden Zustände in der Liste der zu betretenden Zustände abarbeitet.

Funktion	Parameter	Beschreibung
sc:getTargetStates	$transition element()	Gibt die Zielzustände der übergebenen Transitionen zurück.
sc:getEffective-TargetStates	$transition element()	Gibt die tatsächlichen Zielzustände der Transitionen zurück. Diese Funktion berücksichtigt auch Historyzustände bzw. deren Voreinstellung.
sc:getSourceState	$transition	Gibt den Ursprungszustand einer Transition zurück.
sc:getSource-StateTrans	$transition	Gibt den eigentlichen Ausgangszustand einer in einer Variable gespeicherten Transition zurück.
sc:isInternal-Transition	$transition element()	Überprüft ob die übergebene Transition eine interne Transition ist.
sc:getTransition-Domain	$transition element()	Gibt den übergeordneten Zustand aller Zustände zurück, die durch eine Transition verlassen werden. Sollte kein solcher Zustand gefunden werden, wird das SCXML-Element zurückgegeben.
sc:getTransition-DomainExit	$transition element()	Gibt die Zustände zurück, die durch eine Transition verlassen werden. Diese Funktion ist für die Benutzungsschnittstelle von Bedeutung.
sc:getTransition-DomainTrans	$transition element()	Gibt den übergeordneten Zustand aller Zustände zurück, die durch eine Transition verlassen werden. Sollte kein solcher Zustand gefunden werden, wird das SCXML-Element zurückgegeben.

Tabelle 17: Modul SCXML: Funktionen zum Erhalt von Informationen über Transitionen

Funktion	Parameter	Beschreibung
sc:isCompoundState	$state element()	Überprüft, ob der übergebene Zustand ein zusammengesetzter Zustand ist.
sc:isAtomicState	$state element()	Überprüft, ob der übergebene Zustand ein unteilbarer Zustand ist.
sc:isParallelState	$state element()	Überprüft, ob der übergebene Zustand ein paralleler Zustand ist.
sc:isHistoryState	$state element()	Überprüft, ob der übergebene Zustand ein Historyzustand ist.
sc:isFinalState	$state element()	Überprüft, ob der übergebene Zustand ein Finalzustand ist.
sc:isDescendant	$state1 element() $state2 element()	Überprüft, ob `$state1` ein Nachfahre von `$state2` ist.
sc:getProper-Ancestors	$state element()	Gibt die direkten Vorfahren des übergebenen Zustands zurück.
sc:getProper-Ancestors	$state element() $upTo element()	Gibt die Vorfahren des übergebenen Zustands bis zu dem in der Variable `$upTo` angegebenen Zustand zurück.

Tabelle 18: Modul SCXML: Funktionen zum Erhalt von Informationen über Zustände

Die Funktionen in tab:SCXMLTransitions beschäftigen sich mit Transitionen und deren Herkunft bzw. Ziel. Die Funktion `sc:getTargetStates` berechnet die Zielzustände der übergebenen Transition. Im Gegensatz zur Funktion `sc:getEffectiveTargetStates` werden dabei die Historyzustände nicht beachtetet, daher ist diese Funktion nur bei der Berechnung der zu betretenden Zustände relevant, da in dieser die Historyzustände extra betrachtet werden. Die beiden Funktionen `sc:getSourceState` und `sc:getSourceStateTrans` geben den Ursprungsort einer Transition zurück. Die zuletzt genannte Funktion wird verwendet, wenn die Transition bereits in der Liste der aktuell aktiven Transitionen gespeichert ist.

Die Funktionen für genauere Feststellungen über Zustände sind in Tabelle 18 zusammengefasst. Während die erstgenannten Funktionen für die Typbestimmung

des Zustandes verwendet werden, liefern die nachfolgenden Funktionen Informationen über die Beziehungen von Zuständen zueinander.

Die Funktion `sc:addAncestorStatesToEnter` bildet die Gegenfunktion zu `sc:addDescendantStatesToEnter`. Auch ihr wird neben anderen Parametern die Liste der noch zu überprüfenden Zustände übergeben, sowie eine Funktion, die beim Beenden der Funktion durchgeführt wird.

Zuerst wird die Liste der hinzuzufügenden Zustände berechnet, dabei wird auf die Funktion `sc:getProperAncestor` aus Tabelle 18 zugegriffen. Ist die Liste der noch hinzuzufügenden Zustände leer, wird die übergebene Funktion aufgerufen und die Funktion ist beendet.

```
1 sc:foldAncestorStatesToEnter(
2     $states[position() > 1],
3     ($statesToEnter, $states[1]),
4     $statesForDefaultEntry,
5     $cont, $historyContent)
```

Quellcode 12: Modul SCXML: Rekursiver Aufruf der Funktion sc:foldAncestorStatesToEnter

Ansonsten wird in der Funktion `sc:foldAncestorStatesToEnter` erneut der erste Zustand der noch hinzuzufügenden Zustände überprüft. Sollte dies ein paralleler Zustand sein, wird erneut die Funktion `sc:addDescendantStatesToEnter` mit den Zuständen aufgerufen, die sich noch nicht in der Liste der zu betretenden Zustände befinden. Wenn kein paralleler Zustand vorliegt, wird der Zustand zur Liste der zur betretenden Zustände hinzugefügt und die Funktion `sc:foldAncestorStatesToEnter` mit den restlichen Zuständen der Liste erneut aufgerufen, wie in Quellcode 12 dargestellt.

Funktion	Parameter	Beschreibung
mba:getExternal-EventQueue	$mba element()	Gibt die Liste der eingereihten externen Events zurück.
mba:enqueue-ExternalEvent	$mba element() $event element()	Fügt das Event zur Liste der externen Events hinzu.
mba:dequeue-ExternalEvent	$mba element()	Entfernt das aktuelle Event von der Liste der externen Events und löscht es.
mba:loadNext-ExternalEvent	$mba element()	Holt das aktuell gültige Event von der Liste der externen Events und fügt dieses als aktuell aktives Event ein. Anschließend wird es aus der Liste der externen Events entfernt.

Tabelle 19: Modul MBA: Funktionen für externe Events

3.3.2 Verwaltung von Multilevel Business Artifacts

Das MBA Modul enthält neben Funktionen für die Verwaltung von MBAs Zugriffsfunktionen für die SCXML-Variablen. Die in Tabelle 19 und Tabelle 20 angegebenen Funktionen dienen zur Verwaltung der internen und externen Events.

Wie in Abschnitt 3 beschrieben, werden interne und externe Events voneinander getrennt gespeichert, da interne immer vor externen Events abgearbeitet werden. Die Speicherung der Events erfolgt jedoch gleich, daher ist auch die Funktionsweise der Funktionen in Tabelle 19 die gleiche wie die der Funktionen in Tabelle 20.

Die Funktionen `mba:getExternalEventQueue` sowie `mba:getInternalEventQueue` werden als Zugriffsfunktionen zum Speicherort der jeweiligen Queues verwendet. Diese Funktionen ermöglichen es außerdem festzustellen, ob aktuell interne bzw. externe Events in der Liste vorhanden sind.

Bei der Abarbeitung der Events gilt das Warteschlangenprinzip, d.h. dass die zuerst angekommen Events vor später ankommenden verarbeitet werden. Um das älteste externe Event als aktuelles Event zu speichern, löscht die Funktion `mba:loadNextExternalEvent` das aktuell aktive Event. Anschließend entfernt die Funktion `mba:dequeueExternalEvent` das älteste Event aus der Liste und fügt es als aktives Event ein.

Um das aktuelle Event zu löschen wird die in Tabelle 21 beschriebenen Funktion `mba:removeCurrentEvent` verwendet. Die Funktion `mba:getCurrentEvent`

Funktion	Parameter	Beschreibung
mba:getInternal-EventQueue	$mba element()	Gibt die Liste der eingereihten internen Events zurück.
mba:dequeue-InternalEvent	$mba element()	Entfernt das aktuelle Event von der Liste der internen Events und löscht es.
mba:enqueue-InternalEvent	$mba element() $event element()	Fügt das Event zur Liste der internen Events hinzu.
mba:loadNext-InternalEvent	$mba element()	Holt das aktuell gültige Event von der Liste der internen Events und fügt dieses als aktuell aktives Event ein. Anschließend wird es aus der Liste der internen Events entfernt.

Tabelle 20: Modul MBA: Funktionen für interne Events

Funktion	Parameter	Beschreibung
mba:getCurrentEvent	$mba element()	Gibt das aktuell aktive Event zurück.
mba:removeCurrentEvent	$mba element()	Löscht das aktuell aktive Event.

Tabelle 21: Modul MBA: Funktionen für das aktuelle Event

bestimmt die zu feuerenden Transitionen. Sie liefert das aktuelle Event an die aufrufende Funktion zurück.

In der Tabelle 22 sind die Funktionen zur Verwaltung der aktuell aktiven Zustände bzw. der aktuellen Konfiguration zusammengefasst. Darunter fallen Funktionen zum Hinzufügen von Zuständen zur Konfiguration (`mba:addCurrentStates` und `mba:addCurrentState`) sowie Funktionen zum Entfernen eines aktiven Zustandes. Die Speicherung ist in Quellcode 13 dargestellt. Um aktive Zustände zu speichern wird nicht das gesamte Zustandselement, sondern die eindeutige Bezeichnung des Elements der Liste hinzugefügt. Dadurch wird der benötigte Speicherplatz erheblich verringert. Da für die Verarbeitung jedoch der gesamte Zustand erforderlich ist, nimmt die Funktion `mba:getConfiguration` diese Liste von der Funktion `mba:getCurrentStatus` und bestimmt die Zustandselemente zur weiteren Verwendung aus dieser Liste.

Funktion	Parameter	Beschreibung
mba:isInState	$mba element() $stateId xs:string	Überprüft, ob sich das SCXML-Modell im übergebenen Zustand befindet.
mba:getCurrentStatus	$mba element()	Gibt eine Liste zurück, welche die eindeutigen Ids der Zustände beinhaltet, die im aktuellen MBA und dem darin beinhaltendem SCXML-Modell aktuell gültig sind.
mba:getConfiguration	$mba element()	Gibt die kompletten Zustände zurück, die sich in der Liste der aktuellen Zustände befinden.
mba:addCurrentStates	$mba element() $states element()*	Fügt die übergebenen Zustände zur Liste der aktuell aktiven Zustände hinzu.
mba:addCurrentState	$mba element() $state element()	Fügt den übergebenen Zustand zur Liste der aktuell aktiven Zustände hinzu.
mba:removeCurrentStates	$mba element() $states element()*	Entfernt die übergebenen Zustände von der Liste der aktuell aktiven Zustände, falls sie darin enthalten sind.
mba:removeCurrentState	$mba element() $state element()	Entfernt den übergebenen Zustand aus der Liste der aktuell aktiven Zustände, falls dieser darin enthalten ist.

Tabelle 22: Modul MBA: Funktionen für systemspezifische Variablen 1

```
1 declare updating function mba:addCurrentState(
2     $mba element(), $state element()){
3     let $currentStatus := mba:getCurrentStatus($mba)
4     let $newState :=
5         <state xmlns="" ref="{$state/@id}"/>
6     return insert node $newState into $currentStatus
7 };
```

Quellcode 13: Modul MBA: Funktionsweise der Funktion mba:addCurrentState

Tabelle 23 beinhaltet jene Funktionen, die sich mit den aufzurufenden Zuständen beschäftigen. Hierzu zählt insbesondere die Liste der noch auszuführenden Zustände sowie die Zugriffsfunktionen zum Setzen der aufrufenden bzw. aufgerufenen Interpreter.

Die Variablen , in denen Daten der aufzurufenden MBAs gespeichert sind, werden mit den Funktionen aus Tabelle 24 bearbeitet. Neben den üblichen Zugriffsfunktionen zum Erhalten, Hinzufügen und Löschen von einzelnen Elementen aus dieser Liste gibt es auch die Funktion `mba:deleteStatesToInvokeQueue`, welche die Liste vollständig leert. Sie wird aufgerufen, nachdem alle aufzurufenden `invoke`-Elemente in der RESTXQ-Funktion `invokeStates` aufgerufen wurden.

In Tabelle 25 werden die Funktionen zur Bearbeitung von MBAs kurz erklärt. Diese Funktionen sind für die in Unterabschnitt 3.5 beschriebene Process Enviroment wichtig, da dadurch veränderte MBAs erneut aufgerufen bzw. unveränderte MBAs nicht bearbeitet werden.

In der Datenbank werden für MBAs einige XML-Elemente angelegt. Die ersten beiden Elemente sind jeweils Schemadefinitionen für die nachfolgenden Elemente. Das anschließende Element ist die `collections`-Datei. Sie enthält alle Collections der jeweiligen Datenbank. In diesen Dateien wird auch der Status der MBAs definiert. Soll beispielsweise ein MBA als geändert markiert werden, wird das in dieser Datei vermerkt indem unterhalb des Elements `updated` eine Referenz des MBAs eingefügt wird.

In Quellcode 14 ist die Speicherung der Collections in der Datenbank dargestellt. Für jede Collection wird ein XML-Tag eingefügt, der den Status eines jeden MBAs der Collection beinhaltet. Im Beispiel von Quellcode 14 sind drei MBAs der Collection `JohannesKeplerUniversity` als geändert markiert.

Die ebenfalls in Tabelle 25 beschriebene Funktion `mba:marksAsUpdated` funktioniert analog zu `mba:markAsNew`. Der einzige Unterschied ist, dass ein Eintrag beim Element `new` hinzugefügt wird.

Die in Tabelle 26 beschriebenen Funktionen sind für die Benutzung des Interpreters erforderlich. Die Funktion `mba:init` fügt dem SCXML-Modell einige Datenelemente hinzu, die vom Interpreter benötigt werden bzw. vom Standard

Funktion	Parameter	Beschreibung
mba:getChildInvokeQueue	$mba element()	Gibt die Liste mit den aufzurufenden Zuständen eines MBAs sowie deren Daten zurück.
mba:getParentInvoke	$mba element()	Gibt die Daten des aufrufenden MBAs zurück.
mba:setParentInvoke	$mba element() $mbaparent element()	Fügt dem aktuellen MBA eine Referenz des aufrufenden MBAs hinzu.
mba:updateChildInvoke	$mba element() $state element() $dbName xs:string $collectionName as xs:string $mbaName xs:string $id xs:string	Fügt dem aktuellen MBA eine Referenz zum aufgerufenen MBA hinzu. Hierbei ist es wichtig, dass diese Referenz auch den aufrufenden Zustand beinhaltet.

