Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2433

Analyse und Erweiterung eines bestehenden Choreographiewerkzeugs

Joas Schilling

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer/in:	M.Sc. Wirt.-Inf. Andreas Weiß
Beginn am:	10. Juli 2013
Beendet am:	9. Januar 2014
CR-Nummer:	D.1.7, D.2.11, H.4.1, I.3.4

Kurzfassung

Diese Arbeit befasst sich mit dem, von Oliver Sonnauer im Rahmen seiner Diplomarbeit entwickelten, BPEL4Chor Designer, zum Modellieren von Choreographien. Im ersten Schritt soll der Designer analysiert werden sowie fehlende Feature und andere Fehler aufgezeigt werden. Im weiteren Verlauf der Arbeit wird der Designer um fehlende Punkte erweitert. Dabei liegt der Fokus der Arbeit speziell auf dem Modellieren der Choreographie und weniger auf der Transformation zum Erstellen der BPEL4Chor Prozesse.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Verzeichnis der Abkürzungen

API – Application Programming Interface

BPEL – Web Service Business Process Execution Language

BPEL4Chor – BPEL for Choreographies

EMF – Eclipse Modeling Framework

GEF – Graphical Editing Framework

GMF – Graphical Modeling Framework

IDE – Integrated Development Environment

MVC – Model-View Controller

PBD – Participant Behaviour Description

SWT – Standard Widget Toolkit

UML – Unified Modeling Language

XMI – XML Metadata Interchange

XML Extensible Markup Language

XSD – XML Schema Definition

1 Einleitung

Das Bestellen eines Taxis ist heutzutage so einfach wie eh und je. Schnell die passende App auf dem Smartphone gestartet, Standort und Zielort angegeben und schon ist das Taxi auf dem Weg, um den Kunden abzuholen und an das gewünschte Ziel zu befördern. Was dabei im Hintergrund alles abläuft, bekommt der Kunde nicht zu sehen. Nur das Unternehmen selbst weiß, welche Prozesse dabei in Gang gesetzt werden müssen: Freie Taxis werden ermittelt, diese angefragt ob sie die Beförderung übernehmen möchten, die Informationen für die Beförderung gesammelt und dann an den Kunden zurück gesendet werden.

Soll dieser *Workflow* nun modelliert werden soll, muss für jeden der einzelnen Teilnehmer ein eigener *Business Prozess* erstellt werden. Wenn die einzelnen Prozesse modelliert werden, gehen jedoch die Verbindung der einzelnen Prozesse, die Kommunikation zwischen ihnen und andere Abhängigkeiten, die die Prozesse untereinander haben, verloren. Diesen Aspekt der Interaktion von Business Prozessen nennt man *Choreographie*. [W3C05]

Mit der Modellierung solcher Choreographien hat sich Oliver Sonnauer in seiner Diplomarbeit "Modellierung von Scientific Workflows mit Choreographien" beschäftigt. [Son13] Daraus entstand ein Choreographiewerkzeug, mit dem die einzelnen Prozesse und Choreographie-Aspekte nicht nur graphisch visualisiert werden, sondern auch als *BPEL4Chor*[1]-Fragmente exportiert werden können. Diese so gewonnen BPEL4Chor-Fragmente können dann später mit anderen Tools[2] zu vollständigen *BPEL-Prozessen* erweitert und dann verwendet werden.

Ziele dieser Arbeit

Mit Hilfe eines praxisorientierten Beispiels soll das, von Oliver Sonnauer im Rahmen seiner Diplomarbeit entwickelte, graphische Werkzeug zur Modellierung von Choreographien analysiert werden. Noch nicht implementierte Features sollen im Anschluss implementiert werden und sofern im zeitlichen Rahmen der Arbeit noch möglich ein Teil der in Abschnitt 3.2.2 aufgezeigten Fehler behoben werden. Am Ende wird das Beispiel noch einmal aufgegriffen, um die neuen Möglichkeiten der Modellierung zu zeigen.

[1] Eine Erweiterung für BPEL (Web Service Business Process Execution Language - Sprache zur Ausführung von Business Prozessen) zur Modellierung von Choreographien [DKLW07]
[2] Bspw. Eclipse BPEL Designer [EF14a]

Gliederung

Kapitel 2 – Grundlagen beschreibt die grundlegenden Technologien, die, zum Verständnis der Arbeit oder des Werkzeugs an sich, notwendig sind.

Kapitel 3 – Analyse des Choreographiewerkzeugs beschreibt den aktuellen Stand der Implementierung an Hand eines Beispiels und geht auf einige Details ein, die beim Implementieren der noch fehlenden Feature beachtet werden müssen.