Tabelle 23: Modul MBA: Funktionen zur Verknüpfung von aufgerufenen Interpretern

Funktion	Parameter	Beschreibung
mba:getStatesToInvokeQueue	$mba element()	Gibt die Liste der Zustände zurück, die noch aufgerufen werden müssen.
mba:deleteStatesToInvokeQueue	$mba element()	Leert die Liste der Zustände, welche noch aufgerufen werden müssen, zur Gänze.
mba:addstatesToInvoke	$mba element() $entrySet element()*	Fügt die übergebenen Zustände zur Liste der noch aufzurufenden Zustände hinzu.
mba:removestatesToInvoke	$mba element() $exitSet element()*	Entfernt die in der Variable `$exitSet` enthaltenen Zustände von der Liste der noch aufzurufenden Zustände.

Tabelle 24: Modul MBA: Funktionen zur Speicherung von `invoke`-Elementen

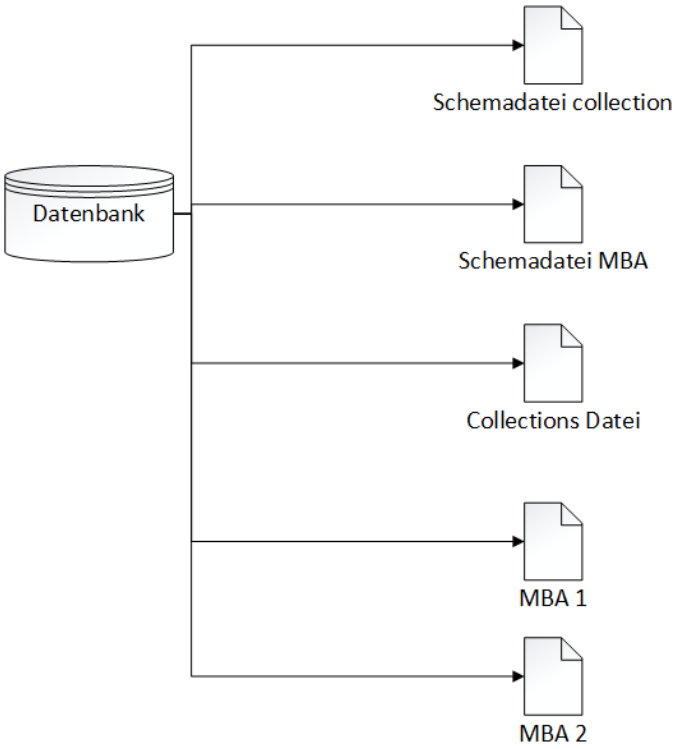

Abbildung 8: Speicherung des MBA in der Datenbank

Funktion	Parameter	Beschreibung
mba:markAsUpdated	$mba element()	Markiert das aktuelle MBA als geändert, damit es von der in Unterabschnitt 3.5 genannten Process Environment richtig aufgerufen werden kann.
mba:markAsNew	$mba element()	Markiert das aktuelle MBA als neu, damit es von der Process Environment richtig aufgerufen werden kann.
mba:removeFromUpdateLog	$mba element()	Entfernt das MBA von der Liste der geänderten MBAs. Dies verhindert einen falschen Aufruf des MBAs in der Process Environment.
mba:removeFromInsertLog	$mba element()	Entfernt das MBA von der Liste der neuen MBAs. Dies verhindert einen falschen Aufruf des MBAs in der Process Environment.

Tabelle 25: Modul MBA: Funktionen zur Markierung von MBAs

```
1 <collections xmlns="http://www.dke.jku.at/MBA"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://www.dke.jku.at/MBA
4   xsd/collections.xsd">
5   <collection name="JohannesKeplerUniversity"
6     file="collections/JohannesKeplerUniversity.xml"
7     hierarchy="simple">
8     <new/>
9     <updated>
10      <mba:mba xmlns:mba="http://www.dke.jku.at/MBA"
11        ref="JohannesKeplerUniversity"/>
12      <mba:mba xmlns:mba="http://www.dke.jku.at/MBA"
13        ref="SocialAndEconomicSciences"/>
14      <mba:mba xmlns:mba="http://www.dke.jku.at/MBA"
15        ref="InformationSystems"/>
16    </updated>
17  </collection>
18 </collections>
```

Quellcode 14: Modul MBA: Speicherung der Collection in der Datenbank

Funktion	Parameter	Beschreibung
mba:init	$mba element()	Initalisiert das MBA mit allen Variablen, die vom Standard [Barnett et al., 2015] bzw. Interpreter benötigt werden um einen korrekten Ablauf zu gewährleisten.

Tabelle 26: Modul MBA: Funktion zum Initialisieren des MBAs

bezüglich SCXML-Interpreter vorgeschriebene sind. Dazu zählen folgende Datenelemente:

- _event
  - Dieses Datenelement beinhaltet das aktuell aktive Event sowie alle zugehörigen Datenelemente.
- _sessionid
  - Beinhaltet die eindeutige SessionId, die das MBA indentifiziert. Diese setzt sich aus den Namen der Datenbank (`mba:dbName`), der Collection (`collectionName`) und des MBAs (`mbaName`) zusammen.
- _name

  - Dieses Element beinhaltet den Namen des SCXML-Modells.

- _ioprocessors
  - Diese Variable beinhaltet all jene Event I/O Prozessoren, die von der Umsetzung unterstützt werden. Die vorliegende Implementierung erlaubt die Verwendung des SCXML-Eventprozessors, der mit folgender URL angegeben wird:
    'http://www.w3.org/TR/scxml/#SCXMLEventProcessor'.

- _x
  Dieses Element ist das Wurzelelement für alle plattformspezifischen Systemvariablen. Plattformspezifische Variablen dürfen nur unterhalb dieses Wurzelknotens vorkommen.
  - db
    * Der Name der Datenbank, in dem das MBA gespeichert ist.
  - collection
    * Der Name der Collection, zu der das MBA gehört.
  - currentStatus
    * Die aktuell aktiven Zustände.
  - externalEventQueue
    * Die aktuelle Liste der externen Events.
  - internalEventQueue
    * Die aktuelle Liste der internen Events.
  - isRunning
    * Beschreibt, ob der Interpreter noch aktiv ist.
  - parentInvoke
    * Die aufrufenden SCXML-Interpreter.
  - childInvoke
    * Die aufgerufenen SCXML-Interpreter.
  - currentEntrySet
    * Die Liste der aktuell zu betretenden Zustände.
  - currentExitSet
    * Die Liste der aktuell zu verlassenden Zustände.

- currentTransitions
  - Die Liste der aktuell aktiven Transitionen.
- statesToInvoke
  - Die Liste der Zustände, die noch auszuführen sind.
- historyStates
  - Die Speicherung der History-Zustände.
- response
  - Die Speicherung der Benutzerrückgabewerte.
- log
  - Das Logging der aktuellen Events.

Tabelle 27 beinhaltet einige Zugriffsfunktionen für Variablen. Dabei handelt es sich um Variablen, welche der Interpreter benötigt, um einen reibungslosen Ablauf garantieren zu können. Die Funktionspaare `mba:getCurrentEntryQueue` und `mba:getCurrentExitQueue` sowie `mba:getCurrentExitSet` und `mba:getCurrentEntrySet` sind ähnlich aufgebaut. Die beiden Queue-Funktionen werden sowohl zum Speichern in der Queue, als auch zum Überprüfen, ob die Variablen aktuell Zustände enthalten, verwendet. Da bei der Speicherung immer nur die Id eines Zustandes gespeichert wird, wandeln die Funktionen zum Erhalten der Sets den Inhalt der Queue-Funktionen wieder in die originalen Zustände um. Um die zu betretenden oder zu verlassenden Zustände in den Variablen zu speichern, überschreiben die Funktionen `sc:updateCurrentExitSet` und `sc:updateCurrentEntryset` eventuell existierende Listen. Zum Evaluieren bzw. für Datenbankänderungen im Allgemeinen wird das aktuelle Datenmodell benötigt, das mithilfe der in Tabelle 28 beschriebenen Funktionen an die aufrufende Funktion übergeben wird.

Die beiden Funktionen `mba:selectDataModels` und `mba:selectAllDataModels` unterscheiden sich bei der Angabe der Datenmodelle in zwei Punkten. Während erstere alle Datenelemente der aktiven Zustände und das globale Datenelement an die aufrufende Funktion zurückliefert, werden bei der zuletzt genannten Funktion alle Datenelemente des MBA zurückgegeben, d.h. neben den globalen Datenelementen auch alle in den einzelnen Zuständen angegebene Datenelemente.

Funktion	Parameter	Beschreibung
mba:getRunning	$mba element()	Gibt den Inhalt der Variable Running zurück, die angibt, ob sich das SCXML-Modell in einem finalen Zustand befindet.
mba:updateRunning	$mba element() $value xs:boolean	Verändert den Wert der Variable Running mit dem im Parameter `$value` angebenden Wert.
mba:getCurrent-EntryQueue	$mba element()	Gibt eine Liste von Bezeichnungen der zu betretenden Zuständen zurück.
mba:getCurrent-EntrySet	$mba element()	Gibt die Liste der zu betretenden Zustände zurück.
mba:getCurrent-ExitSet	$mba element()	Gibt die Liste der zu verlassenden Zustände zurück.
mba:getCurrent-ExitQueue	$mba element()	Gibt die Namen der zu verlassenden Zustände zurück.
mba:getCurrent-TransitionsQueue	$mba element()	Gibt die gespeicherte Liste der aktuell aktiven Transitionen zurück.
mba:updatecurrent-EntrySet	$mba element() $entrySet element()*	Überschreibt die aktuelle Liste der zu betretenden Zustände mit denen, die im Parameter `$entrySet` angegeben sind.
mba:updateCurrent-ExitSet	$mba element() $exitSet element()*	Überschreibt die aktuelle Liste der zu verlassenden Zustände mit denen, die im Paramter `$exitSet` angebenen sind.
mba:updatecurrent-Transitions	$mba element() $transitions	Überschreibt die aktuell gültigen Transitionen mit denen, die im Paramter `$transitions` angegeben sind.

Tabelle 27: Modul MBA: Funktionen für systemspezifische Variablen 2

Funktion	Parameter	Beschreibung
mba:selectDataModels	$configuration element()*	Gibt alle obersten Datenelemente des aktuellen SCXML-Modelles zurück.
mba:selectAllDataModels	$mba element()*	Gibt alle Datenelemente, die im SCXML-Modell enthalten sind, zurück.

Tabelle 28: Modul MBA: Funktionen zum Erhalten von Datenmodellen

Die in Tabelle 29 aufgezählten Funktionen beinhalten zwei der meistgenutzten. Die erste dieser beiden Funktionen, `mba:getMBA`, holt ein eindeutig identifiziertes MBA aus der Datenbank. Die zweite, `mba:getSCXML`, liefert das SCXML-Modell eines MBAs zurück. Darüber hinaus sind in dieser Tabelle weitere Hilfsfunktionen beschrieben, unter anderem zum Erhalten des Datenbanknamens eines MBAs.

Funktion	Parameter	Beschreibung
mba:getMBA	$dbName xs:string $collectionName xs:string $mbaName xs:string	Öffnet die in der Variable `$dbName` genannte Datenbank um die in `$collectionName` gennannte Collection zu öffnen. Anschließend wird das MBA mit dem Namen `$mbaName` zurückgeben. Wie in Unterabschnitt 3.2 angebenden, wird ein MBA eindeutig von diesen drei Parametern identifiziert.
mba:getSCXML	$mba element()	Gibt das SCXML-Element des obersten Levels eines MBAs zurück.
mba:getHistory	$mba element()	Gibt die gesamte gespeicherte History der Zustände zurück.
mba:getDatabaseName	$mba element()	Gibt den Datenbanknamen des übergebenen MBAs zurück.
mba:getCollectionName	$mba element()	Gibt den CollectionName des übergebenen MBAs zurück.
mba:getCounter	$mba element()	Gibt den aktuellen Wert der Zählervariable zurück, die zum Speichern der von Benutzerinnen und Benutzern hinzugefügten Events verwendet wird.

Tabelle 29: Modul MBA: Hilfsfunktionen

3.3.3 Sonstige Module

Das zusätzliche Modul `SCXML-Extension` wird dazu genutzt um das Modul SCXML zu erweitern. Dieses Modul beinhaltet Hilfsfunktionen welche eine vereinfachte und verkürzte Angabe von Funktionen in Bedingungen ermöglicht.

```
1 let $inFunction :=
2     'let $_in := function($stateId) { ' ||
3     'fn:exists($_x/currentStatus/state[@ref=$stateId])' ||  '} '
```

Quellcode 15: Modul SCXML-Extension: Erweiterung inState

Ein Beispiel ist die in Quellcode 15 angegebene Funktion `$_in`. Diese Funktion erlaubt das Abkürzen von der Bedingung innerhalb eines SCXML-Elements, dass sich das SCXML-Modell in einen bestimmten Zustand befinden muss. Die Angabe in der Transitionsbedingung wie in Quellcode 16 sorgt dafür, dass die Transition in Zeile 2 nur dann aktiv wird, wenn sich der Interpreter aktuell im Zustand `s1` befindet.

```
1 <state id="s0">
2      <transition cond="$_in(s1)" target="pass" />
3      <transition event="timeout" target="fail" />
4  </state>
```

Quellcode 16: Modul SCXML-Extension: Verwendung von inState

Bei der Überprüfung von Bedingungen werden die Funktionen dieses Moduls verwendet. Dadurch ist es möglich, weitere Abkürzungen bei Bedingungen auch nachträglich hinzuzufügen ohne das SCXML-Modul abzuändern. Des Weiteren erlaubt das Modul, bereits implementierte SCXML-Funktionen hinzuzufügen, da zusätzliche Module über Deklarationen importiert werden können.

Für die Ausführung von speziellen ausführbaren Inhalten wird das Modul Synchronisation verwendet. Es ermöglicht die Kommunikation zwischen unterschiedlichen Levels von MBAs. Darüber hinaus erlaubt es das Erstellen von neuen MBAs bzw. das Senden von Events an alle Vorfahren eines MBAs. In dieser Arbeit wird das Synchronisations-Modul nur für das Ausführen von ausführbaren Inhalten verwendet.

3.4 Logging

Um eine Analyse des Ablaufs eines Geschäftsprozesses zu ermöglichen, erzeugt der Interpreter einen Log. Für das Loggen des Ablaufes verwendet der Interpreter den XES (eXtensible Event Stream) Standard [Günther und Verbeek, 2014]

Die Umsetzung des Loggings erfolgt in der RESTXQ-Schnittstelle sowie im Modul SCXML. In der RESTXQ-Schnittstelle wird dabei nur die tatsächliche Funktion im Modul SCXML aufgerufen.