Kapitel 4 – Implementierung der fehlenden Features beschreibt das genauere Vorgehen beim Implementieren der Features.

Kapitel 5 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und weist auf neu entstandene Problem hin, die Anknüpfungspunkte für weitere Arbeiten sein können.

2 Grundlagen

2.1 BPEL - Web Service Business Process Execution Language

BPEL, oder auch WS-BPEL genannt, steht für *Web Service Business Process Execution Language*. Die Sprache, deren Version 2.0 seit April 2007 als Standard definierte ist, ermöglicht es, das Verhalten von *Web Services* zu beschreiben. BPEL selbst ist dabei XML basiert. [OAS07b] Die Hauptelemente von BPEL sind dabei *Partner Links*, *Variables*, *Correlation Sets*, *Handlers* und *Activities*. [LK12]

Die Aktivitäten kann man hauptsächlich in zwei Gruppen aufteilen:

Basic Activities sind einfache Aktivitäten die nicht aus anderen Aktivitäten aufgebaut sind. Beispiele hierfür sind z.B. *Reply* und *Receive*, die zum Versenden und Empfangen von Nachrichten benutzt werden:

Listing 2.1 Einfaches Beispiel einer *Receive*-Aktivität in Anlehnung an [OAS07a]

```
<receive partnerLink="NCName" operation="NCName" />
```

Structured Activities dagegen sind solche Aktivitäten, die sich aus anderen Aktivitäten aufbauen. Ein Beispiel hierfür wäre die *If*-Aktivität, welche dazu benutzt werden kann, eine andere Aktivität nur unter bestimmten Bedingungen auszuführen.

Listing 2.2 Einfaches Beispiel einer *If*-Aktivität in Anlehnung an [OAS07a]

```
<if>
       <condition>$nochKeineAntwortBekommen</condition>
       <receive partnerLink="NCName" operation="NCName" />
</if>
```

2.2 BPEL4Chor - BPEL für Choreographien

Das Einbinden von Services in (BPEL)-Prozesse wird als *Composition* oder *Orchestrierung* bezeichnet. Unter einer *Choreographie* versteht man das Verbinden mehrerer Orchestrierungen. In BPEL4Chor, eine Erweiterung für BPEL, liegt das Augenmerk auf eben dieser Choreographie. Dabei wird darauf geachtet, dass die 3 Hauptpunkte einer Choreographie von einander getrennt bleiben: [DKLW07]

- In der *Participant Behaviour Description* geht es darum, festzulegen, wie der Daten-Kontrollfluss zwischen den kommunizierenden Prozessen und den einzelnen Aktivitäten verläuft.
- In der *Topology* werden die einzelnen *Message Links* (Nachrichten), aber auch die Choreographie-Teilnehmer an sich, definiert.
- Das *Grounding* ist dafür verantwortlich die technischen Details der Implementierung aufzunehmen. Dadurch wird erreicht, dass die einzelnen Details für die Kommunikation nicht direkt mit den Modellen der anderen zwei Punkte verbunden sind und somit schnell und einfach ersetzt werden können.

2.3 EMF - Eclipse Modeling Framework

Das *Eclipse Modeling Framework* (EMF) [EF14b] ist ein Framework für die bekannte Java IDE[1] Eclipse. Mit dessen Hilfe können Tools und Anwendungen gebaut werden, die auf strukturierten Datenmodellen basieren. Abhängig von den installierten Erweiterungen, kann das Datenmodell aus vielen verschiedenen Formaten[2] importiert werden, ansonsten muss es mit Hilfe einer einfachen Baumstruktur und einem Bereich für Eigenschaften (*Property Section*) manuell erstellt werden. Auf diese Art kann das Modell danach auch erweitert, vereinfacht und anderweitig bearbeitet werden. Das Modell wird am Ende als *XML Metadata Interchange* (XMI) Serialisierung, in einer `.ecore`-Datei gespeichert. [BSM+04] Aus diesem Modell können anschließend Java-Klassen für die weitere Verwendung erstellt werden.

2.4 GMF - Graphical Modeling Framework

Eine Möglichkeit mit dem Modell zu Arbeiten, bietet *Graphical Modeling Framework* (GMF) [EF14c], eines der *Graphical Modeling Project* von Eclipse. Es wurde dafür entwickelt, das EMF-Modell mit der *Model-View Controller* (MVC) Architektur des *Graphical Editing Framework* (GEF) von Eclipse zu verbinden. Dies hatten davor schon mehrere Projekte versucht umzusetzen. [BSM+04]

GMF besteht dabei aus zwei Hauptkomponenten: Einer *Tooling*-Komponente, mit deren Hilfe die graphischen Elemente definiert werden und mit dem zu Grunde liegenden Modell verknüpft werden. Die zweite Komponente ist die *runtime*-Komponente. Sie ist dafür verantwortlich EMF und GEF mit einander zu verbinden und bietet zusätzlich noch eine API[3] zum Entwickeln des graphischen Editors.

[1] Integrated Development Environment - Integrierte Entwicklungsumgebung, sie beinhalten meist: einen Editor für den Quellcode, einen Compiler zum Übersetzen des Programm Codes, verschiedene Werkzeuge zum Debuggen und leichteren Programmieren

[2] z.B. Unified Modeling Language Version 2 (UML2), XML Schema Definition (XSD) oder annotierten Java-Klassen

[3] Application Programming Interface

3 Analyse des Choreographiewerkzeugs

In diesem Kapitel wird ein genauerer Blick das von Oliver Sonnauer entwickelte Choreographiewerkzeug geworfen. [Son13] Dabei sollen vor allem fehlende Features und andere Fehler aufgezeigt werden, die im Bereich der Modellierung auftreten. Fehler bei der Transformation der Choreographie zu BPEL4Chor stehen nicht im Fokus dieser Arbeit.

3.1 Das Taxi Szenario - Version 1

Um die Funktionalität zu testen wird ein Muster-Szenario, angelehnt an das Taxi-Szenario aus [Hag11], erstellt. Dabei geht es um den Bestellvorgang für ein Taxi. Die Abläufe innerhalb des *Taxi-Unternehmens* samt *Taxis* und *Taxi-Service-Provider* sollen dabei mit Hilfe des Choreographiewerkzeuges dargestellt werden.

1. Zu Beginn des Vorgangs, sendet der Kunde seine Anfrage an das *Taxi-Unternehmen*.
2. Das *Taxi-Unternehmen* leitet die Anfrage an den *Taxi-Service-Provider* weiter und wartet dann auf die Transport-Informationen.
3. Nachdem der *Taxi-Service-Provider* die Anfrage vom *Taxi-Unternehmen* erhalten hat, erstellt er eine Liste aller freien *Taxis*, die in der Nähe des Kunden sind.
4. Anschließend fragt er die Kontaktdaten der Taxi-Fahrer ab und sendet die Transport-Anfrage an das jeweilige *Taxi*.
5. Jetzt wartet er auf eine Antwort durch die *Taxis*:
   * Wenn eine Antwort eintrifft, wird dem *Taxi* eine Bestätigung zu gesendet. Somit ist das *Taxi* gebucht.
   * Trifft jedoch keine Nachricht von einem *Taxi* ein, steht kein *Taxi* zur Verfügung und der Transport kann nicht durch geführt werden.
6. Der *Taxi-Service-Provider* sendet anschließend die Informationen an das *Taxi-Unternehmen*,
7. das zum Abschluss des Vorgangs den Kunden über den Transport informiert.

Dieser Ablauf kann so auch schon mit dem Werkzeug modelliert werden und ist in Abbildung 3.1 auf Seite 15 zu sehen. *Taxi-Unternehmen* und *Taxi-Service-Provider* werden als *Participant* in Form eines abgerundeten Rechtecks dargestellt. Die Taxis, deren genaue Anzahl unbekannt ist, werden als *ParticipantSet* dargestellt. Solche *Sets* haben zusätzlich eine gestrichelte Umrandung am unteren und rechten Rand, um die übereinander liegenden einzelnen Teilnehmer zu symbolisieren.

Die Kommunikation zwischen den einzelnen Teilnehmern wird immer durch einen beschrifteten Pfeil dargestellt, desen Spitze immer auf den Empfänger der Nachricht zeigt.

Aktivitäten die mit am Kontrollfluss oder an der Kommunikation beteiligt sind, z.B. *ForEach*, *Send* und *Receive*, werden ebenfalls angezeigt.

3.2 Fehlerübersicht

3.2.1 Fehlende Features

Während der Modellierung fällt auf, dass nicht alle in BPEL zur Verfügung stehenden Aktivitäten zur Verfügung stehen. Daher mussten im Taxi-Szenario einige Aspekte gekürzt werden um das Modell zu erstellen. Wie in der Diplomarbeit [Son13] selbst schon erwähnt, fehlen:

- `<while>`
- `<if>`
- `<assign>`
- `<repeatUntil>`

Von denen im BPEL Designer als *Actions* und *Controls* bekannten Aktivitäten fehlen außerdem noch, die für die Modellierung einer Choreographie jedoch weniger erforderlich sind, als die zuvor genannten Aktivitäten:

- `<empty>`
- `<validate>`
- `<wait>`

Des Weiteren fehlen alle BPEL-Aktivitäten, die zur Fehlerbehandlung dienen:

- `<catch>`
- `<catchAll>`
- `<exit>`
- `<throw>`
- `<rethrow>`
- `<compensate>`
- `<compensateScope>`

Mit diesen Aktivitäten können Prozesse beendet, Fehler geworfen und abgefangen werden. Auch diese Aspekte sind zwar wichtig, um die volle Funktionalität zu unterstützen, haben aber nicht die höchste Priorität beim Erstellen einer Choreographie.