Die Funktion `doLogging` wird dabei vor dem Aufruf der Funktion `enterStates` eingefügt. Innerhalb der Funktion wird unterschieden, ob es sich um den initialen Aufruf des Interpreters und damit dem Transitionstyp (`transType`) `init` handelt oder ob es sich bereits um einen erneuten Aufruf handelt. Je nachdem wird anschließend die Funktion aus Tabelle 30 aufgerufen.

Die Eventlogs werden wie die restlichen plattformspezifischen Variablen im Datenmodell unterhalb des Elements `_x` gespeichert. Dabei wird sowohl ein `log`-Element als auch ein `trace`-Element eingefügt, welche anschließend die einzelnen Event-Ereignisse beinhalten. Quellcode 10 beinhaltetet einen beispielhaften Log.

Der erste Eventlog zeigt, dass der zuerst aktive Zustand `Developing` war, anschließend wurde in den Zustand `Active` gewechselt. Dafür wurde das Event `done` verwendet. Im zweiten Log wird nun vom Zustand `Active` in den Zustand `Discontinued` gewechselt. Dabei waren sowohl die angegebene Bedingung mit dem Key `sc:cond` als auch das angegebene Event mit dem Namen `discontinue` notwendig. Um weitere Analysen vornehmen zu können, wird dabei jeweils der Durchführungszeitpunkt gespeichert. Weitere Informationen zu den Logs finden sich im verwendeten XES-Standard [Günther und Verbeek, 2014].

Funktion	Parameter	Beschreibung
sc:doLogging	$mba element() $transition element() $transType xs:string)	Führt das Loggen von Transitionen durch.

Tabelle 30: Funktion zum Loggen

```
<xes:log xmlns:xes="www.xes-standard.org/">
    <xes:trace>
        <xes:event>
            <xes:date key="time:timestamp"
            value="2016-08-02T22:15:23.735+02:00"/>
            <xes:string key="sc:state" value="Developing"/>
            <xes:string key="concept:name" value="done"/>
            <xes:string key="sc:event" value="done"/>
            <xes:string key="sc:target" value="Active"/>
        </xes:event>
        <xes:event>
            <xes:date key="time:timestamp"
            value="2016-08-02T22:15:27.961+02:00"/>
            <xes:string key="sc:state" value="Active"/>
            <xes:string key="concept:name" value="discontinue"/>
            <xes:string key="sc:event" value="discontinue"/>
            <xes:string key="sc:target" value="Discontinued"/>
            <xes:string key="sc:cond"
            value="$_everyDescendantAtLevelIsInState(
            'courseInstance',   'Finished') and
            $_everyDescendantAtLevelIsInState(
            'courseInstance',  'Graded')"/>
        </xes:event>
    </xes:trace>
</xes:log>
```

Quellcode 17: Modul SCXML: Logging

3.5 Process Environment

Die Process Environment sorgt für den allgemeinen Ablauf der Automatisierung von MBAs und dient als Umgebung für die in Abschnitt 3 beschriebene Process Engine.

Innerhalb der Process Environment stehen zwei Grundfunktionen zur Verfügung. Die wichtigste der beiden ist das Aufrufen des Interpreters. Die zweite Hauptfunktion ist der Aufruf der Process Engine um ein MBA zu initialisieren. Damit die Process Engine Änderungen von dem in Abschnitt 4 beschriebenen User Interface sowie von externen Anwendungen aufgreifen kann, ruft die Process Environment den Interpreter in regelmäßigen Abständen auf.

Dazu wird der Quartz Scheduler verwendet, ein Framework welches es erlaubt vordefinierte Jobs in bestimmten zeitlichen Abständen auszuführen. Die Process Environment verwendet den Scheduler um den Job, der die Abarbeitung der MBAs beinhaltet, in regelmäßigen, definierten Zeitabständen aufzurufen. Der Job zur Abarbeitung von MBAs kann beispielsweise alle 5-10 Sekunden aufgerufen werden. Die Konfiguration der Process Environment erfolgt durch die drei Dateien

`query.properties`, `xqj.properties` sowie `environment.properties`. Jede dieser Dateien beinhaltet dabei einen wichtigen Aspekt der Process Environment. Sie sind nachfolgend aufgelistet und beschrieben.

- `quartz.properties`
  - Diese Datei beinhaltet die wichtigsten Eigenschaften des Quartz Schedulers. Dazu zählt neben seinem Namen auch die Anzahl der gleichzeitigen Threads. Die Zeiteinstelllungen des Schedulers werden nicht in dieser Datei konfiguriert, sondern in `environment.properties`.
- `xqj.properties`
  - Diese Datei beinhaltet die notwendigen Einstellungen für die Verbindung der Process Environment mit der Datenbank. In dieser Arbeit wird eine BaseX-Datenbank verwendet. Neben dem `Username` und dem `Passwort` sind daher auch der Name sowie Port des Servers angegeben.
- `environment.properties`
  - Diese Datei beinhaltet die wichtigsten Konfigurationen der Process Environment. In diesen Eigenschaften wird die Häufigkeit der Aufrufe der Process Engine festgelegt. Noch entscheidender sind jedoch die Einstellungen `Database` und `Collections`. Diese Werte stehen für den Namen der Datenbank sowie für die der Collections, die von der Process Environment kontrolliert werden sollen. Dadurch werden MBAs, die von der Process Environment aufgegriffen werden, eingeschränkt.

Sobald der Scheduler einen Job startet, verbindet sich die Process Environment mit der Datenbank. Das Abarbeiten der MBAs passiert dabei in einer festgelegten Reihenfolge. Der Ablauf des Jobs ist in Abbildung 9 dargestellt.

1. Im ersten Schritt werden all jene MBAs herausgesucht, welche aufgrund eines `invoke`-Elements erzeugt wurden. Das Erkennen dieser MBAs ist aufgrund der Namenskonvention sehr einfach möglich. Von `invoke`-Elementen erzeugte MBAs werden in automatisch erzeugten Datenbanken gespeichert. Der generierte Name der Datenbank besteht aus der Zeichenkette `invoke` sowie einer generierten Nummer. Diese Liste von MBAs wird anschließend in den beiden nachfolgenden Schritten bearbeitet.
2. Zuerst werden die als neu markierten MBAs aus der Liste bearbeitet. Neue MBAs besitzen noch keine Systemvariablen, die von der Process Engine

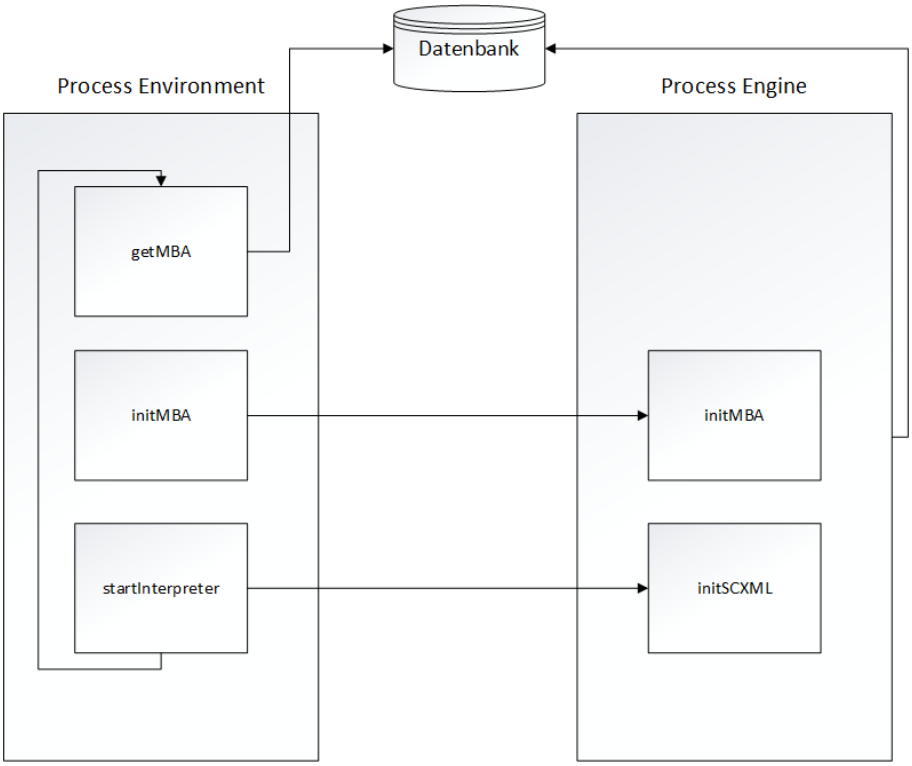

Abbildung 9: Verknüpfung zwischen Process Environment und Process Engine

zum Interpretieren des im MBA enthaltenen SCXML-Modells benötigt werden. Wie in Abbildung 9 dargestellt ruft die Process Environment nun die RESTXQ-Schnittstelle auf, dabei übergibt sie die Daten des aktuellen MBAs als Parameter. Im Anschluss an die Initialisierung markiert die RESTXQ-Schnittstelle das MBA als geändert, was dafür sorgt, dass diese MBAs im nächsten Schritt erneut aufgegriffen werden.

3. Im nächsten Schritt werden nun alle als geändert markierten MBAs der Liste bearbeitet. Die Process Environment ruft abermals die Process Engine auf, welche nun die übergebenen MBAs bearbeitet. Der RESTXQ-Aufruf startet nun die Interpretation des SCXML-Elements des MBAs.

4. Im vierten Schritt werden nun die eigentlichen zu bearbeitenden MBAs herausgesucht. Wie bereits bei der Konfiguration beschrieben ist dies von den in der Datei `environment.properties` angegebenen Namen der Datenbank bzw. der Collection abhängig. MBAs, die aufgrund dieser Einschränkungen bearbeitet werden sollen, werden nun ebenfalls in einer Liste gespeichert. Die nachfolgenden beiden Schritte sind genauso aufgebaut wie das Abarbeiten der Liste der MBAs aus `invoke`-Elementen.

5. Daher werden in Schritt 5 erneut diejenigen MBAS initialisiert, welche als neu markiert wurden.

6. Im letzten Schritt werden als geändert markierte MBAs aus der Liste bearbeitet. Die MBAs werden jeweils vom Interpreter aufgerufen und abgearbeitet.

Die Verbindung zwischen Process Environment und Process Engine funktioniert über REST-URLs sowie XQuery-Statements. XQuery-Statements werden nur zum Abrufen von Daten aus der Datenbank benutzt, Modifikationen von Daten werden über Aufrufe der RESTXQ-Schnittstelle initiiert. In den REST-URLs werden dabei die Daten eines MBAs (`name`, `collectionName`, `dbName`) an die RESTXQ-Schnittstelle übergeben. Die Process Environment bietet noch weitere Funktionen an, wie zum Beispiel zum Einfügen eines neuen MBAs aus einer Datei sowie zum Ausführen von XQuery-Statements jeglicher Art. Diese Funktionen werden für Tests genutzt um einfaches Testen, auch in Verbindung mit dem JUnit Framework, der Process Engine zu ermöglichen. Im Appendix sind die aus dem Standard durchgeführten Tests aufgezählt sowie die Ergebnisse ihrer Durchführung beschrieben.

3.6 Einschränkungen

Bei der Umsetzung des Interpreters wurden einige Elemente aus den nachstehenden Gründen nicht umgesetzt und werden daher von diesem Interpreter nicht unterstützt.

- ECMAScript
  - Aufgrund der zu Beginn getroffenen Entscheidung, wie in Unterabschnitt 3.1 genauer erklärt, war es wichtig das XPath-Datenmodell zu unterstützen. Das ebenfalls vom Standard unterstützte ECMAScript wurde aufgrund der Verwendung der `BaseX`-Datenbank nicht unterstützt.
- delay
  - Die XQuery-Engine beinhaltet keine eigenen Zeitfunktionen. Die FunctX-XQuery-Funktionen [Datypic, 2014] beinhalten zwar einige Funktionen, die den Umgang mit Zeitwerten erlauben, jedoch ist es beispielsweise nicht möglich den Interpreter für fünf Sekunden zu stoppen und anschließend wieder zu starten, ohne dass dieser in der Zwischenzeit die Datenbank blockieren würde. Daher wurde auf eine Umsetzung von ausführbaren Inhalten, die zeitliche Voraussetzungen benötigen, verzichtet.
- cancel
  - Das `cancel`-Event wird dazu verwendet ein etwaiges `delay`-Element abzubrechen. Da wie oben bereits genannt auf die Implementierung von `delay`-Elementen verzichtet wurde, wurde das Gegenstück ebenfalls nicht implementiert.
- script
  - Das `script`-Element erlaubt das Ausführen von externen Scripts. Da sich das XPath-Datenmodell allerdings nicht für Scripting eignet, wurde dieses Element nicht umgesetzt. [Barnett et al., 2014].
- foreach
  - Da das `foreach`-Element erlaubt, dass eine Sammlung von ausführbaren Inhalten auf alle Element einer Liste ausgeführt wird, kann es hier

zu Problemen kommen. Der Grund dafür ist, wie bereits in Unterabschnitt 3.1 genannt, dass XQuery-Änderungen erst am Schluss durchführt. Befinden sich z.B. in einem `foreach`-Element mehrere `assign`-Elemente, die auf die selben Daten zugreifen, wird eine Fehlermeldung geworfen. Daher wird dieses Element nicht unterstützt.

Bei der Umsetzung des Interpreters spielte die Umsetzung von Transaktionssicherheit keine Rolle, deswegen wurde sie nicht beachtet. Nichtsdestotrotz soll hier darauf eingegangen werden wie mit welchen Änderungen der vorliegenden Implementierung Transaktionssicherheit erreicht werden kann. Es wäre nötig, sich um die Konsistenz der Datenbank im Falle eines Fehlers in einer von mehreren, aufeinander folgenden Transaktionen zu kümmern. Die anderen Voraussetzungen für Transaktionssicherheit (Atomizität, Isolation und Dauerhaftigkeit) werden bereits durch das Transaktionsmanagement der `BaseX`-Datenbank gewährleistet [BaseX Team, 2016].

In einem SCXML-Modell kann das Ausführen aller Aktionen (z.B. ausführbare Inhalte) innerhalb einer Transition als Transaktion gesehen werden. Diese Aktionen sollten daher als logische Einheit betrachtet und nur gemeinsam durchgeführt werden. Wenn eine Aktion zu einem Fehler führt, werden die bereits durchgeführten Änderungen in der Datenbank rückgängig gemacht.

Eine relativ einfache Variante wäre, mithilfe von Snapshots des Datenbankinhalts zu wiederherstellbaren Zuständen vor jeder Transaktion zu gelangen. Ein geeigneter Ort um Snapshots zu erzeugen ist in der Funktion `controller`, da diese als Wrapper für alle Funktionen des Interpreters, die zu Transaktionen auf der Datenbank führen, dient. Kommt es während einer Transaktion zu einem Fehler, könnte der zuvor erstellte Snapshot verwendet werden um den ursprünglichen Zustand wiederherzustellen.

Allerdings muss man darauf hinweisen, dass das Wiederherstellen eines Snapshots aus Sicht des Interpreters dazu führt, dass die Zeit auf jenen Zeitpunkt zurückgedreht wird, an dem der Snapshot erstellt wurde. Dies bedeutet, dass externe Events, in der Zeit zwischen Snapshot-Erstellung und Snapshot-Wiederherstellung empfangen wurden verloren gehen und somit erneut an das System übermittelt werden müssen.

4 Worklist Handler

In diesem Kapitel ist die Benutzungsschnittstelle beschrieben, welche dem User erlaubt Events an ein MBA zu schicken und die zu einem Event gespeicherten Logs abzuholen. Dabei wird unterschieden zwischen der REST-Schnittstelle, welche die tatsächlichen Änderungen durchführt, sowie der grafischen Benutzungsschnittstelle, welche die Benutzung der REST-Schnittstelle vereinfacht.

4.1 Funktionen der REST-Schnittstelle

Die Benutzungsschnittstelle ist ebenfalls in der RESTXQ-Schnittstelle des Interpreters eingebettet. Dies erlaubt einen Zugriff über das HTTP-Protokoll. Die von Benutzerinnen und Benutzer aufgerufenen RESTXQ-Funktionen sind in Tabelle 33 beschrieben. Die Daten der Parameter werden jeweils im Gegensatz zur restlichen RESTXQ-Schnittstelle über HTTP-POST und HTTP-GET Requests an die Folgefunktion weitergeleitet.

Die Funktion `getEvents` liefert an den Benutzer alle Events, die im übergebenen MBA eine Transition auslösen können. Dabei werden alle Transitionen der aktiven Zustände des MBAs betrachtet und ihre Eventbedingungen zurückgegeben.

```
1   <?xml-stylesheet href="../static/start.xsl" type="text/xsl" ?>
2     <output>
3         <auswahl>
4             <event>setDegree</event>
5             <event>addCourse</event>
6             <event>done</event>
7             <event>setMaxDuration</event>
8             <event>addModule</event>
9         </auswahl>
10        <hidden>
11            <dbName>myMBAse</dbName>
12            <collectionName>
13                JohannesKeplerUniversity
14            </collectionName>
15            <mbaName>InformationSystems</mbaName>
16        </hidden>
17    </output>
```

Quellcode 18: Benutzungsschnittstelle: Rückgabewert der Funktion getEvents

In Quellcode 18 ist ein beispielhafter Rückgabewert der Funktion angegeben. Unterhalb des Elements `auswahl` befinden sich die einzelnen Namen der Events,

unterhalb des Elements `hidden` befinden sich die Daten zum MBA, welche von der grafischen Benutzungsschnittstelle nicht angezeigt werden.

Funktion	Parameter	HTTP Methode
getEvents	$dbName xs:string $collectionName xs:string $mbaName xs:string	HTTP-GET
getDataElements	$dbName xs:string $collectionName xs:string $mbaName xs:string $event xs:string	HTTP-GET
insertEvent	$data	HTTP-POST
getResult	$dbName xs:string $collectionName xs:string $mbaName xs:string $counter number	HTTP-GET

Tabelle 31: WorklistHandler : RESTXQ-REST-Funktionen

Die aufbauende Funktion `getDataElements` liefert alle Datenelemente eines MBAs, die von den ausführbaren Inhalten der Transitionen, die vom übergebenen Event ausgelöst werden können, benötigt werden.Dazu werden die ausführbaren Inhalte der zu verlassenen Zustände, der Transitionen sowie der wahrscheinlich zu betretenden Zustände untersucht.

Element	Beschreibung
dbName	Name der Datenbank in dem das MBA enthalten ist.
collectionName	Name der Collection in der das MBA enthalten ist.
mbaName	Name des MBAs dem ein Event hinzugefügt werden soll.
event	Name des einzufügenden Events
ownDataName	Name des eigenen Datenelements. Wird für die grafische Benutzungsschnittstelle benötigt.
ownDataData	Inhalt des eigenen Datenelements. Wird für die grafische Benutzungsschnittstelle benötigt.

Tabelle 32: WorklistHandler : insertEvent-Funktion

Für Änderungen in der Datenbank und dem tatsächlichen Hinzufügen eines Events zur Liste der externen Events wird die Funktion `insertEvent` verwendet. In Tabelle 32 sind die definierten Parameter für diese Funktion angegeben.

Die beiden Parameter `ownDataName` sowie `ownDataData` werden für die grafische Benutzungsschnittstelle benötigt. Sind diese Elemente befüllt wird dem Event ein Datenelement mit dem Namen des Parameters `ownDataName` und dem Inhalt des Parameters `ownDataData` hinzugefügt.

Zusätzliche Parameter stehen für zusätzliche Datenelemente die dem Event hinzugefügt werden. Der Name des Parameters wird als Name des Datenelements verwendet, der Inhalt des Datenelements ist der Inhalt des übergebenen Parameters. Eine mögliche REST-URL um ein Event hinzuzufügen ist in Quellcode 19 beschrieben.

```
1 http://<server>/insertEvent
2   &event=setDegree
3   &dbName=myMBAse
4   &collectionName=JohannesKeplerUniversity
5   &mbaName=InformationSystems
6   &degree=Mag.
```

Quellcode 19: Benutzungsschnittstelle: Hinzufügen eines Events

Das Event wird schlussendlich in der Funktion `insertEvent` zusammengesetzt, dabei werden die übergebenen Parameter den zugehörigen Variablen zugeordnet und die restlichen Parameter als Datenelemente des Events festgelegt. Das festlegen der zusätzlichen Datenelemente ist in Quellcode 20 dargestellt.

```
1 let $input :=
2     <input>{
3         let $row :=  fn:tokenize($data,'&amp;')
4             for $r in $row
5             let $data := fn:tokenize( $r, '=')
6             return element{$data[1]}{$data[2]}
7     }</input
8 let $eventData :=
9    $input/*[not(functx:is-value-in-sequence(name(.),
10   ("mbaName","dbName","collectionName","event",
11   "ownDataName","ownDataData")))]
12 ...
```

Quellcode 20: Benutzungsschnittstelle: Berechnung der Eventdaten

```
1 <?xml-stylesheet href="../static/event.xsl" type="text/xsl" ?>
2 <setDegree>
3     <input>
4         <data>degree</data>
5         <hidden>
6             <dbName>myMBAse</dbName>
7             <collectionName>
8                 JohannesKeplerUniversity
9             </collectionName>
10            <mbaName>InformationSystems</mbaName>
11        </hidden>
12    </input>
13 </setDegree>
```

Quellcode 21: Benutzungsschnittstelle: Rückgabewert der Funktion getDataElements

4.2 Grafische Benutzungsschnittstelle

Um das Hinzufügen von Events für den Benutzer zu vereinfachen, wird eine grafische Benutzeroberfläche zur Verfügung gestellt. Über eine URL können Benutzerinnen und Benutzer die Schnittstelle im Browser aufrufen. Die Darstellung im Browser wird mithilfe des CSS-Frameworks Bootstrap vereinheitlicht. Dank Optimierung für unterschiedliche Bildschirmgrößen wird damit auch die Verwendung der Oberfläche auf mobilen Endgeräten ermöglicht [Otto et al., 2016]. Üblicherweise rufen Benutzerinnen und Benutzer dabei zuerst die Funktion `enterMBA` auf,dargestellt in Tabelle 4.2. Die beiden Einstiegsfunktionen sind ebenfalls in der REST-Schnittstelle implementiert, sind jedoch keine REST-Funktionen, sondern nur statische HTML-Seiten.

Funktion	Beschreibung
enterMBA	In dieser Funktion werden die Daten für das MBA eingegeben, welchem anschließend ein Event hinzugefügt werden soll.
queryMBA	In dieser Funktion werden die Daten für das MBA und der Wert des Zählers eingegeben, zu welchem anschließend die gespeicherten Logs angesehen werden möchten.

Tabelle 33: WorklistHandler : RESTXQ-Funktionen

Die erste aufzurufende Funktion `enterMBA` dient den Benutzerinnen und Benutzern zum Eingeben der Parameter des MBAs dem ein Event hinzugefügt werden soll. Sollte das MBA nicht vorhanden sein, werden Benutzerinnen und Benutzer

mithilfe einer Fehlermeldung informiert und die Funktion `enterMBA` wird erneut aufgerufen.

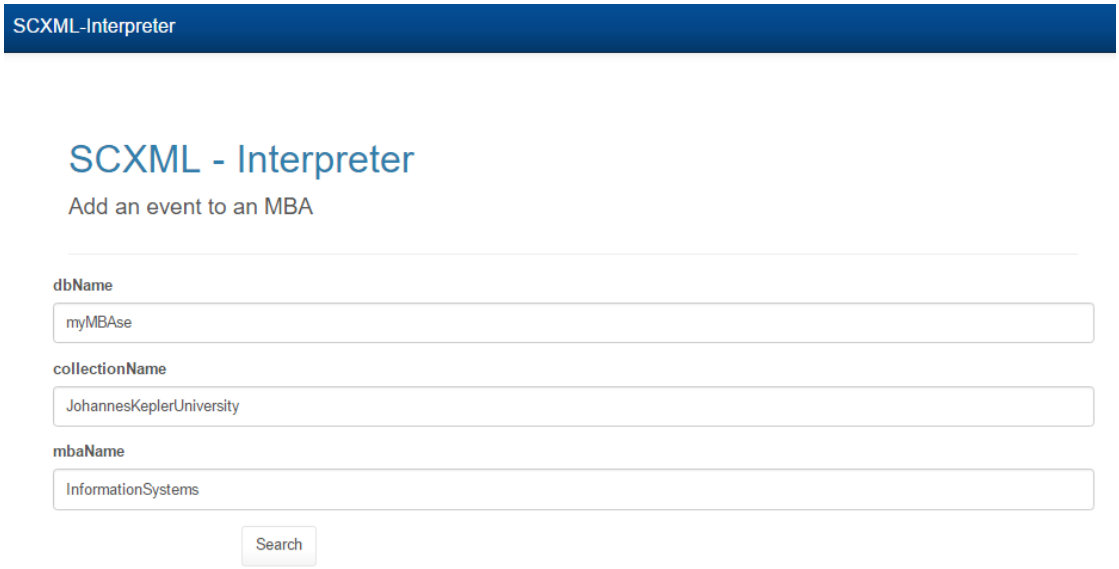

Abbildung 10: Benutzungsschnittstelle: enterMBA

Nach der Eingabe eines korrekten MBAs wird auf die Folgefunktion `getEvents` weitergeleitet, wie in Abbildung 11 dargestellt. Die Funktion sind nimmt das ausgewählte MBA und überprüft, welche Transitionen aufgrund des aktuellen Zustands im Moment möglich wären. Es werden alle Transitionen in den aktuell aktiven Zuständen ausgewählt, und auf etwaige benötigte Events überprüft. Diese Events werden nun den Benutzerinnen und Benutzern als mögliche Events vorgeschlagen. Dies geschieht um den Benutzerinnen und Benutzern die Verwendung des `WorklistHandlers` zu vereinfachen.

Das Ergebnis der Funktion, zu sehen in Quellcode 18, wird an den Browser gesendet. Um eine ansprechende Darstellung zu erreichen wird, wie in Zeile 1 dargestellt, neben dem XML-Dokument auch ein Stylesheet mitgegeben. Der Browser wandelt die erhaltenen XML-Daten mit Hilfe dieses Stylesheets in eine HTML-Seite um, zu sehen in Abbildung 12. Diese Transformation erfolgt über eine XSLT-Transformation. Die Elemente des XML-Dokuments werden dazu von XSLT-Prozessoren anhand der angegebenen Styleinformationen dargestellt. Die Benutzerinnen und Benutzer können aus den vom Interpreter vorgeschlagenen

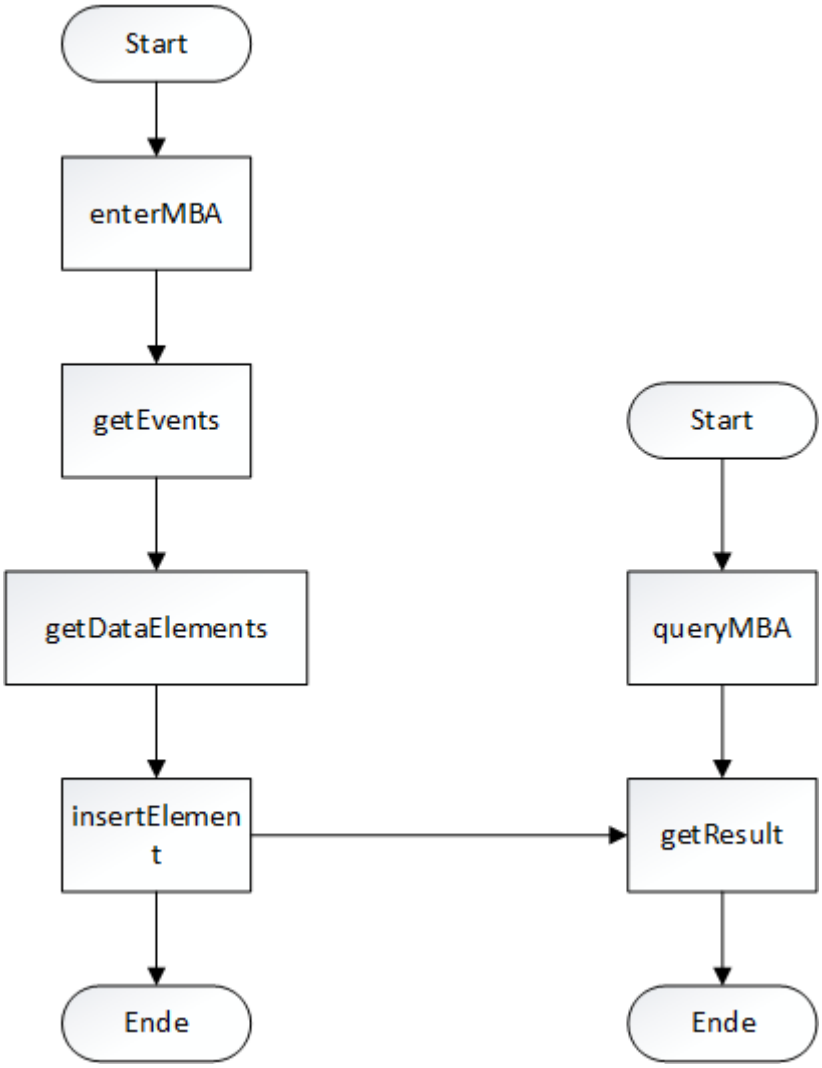

Abbildung 11: Ablauf der Benutzungsschnittstelle

Events auswählen oder auch ein neues eigenes Event definieren. Alle RESTXQ-Funktionen des WorklistHandlers mit Ausnahme von `enterMBA` und `queryMBA` liefern XML-Dokumente mit verknüpften Styleiformationen.

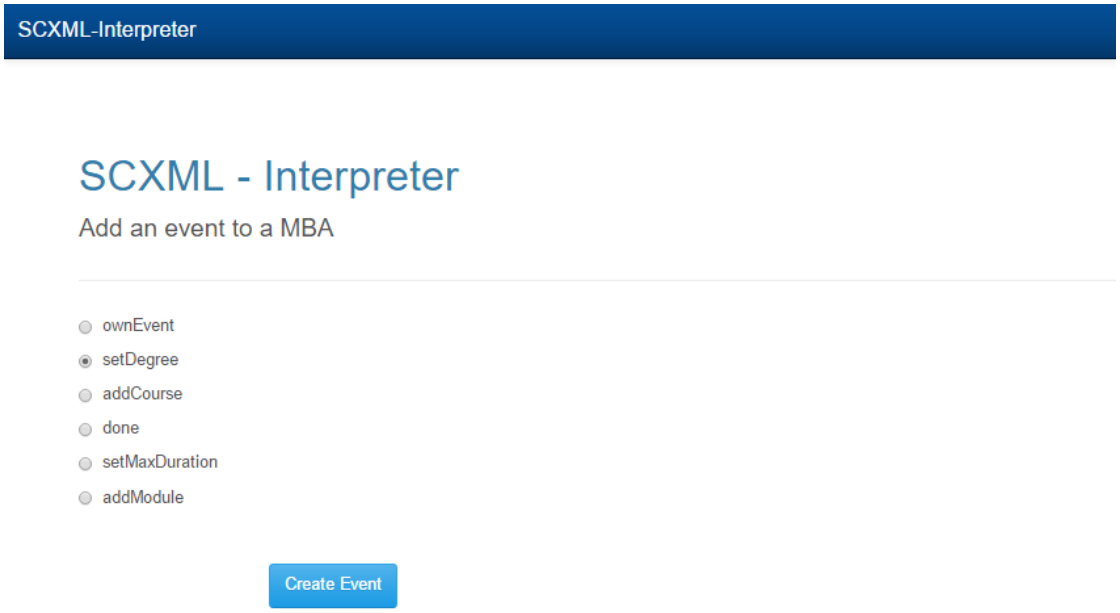

Abbildung 12: Benutzungsschnittstelle: getEvents

Für die Funktion `getEvents` wird der im obigen Teil verwendete Ausschnitt des XSLT-Dokuments verwendet Quellcode 22. Wie in Quellcode 18 erhält der XSLT-Transformator ein XML-Dokument, das ein Element `auswahl` enthält. Für jedes `event`-Element wird ein Radio-Button erstellt, welcher den Namen des Events sowohl als Name als auch als Wert beinhaltet. Zusätzlich werden die Informationen, welche innerhalb des `hidden`-Tags vorhanden sind, ebenfalls an die Folgefunktion weitergeleitet. Dabei werden sie ebenfalls über eine XSLT-Transformation in ein Feld umgewandelt, das jedoch das Attribut `hidden` erhält und somit nicht im Browser angezeigt wird.

Anschließend wird `getDataElements` an die Folgefunktion weitergeleitet. Auch hier versucht die Funktion, den Benutzerinnen und Benutzern sinnvolle Vorschläge zu unterbreiten. Daher ermittelt der Interpreter mögliche Ziele der Transitionen und die dort benötigten Datenelemente. Anschließend stellt die Funktion einen Liste mit diesen Datenelementen zusammen und gibt sie an den Browser zurück. Auch

```
1 ...
2     <xsl:template match="auswahl">
3         <xsl:for-each select="event">
4             <div class="radio">
5                 <label>
6                 <input type="radio" name="event">
7                     <xsl:attribute name="value">
8                         <xsl:value-of select="."/>
9                     </xsl:attribute>
10                </input>
11                <xsl:value-of select="."/>
12                </label>
13            </div>
14        </xsl:for-each>
15    </xsl:template>
16 ...
```

Quellcode 22: Benutzungsschnittstelle: XSL Stylesheet für die Funktion getEvents

hier erhält der Browser ein XML-Dokument, siehe Quellcode 21, das aufgrund des Stylesheets eine XSLT-Transformation durchläuft.

Im Beispiel von Abbildung 13 wurde das Event `setDegree` ausgewählt. Dieses Event verlangt das Datenelement `degree`, das anschließend im SCXML-Modell verarbeitet wird. Es ist jedoch auch möglich benutzerdefinierte Datenelemente zu erstellen.

Durch Klick auf Senden wird das Event mit der Funktion `insertEvent` als externes Event an das MBA angehängt. Die Benutzerinnen und Benutzer erhalten eine kurze Übersicht über die Daten des hinzugefügten Events, d.h. es werden sowohl die Daten des MBAs dem die Daten eingefügt worden sind, als auch die Daten des Events angezeigt.

Das neu erzeugte Event wird über die Funktion `mba:enqueueExternalEvent` einem MBA hinzugefügt. Für die Benutzungsschnittstelle sind darüber hinaus die in Tabelle 34 beschriebenen Funktionen `sc:getCounter` sowie `sc:updateCounter` relevant. Jedem von einer Benutzerin oder einem Benutzer erzeugten Event wird ein eindeutiger Zähler zugewiesen. Er ermöglicht die Auswertung der dem Event zugehörigen Logs. Mithilfe der Funktion `sc:updateCounter` wird der Wert internen Zählers um eine Stelle erhöht.

Damit Benutzerinnen oder Benutzer die Logs zu einem späteren Zeitpunkt auswerten können, steht die Funktion `queryMBA` zur Verfügung. Sie verlangt nach den Daten zur Identifikation eines MBAs sowie den eindeutigen Zähler des Events. Die Eingabemaske dieser Funktion ist eine statische HTML-Seite und beinhaltet keine XSLT-Transformation.

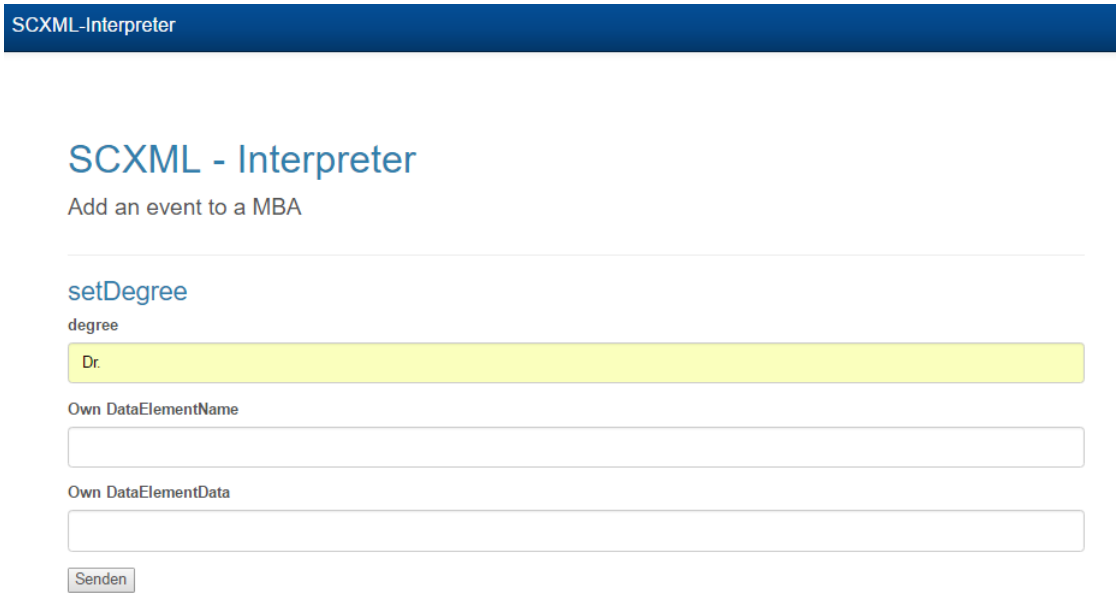

Abbildung 13: Benutzungsschnittstelle: getDataElements

Funktion	Parameter	Beschreibung
sc:getResult	$dbName xs:string $collectionName xs:string $mbaName xs:string $id xs:integer	Gibt das Ergebnis einer Abfrage zurück. Diese Funktion wird für Benutzerabfrage verwendet.
sc:getCounter	$dbName xs:string $collectionName xs:string $mbaName xs:string	Gibt den aktuellen Wert des Zählers für die Speicherung der Anfragen von Benutzerinnen oder Benutzern zurück.
sc:updateCounter	$dbName xs:string $collectionName xs:string $mbaName xs:string	Erhöht den Wert des Zählers für die Speicherung der Anfragen von Benutzerinnen oder Benutzern um 1.

Tabelle 34: WorklistHandler: Benutzungsschnittstelle

SCXML-Interpreter

SCXML - Interpreter

Add an event to a MBA

Added Event

Element	Id	Value
counter		5
eventName		setDegree
dbName		myMBAse
collectionName		JohannesKeplerUniversity
mbaName		InformationSystems
degree		Dr.

getLogs

Abbildung 14: Benutzungsschnittstelle: addEvent

SCXML-Interpreter

SCXML - Interpreter

Hinzufügen von Events

dbName

myMBAse

collectionName

JohannesKeplerUniversity

mbaName

InformationSystems

Counter

1

Search

Abbildung 15: Benutzungsschnittstelle: queryMBA

Die Funktion `getResult` liefert im Anschluss an die Funktionen `queryMBA` bzw. `insertEvent` dem Benutzer alle zu diesem Event geloggten Daten. Diese Logs können sich allerdings noch verändern, da die Daten erst gespeichert werden, wenn der Interpreter das Event aufgreift bzw. erst dann, wenn die Process Environment den Interpreter startet.

SCXML-Interpreter

SCXML - Interpreter

Get Result

Element	Id	Value
data	degree	Mag.
log		Ein InformationSystemsProgram log fuer unseren Log
dbName		myMBAse
collectionName		JohannesKeplerUniversity
mbaName		InformationSystems
counter		1

Abbildung 16: Benutzungsschnittstelle: getResult

Um die für den Benutzer relevanten Logs zu erhalten wird auf die letzte Funktion in Tabelle 34 zugegriffen. Sie selektiert die für den Benutzer relevanten Zähler der Variable `response` und liefert das Ergebnis an die RESTXQ-Funktion `getResult` zurück. In Abbildung 16 sind die Ergebnisse des in diesem Kapitel verwendeten Beispiels dargestellt.

In der Variable `response` werden keine allgemeinen Ereignisse, wie z.B. das Betreten von Zuständen, geloggt. Dieses Logging wird von der in Unterabschnitt 3.4 beschriebenen Funktion übernommen. Die RESTXQ-Schnittstelle liefert Elemente, die bereits im SCXML-Modell vordefiniert sind. Daher werden im für die Benutzerinnen und Benutzer relevanten Log die beiden ausführbaren Inhalte `log` und `getValue` gespeichert.

Benutzerkontrolle In der vorliegenden Implementierung erfolgt keine Benutzerkontrolle, d.h. jeder Benutzer darf den aktuellen Zustand sehen und kann Events

hinzufügen. Um eine tatsächliche Benutzerkontrolle zu ermöglichen wäre es notwendig, sowohl Authentifizierung- als auch und Autorisierungs-Funktionen für Benutzerinnen und Benutzer hinzuzufügen um sicherzustellen, dass keine unerlaubten Operationen von nicht zugelassen Personen durchgeführt werden.

Eine mögliche Implementierung stützt sich auf die Nutzung des von BaseX angebotenen Benutzer-Managements. Dieses bietet fünf hierarchisch aufgebaute Berechtigungsstufen für Benutzer (admin, create, write, read, none). Im vorliegenden Fall könnte man ein vereinfachtes, aus lediglich drei Stufen bestehendes Rechtemodell verwenden. Benutzerinnen und Benutzer mit höchster Sicherheitsstufe bekommen administrative Rechte und können die gesamte Datenbank verändern sowie neue Benutzer anlegen und Rechte vergeben. Die zweite Sicherheitsstufe berechtigt zur Veränderung der Datenbank, beispielsweise durch das Hinzufügen von Events. Benutzerinnen und Benutzer mit der niedrigsten Sicherheitsstufe können nur Informationen lesen und keine Events hinzufügen.

5 Fazit und Ausblick

Die Abbildung von Prozessmodellen von Multilevel Business Artifacts mittels State Chart XML (SCXML) ermöglicht die Automatisierung von mehrstufigen Geschäftsprozessen. Ein SCXML-Interpreter garantiert dabei den gültigen Ablauf eines Prozessmodells und führt automatische Aktionen aus, die mit den Prozessschritten verknüpft sind. Der in dieser Arbeit vorgestellte SCXML-Interpreter orientiert sich am SCXML-Standard (mit kleineren Einschränkungen) und unterstützt das XPath-Datenmodell, welches aufgrund mangelnder Werkzeugunterstützung nicht Teil der letztgültigen W3C-Empfehlung ist. Der in dieser Arbeit gewählte Implementierungsansatz mittels RESTXQ erlaubt die sequenzielle Abarbeitung von Prozessmodellen, obwohl XQuery Update-Operationen grundsätzlich ohne vorgegebene Reihenfolge ausführt. Bei der vorgestellten Implementierung handelt es sich um einen Proof-of-Concept-Prototypen. Aufbauend auf die vorliegende Arbeit können folgende Themen in Zukunft bearbeitet werden:

- **Quantitative und qualitative Prozessanalyse:** Die quantitative und qualitative Analyse von Logdaten, die bei der Ausführung von mehrstufigen Geschäftsprozessen entstehen, kann als Grundlage für die Optimierung und Überwachung von Prozessen verwendet werden.

- **Werkzeuge für Modellierung mehrstufiger Geschäftsprozesse:** Eine wichtige Komponente von BPMS ist das Modellierungswerkzeug. Die vorliegende Arbeit konzentriert sich auf die Automatisierung von mehrstufigen Geschäftsprozessen und erwartet syntaktisch und semantisch richtige Prozessmodelle.

- **Mehrbenutzerkontrolle und Transaktionssicherheit:** Die vorliegende Arbeit unterscheidet nicht zwischen verschiedenen Benutzerinnen und Benutzern. Darüber hinaus wird Transaktionssicherheit nicht gewährleistet. Zukünftige Arbeiten müssen sicherstellen, dass die Aktionen einer Transition als Transaktion ausgeführt werden.

6 Literatur

[Apache Commons, 2015] Apache Commons (2015). Apache Commons SCXML, Version 2.0. Abgerufen am 08.09.2016 von http://commons.apache.org/proper/commons-scxml/.

[Apache Commons, 2016] Apache Commons (2016). Apache Commons SCXML, Roadmap. Abgerufen am 08.09.2016 von http://commons.apache.org/proper/commons-scxml/roadmap.html.

[Auburn et al., 2011] Auburn, R., Baggia, P., und Scott (2011). Voice browser call control: Ccxml version 1.0. *Recommendation, W3C*.

[Barbier, 2016a] Barbier, F. (2016a). Pauware view. Abgerufen am 08.09.2016 von http://www.franckbarbier.com/PauWare/PauWare_view/.

[Barbier, 2016b] Barbier, F. (2016b). *Reactive Internet Programming: State Chart XML in Action*. Association for Computing Machinery and Morgan; Claypool, New York, NY, USA.

[Barbier et al., 2015] Barbier, F., Le Goaer, O., und Cariou, E. (2015). Energized state charts with pauware. *Engineering Interactive Computer Systems with SCXML*, page 22.

[Barnett, 2015] Barnett, J. (2015). SCXML 1.0 Implementation Report. Abgerufen am 08.09.2016.

[Barnett et al., 2014] Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D. C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T., Reifenrath, K., Rosenthal, N., und Roxendal, J. (2014). State Chart XML (SCXML): State Machine Notation for Control Abstraction W3C Candidate Recommendation 13 March 2014. Abgerufen am 08.09.2016 https://www.w3.org/TR/2014/CR-scxml-20140313/.

[Barnett et al., 2015] Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D. C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T., Reifenrath, K., Rosenthal, N., und Roxendal, J. (2015). State Chart XML (SCXML): State Machine Notation for Control Abstraction W3C Recommendation 1 September 2015. Abgerufen am 08.09.2016 https://www.w3.org/TR/2015/REC-scxml-20150901/.

[BaseX Team, 2016] BaseX Team (2016). BaseX Documentation, Version 8.5. *URL: http://files.basex.org/releases/8.5/BaseX85.pdf*. Abgerufen am 08.09.2016 von http://files.basex.org/releases/8.5/BaseX85.pdf.

[Becker, 2012] Becker, J. (2012). Geschäftsprozessmodellierung. Abgerufen am 04.11.2016 von http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Hauptaktivitaten-der-Systementwicklung/Problemanalyse-/Geschaftsprozessmodellierung/.

[Datypic, 2014] Datypic (2014). FunctX XQuery Function Library. Abgerufen am 08.09.2016 von http://www.xqueryfunctions.com/.

[Dumas et al., 2013] Dumas, M., La Rosa, M., Mendling, J., Reijers, H. A., et al. (2013). *Fundamentals of business process management*, Band 1. Springer.

[Dyck und Robie, 2014] Dyck, M. und Robie, J. (2014). XML Path Language (XPath) 3.1. W3C Working Draft.

[Gadatsch, 2008] Gadatsch, A. (2008). *Geschäftsprozessmodellierung und-simulation*, Seiten 135–263. Vieweg, Wiesbaden.

[Günther und Verbeek, 2014] Günther, C. W. und Verbeek, H. (2014). Xes-standard definition (2014).

[Harel, 1987] Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231 – 274.

[Ierusalimschy et al., 1996] Ierusalimschy, R., De Figueiredo, L. H., und Celes Filho, W. (1996). Lua-an extensible extension language. *Softw., Pract. Exper.*, 26(6):635–652.

[Kistner, 2014] Kistner, G. (2014). LXSC - Lua XML StateCharts. Abgerufen am 08.09.2016 von https://github.com/Phrogz/LXSC.

[Mahmood et al., 2009] Mahmood, I., Ayani, R., Vlassov, V., und Moradi, F. (2009). Statemachine matching in bom based model composition. In *Distributed Simulation and Real Time Applications, 2009. DS-RT '09. 13th IEEE/ACM International Symposium on*, Seiten 136–143.

[Moura und Guedes, 2007] Moura, R. S. und Guedes, L. A. (2007). Simulation of industrial applications using the execution environment scxml. In *2007 5th IEEE International Conference on Industrial Informatics*, Band 1, Seiten 255–260.

[Otto et al., 2016] Otto, M., Thornton, J., et al. (2016). Bootstrap· the world's most popular mobile-first and responsive front-end framework.'. *Getbootstrap.com*. Abgerufen am 08.09.2016 von http://getbootstrap.com/about/.

[Retter, 2016] Retter, A. (2016). RESTXQ 1.0: RESTful Annotations for XQuery An EXQuery Project Unofficial Draft 21 March 2016 . Abgerufen am 08.09.2016 von http://exquery.github.io/exquery/exquery-restxq-specification/restxq-1.0-specification.html.

[Robie et al., 2014] Robie, J., Chamberlin, D., Dyck, M., und Snelson, J. (2014). XQuery 3.0: An XML Query language. *Recommendation, W3C*.

[Romeo et al., 2016] Romeo, F., Ballagny, C., und Barbier, F. (2016). Pauware. Abgerufen am 08.09.2016 von http://www.pauware.com/.

[Schuetz et al., 2015] Schuetz, C. G., Neumayr, B., und Schrefl, M. (2015). Multi-level modeling for business process automation. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, Seiten 51–60.

[Schuetz, 2015a] Schuetz, G. C. (2015a). *Multilevel Business Artifacts*, Seiten 57–104. Springer Fachmedien Wiesbaden, Wiesbaden.

[Schuetz, 2015b] Schuetz, G. C. (2015b). *Multilevel Business Process Automation*, Seiten 171–195. Springer Fachmedien Wiesbaden, Wiesbaden.

[Steiner, 1971] Steiner, G. A. (1971). *Top Management Planung*. Verlag Moderne Industrie.

[Steinmann und Schreyögg, 2000] Steinmann, H. und Schreyögg, G. (2000). Management: Grundlagen der Unternehmensführung, 5. *Aufl., Wiesbaden*, 1.

[Touffy, 2015] Touffy (2015). JavaScript State Chart Interpreter. Abgerufen am 08.09.2016 von https://github.com/Touffy/JSSCxml.

[Weske, 2007] Weske, M. (2007). *Business Process Modelling Foundation*, Seiten 73–124. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Zhang et al., 2012] Zhang, L., Shen, X., und Xiong, W. (2012). The Query and Application of XML Data Based on XQuery. In *Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on*, Seiten 574–577.

Testfälle

Um die Umsetzung des Business Process Management Systems zu testen wurde unter anderem das JUnit-Framework verwendet. Die Verwendung dieses Frameworkes erfolgte in der Process Environment. Die zu testenden SCXML-Diagramme wurden dem SCXML 1.0 Implementation Report [Barnett, 2015] entnommen. Um dieses Tests verwenden zu können, war es zuerst notwendig diese, in einer Pseudonotation angegebenen Tests, mit XSLT in SCXML-Modelle die das XPath-Datenmodell unterstützen zu verwandeln. Im nächsten Schritt wurde nun dieses SCXML-ELement in ein MBA eingefügt das anschließend von der Process Environment aufgerufen werden konnte. Um Fehlerfälle durch falsche Konfigurationen ausschließen zu können wurde für die im Standard vorhandenen Elemente immer das gleiche MBA verwendet.

Assertion Number	Conformance	Abstract	Test	Notiz
355	m	At system initialization time, if the 'initial' attribute is not present, the Processor MUST enter the first state in document order.	OK	
576	m	At system initialization time, the SCXML Processor MUST enter the states specified by the 'initial' attribute, if it is present.	OK	
364	m	Definition: The default initial state(s) of a compound state are those specified by the 'initial' attribute or initial element, if either is present. Otherwise it is the state's first child state in document order. If a compound state is entered either an initial state or the target of a transition (i.e. and no descendent of it is specified), then the SCXML Processor MUST enter the default initial state(s) after it enters the parent state.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
372	m	When the state machine enters the final child of a state element, the SCXML processor MUST generate the event done.state.id after completion of the onentry elements, where id is the id of the parent state.	OK	
570	m	Immediately after generating done.state.id upon entering a final child of state, if the parent state is a child of a parallel element, and all of the parallel's other children are also in final states, the Processor MUST generate the event done.state.id where id is the id of the parallel element.	OK	
375	m	The SCXML processor MUST execute the onentry handlers of a state in document order when the state is entered.	OK	
376	m	The SCXML processor MUST treat each [onentry] handler a separate block of executable content.	OK	
377	m	The SCXML processor MUST execute the onexit handlers of a state in document order when the state is exited.	OK	
378	m	The SCXML processor MUST treat each [onexit] handler a separate block of executable content.	OK	
387	m	Before the parent state has been visited for the first time, if a transition is executed that takes the history state its target, the SCXML processor MUST behave if the transition had taken the default stored state configuration its target.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
579	m	Before the parent state has been visited for the first time, if a transition is executed that takes the history state its target, the SCXML processor MUST execute any executable content in the transition after the parent state's onentry content and any content in a possible initial transition.	OK	
580	m	It follows from the semantics of history states that they never end up in the state configuration	OK	
388	m	After the parent state has been visited for the first time, if a transition is executed that takes the history state its target, the SCXML processor MUST behave if the transition had taken the stored state configuration its target.	OK	
396	m	The SCXML processor MUST use this same name value [the one reflected in the event variable] to match against the 'event' attribute of transitions.	OK	
399	m	[Definition: A transition matches an event if at least one of its event descriptors matches the event's name.] [Definition: An event descriptor matches an event name if its string of tokens is an exact match or a prefix of the set of tokens in the event's name. In all cases, the token matching is case sensitive.]	OK	
401	m	The processor MUST place these [error] events in the internal event queue.	OK	
402	m	The processor MUST process them [error events] like any other event.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
403	m	To execute a microstep, the SCXML Processor MUST execute the transitions in the corresponding optimal enabled transition set, where the optimal transition set enabled by event E in state configuration C is the largest set of transitions such that a) each transition in the set is optimally enabled by E in an atomic state in C b) no transition conflicts with another transition in the set c) there is no optimally enabled transition outside the set that has a higher priority than some member of the set.	OK	
404	m	To execute a set of transitions, the SCXML Processor MUST first exit all the states in the transitions' exit set in exit order.	OK	
405	m	[the SCXML Processor executing a set of transitions] MUST then [after the onexits] execute the executable content contained in the transitions in document order.	OK	
406	m	[the SCXML Processor executing a set of transitions] MUST then [after the exits and the transitions] enter the states in the transitions' entry set in entry order.	OK	
407	m	To exit a state, the SCXML Processor MUST execute the executable content in the state's onexit handler.	OK	
409	m	Finally [after the onexits and canceling the invocations], the Processor MUST remove the state from the active state's list.	OK	
411	m	To enter a state, the SCXML Processor MUST add the state to the active state's list. Then it MUST execute the executable content in the state's onentry handler.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
412	m	If the state is a default entry state and has an initial child, the SCXML Processor MUST then [after doing the active state add and the onentry handlers] execute the executable content in the initial child's transition.	OK	
413	m	At startup, the SCXML Processor MUST place the state machine in the configuration specified by the 'initial' attribute of the scxml element.	OK	
415	m	If it [the SCXML Processor] has entered a final state that is a child of scxml [during the last microstep], it MUST halt processing.	OK	
416	m	If it [the SCXML processor] has entered a final state that is a child of a compound state [during the last microstep], it MUST generate the event done.state.id, where id is the id of the compound state.	OK	
417	m	If the compound state [which has the final element that we entered this microstep] is itself the child of a parallel element, and all the parallel element's other children are in final states, the Processor MUST generate the event done.state.id, where id is the id of the parallel element.	OK	
419	m	After checking the state configuration, the Processor MUST select the optimal transition set enabled by NULL in the current configuration. If the [optimal transition] set [enabled by NULL in the current configuration] is not empty, it [the SCXML Processor] MUST execute it [the set] a microstep.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
421	m	If the set (of eventless transitions) is empty, the Processor MUST remove events from the internal event queue until the queue is empty or it finds an event that enables a non-empty optimal transition set in the current configuration. If it finds such a set [a non-empty optimal transition set], the processor MUST then execute it a microstep.	OK	
422	m	After completing a macrostep, the SCXML Processor MUST execute in document order the invoke handlers in all states that have been entered (and not exited) since the completion of the last macrostep.	OK , manually	delay[2]
423	m	Then [after invoking the new invoke handlers since the last macrostep] the Processor MUST remove events from the external event queue, waiting till events appear if necessary, until it finds one that enables a non-empty optimal transition set in the current configuration. The Processor MUST then execute that set [the enabled non-empty optimal transition set in the current configuration triggered by an external event] a microstep.	OK	
503	m	If the transition does not contain a 'target', its exit set is empty.	OK	
504	m	if [a transition's] 'type' is ëxternal", its exit set consists of all active states that are proper descendents of the Least Common Compound Ancestor (LCCA) of the source and target states.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
505	m	Otherwise, if the transition has 'type' ïnternal", its source state is a compound state and all its target states are proper descendents of its source state, the target set consists of all active states that are proper descendents of its source state.	OK	
506	m	If a transition has 'type' of ïnternal", but its source state is not a compound state or its target states are not all proper descendents of its source state, its exit set is defined if it had 'type' of ëxternal".	OK	
533	m	If a transition has 'type' of ïnternal", but its source state is not a compound state, its exit set is defined if it had 'type' of ëxternal".	OK	
144	m	The SCXML processor MUST place the event that is generated by the raise element at the rear of the session's internal event queue.	OK	
147	m	When the if element is executed, the SCXML processor MUST execute the first partition in document order that is defined by a tag whose 'cond' attribute evaluates to true, if there is one.	OK	
148	m	When the if element is executed, if no 'cond'attribute evaluates to true, the SCXML Processor must execute the partition defined by the else tag, if there is one.	OK	
149	m	When it executes an if element, if no 'cond' attribute evaluates to true and there is no else element, the SCXML processor must not evaluate any executable content within the element.	OK	
150	m	In the foreach element, the SCXML processor MUST declare a new variable if the one specified by 'item' is not already defined.	No	foreach[5]

Assertion Number	Conformance	Abstract	Test	Notiz
151	m	In the foreach element, if 'index' is present, the SCXML processor MUST declare a new variable if the one specified by 'index' is not already defined.	No	foreach[5]
152	m	In the foreach element, if 'array' does not evaluate to a legal iterable collection, or if 'item' does not specify a legal variable name, the SCXML processor MUST terminate execution of the foreach element and the block that contains it, and place the error error.execution on the internal event queue.	No	foreach[5]
153	m	When evaluating foreach, the SCXML processor MUST start with the first item in the collection and proceed to the last item in the iteration order that is defined for the collection. For each item in the collection in turn, the processor MUST assign it to the item variable.	No	foreach[5]
155	m	when evaluating foreach, for each item, after making the assignment, the SCXML processor MUST evaluate its child executable content. It MUST then proceed to the next item in iteration order.	No	foreach[5]
156	m	If the evaluation of any child element of foreach causes an error, the processor MUST cease execution of the foreach element and the block that contains it.	No	foreach[5]
525	m	The SCXML processor MUST act if it has made a shallow copy of the collection produced by the evaluation of 'array'. Specifically, modifications to the collection during the execution of foreach MUST NOT affect the iteration behavior.	No	foreach[5]
158	m	The SCXML processor MUST execute the elements of a block of executable contentin document order.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
159	m	If the processing of an element of executable content causes an error to be raised, the processor MUST NOT process the remaining elements of the block.	No	[6]
276	m	The SCXML Processor MUST allow the environment to provide values for top-level data elements at instantiation time. (Top-level data elements are those that are children of the datamodel element that is a child of scxml). Specifically, the Processor MUST use the values provided at instantiation time instead of those contained in these data elements.	OK	
277	m	If the value specified for a data element (by 'src', children, or the environment) is not a legal data value, the SCXML Processor MUST raise place error.execution in the internal event queue and MUST create an empty data element in the data model with the specified id.	OK	
279	m	When 'binding' attribute on the scxml element is assigned the value ëarly"(the default), the SCXML Processor MUST create all data elements and assign their initial values at document initialization time.	OK	
280	m	When 'binding' attribute on the scxml element is assigned the value "late", the SCXML Processor MUST create the data elements at document initialization time, but MUST assign the specified initial value to a given data element only when the state that contains it is entered for the first time, before any onentry markup.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
550	m	If the 'expr' attribute is present, the Platform MUST evaluate the corresponding expression at the time specified by the 'binding' attribute of scxml and MUST assign the resulting value the value of the data element	OK	
551	m	f child content is specified, the Platform MUST assign it the value of the data element at the time specified by the 'binding' attribute of scxml.	OK	
552	m	If the 'src' attribute is present, the Platform MUST fetch the specified object at the time specified by the 'binding' attribute of scxml and MUST assign it the value of the data element.	OK	
286	m	If the location expression of an assign does not denote a valid location in the datamodel the processor MUST place the error error.execution in the internal event queue.	OK	
287	m	If the location expression of an assign denotes a valid location in the datamodel and if the value specified by 'expr' is a legal value for the location specified, the processor MUST place the specified value at the specified location.	OK	
487	m	If the value specified (by 'expr' or children) is not a legal value for the location specified, the processor MUST place the error error.execution in the internal event queue.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
294	m	In cases where the SCXML Processor generates a 'done' event upon entry into the final state, it MUST evaluate the donedata elements param or content children and place the resulting data in the _event.data field. The exact format of that data will be determined by the datamodel	OK	
527	m	When the SCXML Processor evaluates the content element, if the 'expr' value expression is present, the Processor MUST evaluate it and use the result the output of the content element.	OK	
528	m	f the evaluation of 'expr' produces an error, the Processor MUST place error.execution in the internal event queue and use the empty string the output of the content element.	OK	
529	m	If the 'expr' attribute is not present, the Processor MUST use the children of content the output.	OK	
298	m	If the 'location' attribute on a param element does not refer to a valid location in the data model, the processor MUST place the error error.execution on the internal event queue.	OK	
343	m	If the 'location' attribute on a param element does not refer to a valid location in the data model, or if the evaluation of the 'expr' produces an error, the processor MUST ignore the name and value.	OK	
488	m	if the evaluation of the 'expr' produces an error, the processor MUST place the error error.execution on the internal event queue.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
301	m	If the script specified by the 'src' attribute of a script element cannot be downloaded within a platform-specific timeout interval, the document is considered non-conformant, and the platform MUST reject it. N.B. This test is valid only for datamodels that support scripting.	No	
302	m	The SCXML Processor MUST evaluate any script element that is a child of scxml at document load time. N.B. This test is valid only for datamodels that support scripting.	No	script[4]
303	m	The SCXML Processor MUST evaluate all script elements not children of scxml part of normal executable content evaluation. N.B. This test is valid only for datamodels that support scripting.	No	script[4]
304	m	In a conformant SCXML document, the name of any script variable MAY be used a location expression. N.B. This test is valid only for datamodels that support scripting.	No	script[4]
307	m	When "late"data binding is used, accessing data substructure in expressions before the corresponding data element is loaded MUST yield the same execution-time behavior accessing non-existent data substructure in a loaded data instance.	OK	script[4]
309	m	If a conditional expression cannot be evaluated a boolean value ('true' or 'false') or if its evaluation causes an error, the SCXML processor MUST treat the expression if it evaluated to 'false'.	, manually	delay[2]
310	m	All datamodels MUST support the 'In()' predicate, which takes a stateID its argument and returns true if the state machine is in that state.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
311	m	If a location expression cannot be evaluated to yield a valid location, the SCXML processor MUST place the error error.execution in the internal event queue.	OK	
312	m	If a value expression does not return a legal data value, the SCXML processor MUST place the error error.execution in the internal event queue.	OK	
313	m	The SCXML processor MAY reject documents containing syntactically ill-formed expressions at document load time, or it MAY wait and place error.execution in the internal event queue at runtime when the expressions are evaluated.	OK	
314	m	If the SCXML processor waits until it evaluates the expressions at runtime to raise errors, it MUST raise errors caused by expressions returning illegal values at the points at which Appendix A Algorithm for SCXML Interpretation indicates that the expressions are to be evaluated.	OK	
344	m	If a conditional expression cannot be evaluated a boolean value ('true' or 'false') or if its evaluation causes an error, the SCXML processor MUST place the error 'error.execution' in the internal event queue.	NO	[7]
318	m	The SCXML Processor MUST bind the _event variable when an event is pulled off the internal or external event queue to be processed, and MUST keep the variable bound to that event until another event is processed.	OK	
319	m	The SCXML Processor MUST NOT bind _event at initialization time until the first event is processed.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
321	m	The Processor MUST bind the variable _sessionid at load time to the system-generated id for the current SCXML session.	OK	
322	m	The Processor MUST keep the _sessionid variable bound to the system-generated id until the session terminates.	OK	
323	m	The Processor MUST bind the variable _name at load time to the value of the 'name' attribute of the scxml element. a	OK	
324	m	The Processor MUST keep the _name variable bound to the value of the 'name' attribute of the scxml element until the session terminates.	OK	
325	m	The Processor MUST bind the variable _ioprocessors to a set of values, one for each Event I/O Processor that it supports.	OK	
326	m	The Processor MUST keep the _ioprocessors variable bound to its set of values until the session terminates.	OK	
329	m	The Processor MUST cause any attempt to change the value of a system variable to fail.	OK	
330	m	The SCXML Processor MUST insure that the following fields (name, type, sendid, origin, origintype, invokeid, data) are present in all events (_event variable), whether internal or external.	OK	
331	m	The SCXML Processor MUST set the type property of _event to: "platform"(for events raised by the platform itself, such error events), ïnternal"(for events raised by raise and send with target '_internal') or ëxternal"(for all other events).	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
332	m	If the sending entity has specified a value for this, the Processor MUST set this field to that value. Otherwise, in the case of error events triggered by a failed attempt to send an event, the Processor MUST set the sendid field to the send id of the triggering send element.	OK	
333	m	For events other than error events triggered by a failed attempt to send an event, if the sending entity did not specify a value for the sendid field, the Processor MUST leave the sendid field (of _event) blank.	OK	
335	m	If an event was not received from an external entity, the Processor MUST leave the origin field blank.	OK	
336	m	For external events, the SCXML Processor SHOULD set the origintype field to a value which, in combination with the 'origin' field, will allow the receiver of the event to send a response back to the originating entity.	OK	
337	m	For internal and platform events, the Processor MUST leave the origintype field blank.	OK	
338	m	If an event is generated from an invoked child process, the Processor MUST set the invokeid field to the invoke id of the invocation that triggered the child process.	OK	
339	m	If an event is not generated from an invoked child process, the Processor MUST leave the invokeid field blank.	OK	
342	m	The SCXML Processor MUST set the name field (of the _event variable) to the name of the event.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
346	m	The Processor MUST place the error error.execution on the internal event queue when any attempt to change the value of a system variable is made.	OK	
172	m	If 'eventexpr' is present, the SCXML Processor MUST evaluate it when the parent send element is evaluated and treat the result if it had been entered the value of 'event'.	OK	
173	m	If 'targetexpr' is present, the SCXML Processor MUST evaluate it when the parent send element is evaluated and treat the result if it had been entered the value of 'target'.	OK	
174	m	If 'typexpr' is present, the SCXML Processor MUST evaluate it when the parent send element is evaluated and treat the result if it had been entered the value of 'type'.	OK	
175	m	If 'delayexpr' is present, the SCXML Processor MUST evaluate it when the parent send element is evaluated and treat the result if it had been entered the value of 'delay'.	OK	
176	m	The SCXML Processor MUST evaluate param when the parent send element is evaluated and pass the resulting data unmodified to the external service when the message is delivered	No	delay[2]
178	m	The SCXML Processor MUST include all attributes and values provided by param and/or 'namelist' even if duplicates occur.	OK	
179	m	The SCXML Processor MUST evaluate the content element when the parent send element is evaluated and pass the resulting data unmodified to the external service when the message is delivered.	OK , manually	delay[2]

Assertion Number	Conformance	Abstract	Test	Notiz
183	m	If 'idlocation' is present, the SCXML Processor MUST generate an id when the parent send element is evaluated and store it in this location	OK	
185	m	If a delay is specified via 'delay' or 'delayexpr', the SCXML Processor MUST interpret the character string a time interval. the SCXML Processor MUST dispatch the message only when the delay interval elapses.	OK	
186	m	The Processor MUST evaluate all arguments to send when the send element is evaluated, and not when the message is actually dispatched.	No	delay[2]
187	m	If the SCXML session terminates before the delay interval has elapsed, the SCXML Processor MUST discard the message without attempting to deliver it.	No	delay[2]
194	m	If the value of the 'target' or 'targetexpr' attribute is not supported or invalid, the Processor MUST place the error error.execution on the internal event queue	No	delay[2]
198	m	If neither the 'type' nor the 'typeexpr' is defined, the SCXML Processor MUST assume the default value of http://www.w3.org/TR/scxml/#SCXMLEventProcessor.	OK	
199	m	f the SCXML Processor does not support the type that is specified, it MUST place the event error.execution on the internal event queue.	OK	
200	m	SCXML Processors MUST support the type http://www.w3.org/TR/scxml/#SCXMLEventProcessor	OK	
201	o	Processors that support HTTP POST must use the value http://www.w3.org/TR/scxml/#BasicHTTPEventProcessor for the "typeättribute	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
205	m	The sending SCXML Interpreter MUST not alter the content of the send and include it in the message that it sends to the destination specified in the target attribute of send.	No	optional[1]
521	m	f the Processor cannot dispatch the event, it MUST place the error error.communication on the internal event queue of the session that attempted to send the event.	OK	
553	m	If the evaluation of send's arguments produces an error, If the evaluation of send's arguments produces an error, the Processor MUST discard the message without attempting to deliver it.	OK	
207	m	The SCXML Processor MUST NOT allow cancel to affect events that were not raised in the same session.	OK	
208	m	The Processor SHOULD make its best attempt to cancel all delayed events with the specified id.	No	cancel[3] delay[2]
210	m	If the 'sendidexpr' attribute is present, the SCXML Processor MUST evaluate it when the parent cancel element is evaluated and treat the result if it had been entered the value of 'sendid'.	No	cancel[3]
215	m	If the typeexpr attribute is present, the SCXML Processor MUST evaluate it when the parent invoke element is evaluated and treat the result if it had been entered the value of 'type'.	No	delay[2]
216	m	If the srcexpr attribute is present, the SCXML Processor MUST evaluate it when the parent invoke element is evaluated and treat the result if it had been entered the value of 'src'.	OK	cancel[3] delay[2]

Assertion Number	Conformance	Abstract	Test	Notiz
220	m	Platforms MUST support http://www.w3.org/TR/scxml/, a value for the 'type' attribute	OK	
223	m	If the 'idlocation' attribute is present, the SCXML Processor MUST generate an id automatically when the invoke element is evaluated and store it in the location specified by 'idlocation'.	OK	
224	m	When the platform generates an identifier for 'idlocation', the identifier MUST have the form stateid.platformid, where stateid is the id of the state containing this element and platformid is automatically generated.	OK	
225	m	n the automatically generated invoke identifier, platformid MUST be unique within the current session	OK	
226	m	When the invoke element is executed, the SCXML Processor MUST start a new logical instance of the external service specified in 'type' or 'typexpr', passing it the URL specified by 'src' or the data specified by content, or param.	OK	
228	m	the Processor MUST keep track of the unique invokeid and insure that it is included in all events that the invoked service returns to the invoking session.	OK	
229	m	When the 'autoforward' attribute is set to true, the SCXML Processor MUST send an exact copy of every external event it receives to the invoked process.	OK	
230	m	When the SCXML Processor autoforwards an event to the invoked process, all the fields specified in 5.11.1 The Internal Structure of Events MUST have the same values in the forwarded copy of the event	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
232	m	he invoked external service MAY return multiple events while it is processing	OK , manually	
233	m	If there is a finalize handler in the instance of invoke that created the service that generated the event, the SCXML Processor MUST execute the code in that finalize handler right before it removes the event from the event queue for processing.	OK	delay[2]
234	m	t MUST NOT execute the finalize handler in any other instance of invoke besides the one in the instance of invoke that created the service that generated the event.	OK	
235	m	Once the invoked external service has finished processing it MUST return a special event 'done.invoke.id' to the external event queue of the invoking process, where id is the invokeid for the corresponding invoke element.	OK	
236	m	The external service MUST NOT generate any other events after the invoke.done.invokeid event.	OK	
237	m	If the invoking session takes a transition out of the state containing the invoke before it receives the 'done.invoke.id' event, the SCXML Processor MUST automatically cancel the invoked component and stop its processing.	OK	
239	m	Invoked services of type http://www.w3.org/TR/scxml/, http://www.w3.org/TR/ccxml/, http://www.w3.org/TR/voicexml30/, or http://www.w3.org/TR/voicexml21 MUST interpret values specified by the content element or 'src' attribute markup to be executed	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
240	m	Invoked services of type http://www.w3.org/TR/scxml/, http://www.w3.org/TR/ccxml/, http://www.w3.org/TR/voicexml30/, or http://www.w3.org/TR/voicexml21 MUST interpret values specified by param element or 'namelist' attribute values that are to be injected into their data models	OK	
241	m	Invoked services MUST treat values specified by param and namelist identically.	OK	
242	m	Invoked services MUST also treat values specified by 'src' and content identically.	OK	
243	m	If the invoked process is of type http://www.w3.org/TR/scxml/ and 'name' of a param element in the invoke matches the 'id' of a data element in the top-level data declarations of the invoked session, the SCXML Processor MUST use the value of the param element the initial value of the corresponding data element.	OK	
244	m	If the invoked process is of type http://www.w3.org/TR/scxml/ and the key of namelist item in the invoke matches the 'id' of a data element in the top-level data declarations of the invoked session, the SCXML Processor MUST use the corresponding value the initial value of the corresponding data element.	OK	
245	m	If the invoked process is of type http://www.w3.org/TR/scxml/, and the name of a param element or the key of of a namelis item do not match the name of a data element in the invoked process, the Processor MUST NOT add the value of the param element or namelist key/value pair to the invoked session's data model.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
247	m	If the invoked state machine is of type http://www.w3.org/TR/scxml/ and it reaches a top-level final state, the Processor MUST place the event done.invoke.id on the external event queue of the invoking machine, where id is the invokeid for this invocation	OK	
250	m	When an invoked process of type http://www.w3.org/TR/scxml/is cancelled by the invoking process, the Processor MUST execute the onexit handlers for all active states in the invoked session	OK	
252	m	Once it cancels an invoked session, the Processor MUST NOT insert any events it receives from the invoked session into the external event queue of the invoking session.	OK	
253	m	When the invoked session is of type http://www.w3.org/TR/scxml/, The SCXML Processor MUST support the use of SCXML Event/IO processor (E.1 SCXML Event I/O Processor) to communicate between the invoking and the invoked sessions.	, manually	cancel[3]
530	m	The SCXML Processor MUST evaluate a child content element when the parent invoke element is evaluated and pass the resulting data to the invoked service.	OK , manually	cancel[3]
554	m	if the evaluation of the invoke element's arguments arguments produces an error, the SCXML Processor MUST terminate the processing of the element without further action.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
436	m	When the "datamodelättribute of the scxml element has the value "null", the In() predicate must return 'true' if and only if that state is in the current state configuration.	OK	
278	o	In the ECMAScript data model, the SCXML processor MUST allow any data element to be accessed from any state.	OK	
444	o	In the ECMAScript datamodel, for each data element in the document, the SCXML Processor must create an ECMAScript variable object whose name is the value of the id attribute of the data element.	OK	
445	o	In the ECMAScript datamodel for each data element in the document, if the variable object associated with the element is not assigned at the time indicated by the 'binding' attribute on the scxml element, then the SCXML Processor must assign the variable the default value ECMAScript undefined.	No	optional[1]
448	o	In the ECMAScript datamodel, the SCXML Processor must place all variables in a single global ECMAScript scope.	No	optional[1]
449	o	In the ECMAScript datamodel, the SCXML Processor must convert ECMAScript expressions used in conditional expressions into their effective boolean value using the ToBoolean operator described in Section 9.2 of [ECMASCRIPT-262].	No	optional[1]

Assertion Number	Conformance	Abstract	Test	Notiz
451	o	In the ECMAScript datamodel, the SCXML Processor must add an ECMAScript function 'In()' to the SCXML namespace that takes a stateID its argument and returns 'true' if and only if that state is in the current state configuration, described in 5.10.1 Conditional Expressions.	No	optional[1]
452	o	In the ECMAScript datamodel, the SCXML Processor must accept any ECMAScript left-hand-side expression a location expression.	No	optional[1]
453	o	In the ECMAScript datamodel, the SCXML Processor must accept any ECMAScript expression a value expression.	No	optional[1]
456	o	the SCXML Processor must accept any ECMAScript program defined in Section 14 of [ECMASCRIPT-262] the content of a script element.	No	optional[1]
446	o	In the ECMAScript data model, if either the 'src' attribute or in-line content is provided in the data elemenet, then if the content (whether fetched or provided in-line) is JSON and the processor supports JSON, the SCXML Processor MUST create the corresponding ECMAScript structure and assign it the value of the data element.	No	optional[1]
557	o	In the ECMAScript data model, if either the 'src' attribute or in-line content is provided in the data element, then if the content (whether fetched or provided in-line) is an XML document, the SCXML Processor MUST create the corresponding DOM structure and assign it the value of the data element.	No	optional[1]

Assertion Number	Conformance	Abstract	Test	Notiz
558	o	In the ECMAScript datamodel, if either the 'src' attribute or in-line content is provided in the data element, and the content (whether fetched or provided in-line) is not an XML document or JSON (or the processor does not support JSON), then the Processor MUST treat the content a space-normalized string literal and assign it the value of the variable	No	optional[1]
560	o	In the ECMAScript data model, if the content provided to populate _event.data can be interpeted key-value pairs, then for each unique key, the SCXML Processor MUST create a property of _event.data whose name is the name of the key-value pair and whose value is the value of the key-value pair.	No	optional[1]
578	o	In the ECMAScript data model, if the content provided to populate _event.data cannot be interpreted key value pairs and can be interpeted JSON and the processor supports JSON, then the Processor MUST create the corresponding ECMAScript structure(s) properties of _event.data.	No	optional[1]
561	o	In the ECMAScript data model, if the content provided to populate _event.data can be interpreted a valid XML document (and cannot be interpreted key-value pairs or JSON), the Processor MUST create the corresponding DOM structure and assign it the value _event.data.	No	optional[1]

Assertion Number	Conformance	Abstract	Test	Notiz
562	o	In the ECMAScript data model, if the content provided to populate _event.data is neither key-value pairs nor JSON nor a valid XML document, the Processor MUST treat the content treat the content a space-normalized string literal and assign it the value of _event.data.	No	optional[1]
569	o	For the _ioprocessors system variable in the ECMAScript datamodel the Processor MUST create an array with an object for each Event I/O processor that it supports, where the name of the object is the same that of the I/O processor. For the SCXML and BasicHTTP Event I/O processors, the Processor MUST create a location property under the object, assigning the access URI its String value.	No	optional[1]
457	o	In the ECMAScript data model, the legal iterable collections are arrays, namely objects that satisfy instanceof(Array) in ECMAScript. The legal values for the 'item' attribute on foreach are legal ECMAScript variable names.	No	optional[1]
459	o	In the ECMAScript data model, the iteration order for the foreach element is the order of the underlying ECMAScript array, and goes from an index of 0 by increments of one to an index of array_name.length - 1.	No	optional[1]
460	o	In the ECMAScript data model, since shallow copy is required for the foreach element, foreach assignment is equivalent to item = array_name[index] in ECMAScript.	No	optional[1]

Assertion Number	Conformance	Abstract	Test	Notiz
189	m	When using the scxml event i/o processor] If the target is the special term '#_internal', the Processor MUST add the event to the internal event queue of the sending session	No	optional[1]
190	m	[When using the scxml event i/o processor] If the target is the special term '#_scxml_sessionid', where sessionid is the id of an SCXML session that is accessible to the Processor, the Processor MUST add the event to the external queue of that session.	No	optional[1]
191	m	[When using the scxml event i/o processor] If the target is the special term '#_parent', the Processor MUST add the event to the external event queue of the SCXML session that invoked the sending session, if there is one.	OK	
192	m	[When using the scxml event i/o processor] If the target is the special term '#_invokeid', where invokeid is the invokeid of an SCXML session that the sending session has created by invoke, the Processor MUST must add the event to the external queue of that session.	OK	
193	o	[When using the scxml event i/o processor] If neither the 'target' nor the 'targetexpr' attribute is specified, the SCXML Processor MUST add the event to the external event queue of the sending session.	OK	
347	m	SCXML Processors MUST support sending messages to and receiving messages from other SCXML sessions using the SCXML Event I/O Processor.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
348	m	name'. The sending SCXML Processor MUST take the value of this attribute from the 'event' attribute of the send element. The receiving SCXML Processor MUST use it the value the 'name' field in the event that it generates.	OK	optional[1]
349	m	source'. The sending SCXML Processor MUST populate this attribute with a URI that the receiving processor can use to reply to the sending processor. The receiving SCXML Processor MUST use this URI the value of the 'origin' field in the event that it generates.	OK	
350	m	target'. The sending SCXML Processor MUST take the value of this attribute from the 'target' attribute of the send element. The receiving SCXML Processor MUST use this value to determine which session to deliver the message to.	OK	
351	m	sendid'. the sending SCXML Processor MUST populate this attribute with the identifier specified in the 'id' attribute or automatically generated by the platform when the send tag is executed in the sending session. (See 6.2 send.) The receiving SCXML Processor MUST use this value the value of the 'sendid' field in the event that it generates. If the author of the sending session did not specify either the 'id' or 'idlocation' attribute, the sending SCXML Processor MUST leave this attribute empty.	OK	

Assertion Number	Conformance	Abstract	Test	Notiz
352	m	'sourcetype'. The sending SCXML Processor MUST assign this attribute the value ßcxml". (Note that other types of senders will assign different values.) The receiving Processor MUST use this value the value of the 'origintype' field of the event that it generates.	OK	
354	m	The 'data' field of the event raised in the receiving session MUST contain a copy of the data specified in the 'namelist' attribute or in param or content elements in the sending session. The nature of the copy operation depends on the datamodel in question. However, the Processor MUST ensure that changes to the transmitted data in the receiving session do not affect the data in the sending session and vice-versa. The format of the 'data' field will depend on the datamodel of the receiving session.	OK	
495	m	If no errors occur, the receiving Processor MUST convert the message into an SCXML event, using the mapping defined above and insert it into the appropriate queue, defined in Send Targets.	OK	
496	m	If the sending SCXML session specifies a session that does not exist or is inaccessible, the SCXML Processor MUST place the error error.communication on the internal event queue of the sending session.	OK , manually	

Assertion Number	Conformance	Abstract	Test	Notiz
500	m	SCXML Processors that support the BasicHTTP Event I/O Processor MUST maintain a 'scxml' entry in the _ioprocessors system variable. The Processor MUST maintain a 'location' field inside this entry whose value holds an address that external entities can use to communicate with this SCXML session using the SCXML Event I/O Processor.	OK	
501	m	The 'location' field inside the entry for the SCXML Event I/O Processor in the _ioprocessors system variable MUST hold an address that external entities can use to communicate with this SCXML session using the SCXML Event I/O Processor.	OK	
509	o	An SCXML Processor that supports the Basic HTTP Event I/O Processor MUST accept messages at the access URI HTTP POST requests	OK	
510	o	The SCXML Processor MUST validate the message it receives [via the Basic HTTP Event I/O Processor] and then MUST build the appropriate SCXML event and MUST add it to the external event queue	OK	
513	o	After it adds the received message to the appropriate event queue, the SCXML Processor MUST then indicate the result to the external component via a success response code 2XX.	No	optional[1]
518	o	If the namelist attribute is defined [in send], the SCXML Processor MUST map its variable names and values to HTTP POST parameters	No	optional[1]

Assertion Number	Conformance	Abstract	Test	Notiz
519	o	If one or more param children are present [in send], the SCXML Processor MUST map their names (i.e. name attributes) and values to HTTP POST parameters	No	optional[1]
520	o	If a content child is present, the SCXML Processor MUST use its value the body of the message.	No	optional[1]
522	o	SCXML Processors that support the BasicHTTP Event I/O Processor MUST maintain a 'basichttp' entry in the _ioprocessors system variable. The Processor MUST maintain a in 'location' field inside this entry whose value holds an address that external entities can use to communicate with this SCXML session using the Basic HTTP Event I/O Processor.	No	optional[1]
531	o	If a single instance of the parameter '_scxmleventname' is present, the SCXML Processor MUST use its value the name of the SCXML event that it raises.	No	optional[1]
532	o	If _scxmleventname is not present, the Processor MUST the name of the HTTP method that was used to deliver the event name of the SCXML event that it raises	No	optional[1]
534	o	If the 'event' parameter of send is defined, the SCXML Processor MUST use its value the value of the HTTP POST parameter _scxmleventname	No	optional[1]
567	o	The processor MUST use any message content other than '_scxmleventname' to populate _event.data.	No	optional[1]
577	o	If neither the 'target' nor the 'targetexpr' attribute is specified, the SCXML Processor MUST add the event error.communication to the internal event queue of the sending session.	No	optional[1]

Informationen zu den Notizen

Nachfolgend Informationen zu den Testfällen, weitere Informationen befinden sich im Unterabschnitt 3.1

[1] Dieser Test war optional und musste daher nicht durchgeführt werden. Optionale Tests behandeln z.B. des ECMAScript Datenmodel oder die Unterstützung des BasicHTTP Event I/O Prozessor. Beide Elemente werden vom Interpreter nicht unterstützt.

[2] Dieser Test konnte aufgrund der fehlenden Implementierung des `delay` nicht durchgeführt werden. Da dieser Test Zeiteinschränkungen verwendet, kann er ohne dieses Element nicht durchgeführt werden.

[3] Dieser Test konnte aufgrund der fehlenden Implementierung des Elements `cancel` nicht durchgeführt werden. Die beiden Elemente `delay` und `cancel` gehören zusammen, aufgrund der fehlenden Umsetzung von delay, kann auch ein Test mit dem dazugehörigen Element `cancel` durchgeführt werden.

[4] Dieser Test konnte aufgrund der fehlenden Implementierung des Elements `script` nicht durchgeführt werden.

[5] Dieser Test konnte aufgrund der fehlenden Implementierung des Elements `foreach` nicht durchgeführt werden.

[6] Sollte während eines Blockes von ausführbaren Inhalten ein Fehler passieren, werden die restlichen Elemente trotzdem durchgeführt und nicht wie im Standard angegeben, ausgelassen.

[7] Kann eine Bedingung aufgrund eines Fehlers in der Bedingung nicht evaluiert werden, wird im Gegensatz zum Standard dieses Element nur als negativ evaluiert und kein Fehlerevent erzeugt.