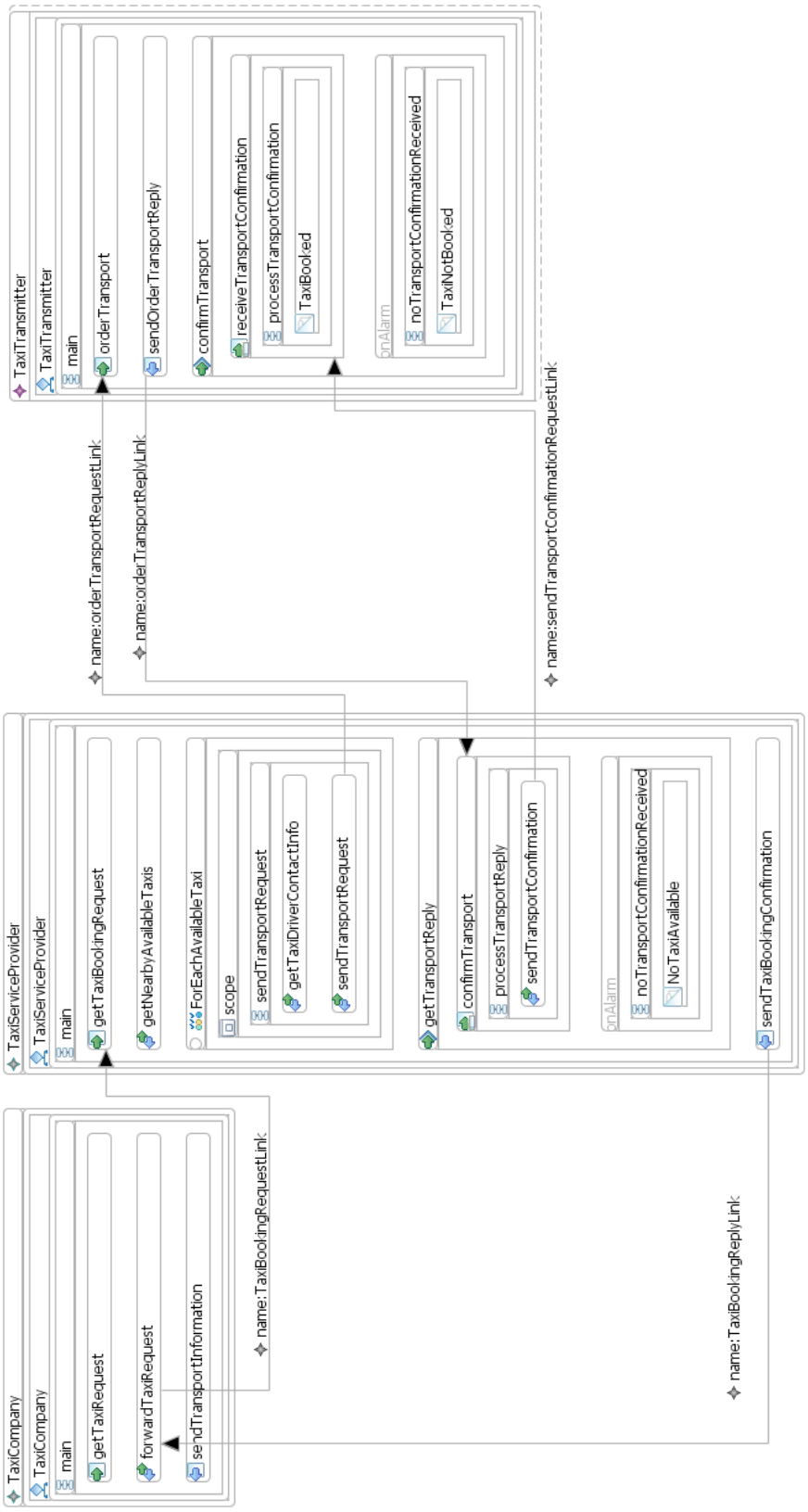

Abbildung 3.1: Erste Version der Taxi App

3.2.2 Andere Fehler

Zusätzlich zu den fehlenden Features sind mehrere Fehler auffällig, die teilweise auch schon in einem technischen Report, über das Choreographiewerkzeug, veröffentlicht wurden. [WAS+13] Einige davon sind lediglich optisch unangenehm:

- Werden zwei *Flow*-Aktivitäten in einander eingefügt, wird die Darstellung des Choreographiewerkzeugs beschädigt. Alle Elemente haben einen zusätzlichen Innenabstand zu ihrer Umrandung. Die Darstellung ist auch in allen anderen geöffneten Choreographie-Fenstern defekt. Erst wenn die verschachtelten *Flow*-Elemente wieder gelöscht werden und dann die Eclipse Instanz neu gestartet wurde, wird der Editor wieder korrekt dargestellt.
- Die einzelnen *OnMessage*-Abschnitte der *Pick*-Aktivität werden nicht, wie vom Eclipse BPEL Designer bekannt, nebeneinander, sondern untereinander dargestellt. Bereits kurze Prozesse können dadurch nicht mehr vollständig auf einem Bildschirm angezeigt werden.
- Alle *MessageLinks* tragen den Text "name:" vor ihrem Label, der nicht gelöscht werden kann.

Weitere Fehler sind zum Beispiel:

- Wenn zwei Aktivitäten aus zwei verschiedenen *Participants* mit einem *MessageLink* verbunden werden und dann einer der *Participants* gelöscht wird, kann das Choreographie-Diagramm nicht mehr gespeichert werden. Grund dafür ist, dass die angelegten *MessageLinks* nicht gelöscht werden und dann auf nicht mehr existierende Elemente zeigen. Wenn die Aktivität anstelle des *Participant* gelöscht wird, tritt der Fehler nicht auf.
- Wird im Nachhinein die *Receive Activity* oder *Send Activity* eines *MessageLinks* über die zugehörige *Property Section* geändert, wird der Link in der graphischen Darstellung nicht aktualisiert und verbindet immer noch die ursprünglichen Elemente. Erst wenn das *Diagramm* geschlossen und neu geöffnet wurde, werden die richtigen Elemente miteinander verbunden.

3.3 Vorüberlegungen zu den fehlenden Features

3.3.1 Sonderzeichen im *Condition*-Element der *If* und *While*-Aktivitäten

Beim Erstellen von `If.condition`, `Elseif.condition` und `While.condition` ist Vorsicht geboten. Um normale Vergleiche wie "größer" (>) und "kleiner" (<) zu Ermöglichen, muss der Inhalt des Elements besonders gespeichert werden, ansonsten wird das XML-Schema der erzeugten Dateien zerstört. Ein Beispiel dafür wäre der Vergleich "`$test.Kleiner < $test.Vergleich`". Durch das Kleinerzeichen wird ein neues XML-Element geöffnet, das nicht mehr vor dem schließenden `</condition>`-Tag geschlossen wird:

Listing 3.1 Fehler im XML-Schema durch die Verwendung eines Kleinerzeichens

```
<while>
       <condition>$test.Kleiner < $test.Vergleich</condition>
                          <!-- ^ Fehler -->
       ...
</while>
```

3.3.2 Varianten der *Copy*-Operation

In BPEL stehen mehrere Möglichkeiten zur Verfügungen, durch die Daten von verschiedenen Elementen und Abschnitten kopiert werden können, als auch wohin sie gespeichert werden können.[1] Als Quelle (`<from>`) stehen dabei zur Verfügung:

- Ein *Query* oder eine Expression in einer angegebenen Sprache,
- ein Punkt der an einem Kommunikations-Kanal beteiligt ist (*Endpoint Reference*),
- eine *Variable* oder die *Property* einer Variable,
- ein *Literal* oder ein leeres Elemente (`<from />`).

Für das Ziel des Kopiervorgangs (`<to>`) steht dabei lediglich die *Literal*-Variante nicht zur Verfügung. Während der Implementierung wird lediglich die "*von Variable zu Variable*"-Variante erläutert. Die Implementierung der anderen Varianten weicht nämlich nur geringfügig von dieser einen Variante ab.

[1] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html#SA00032

4 Implementierung der fehlenden Features

Im Folgenden wird zuerst die Implementierung der <while>-Aktivität genauer erklärt. Bei den weiteren Features wird nur noch auf mögliche Besonderheiten eingegangen.

4.1 Implementierung der <while>-Aktivität

Listing 4.1 zeigt den Aufbau einer <while>-Aktivität in BPEL.

Listing 4.1 Aufbau einer <while>-Aktivität in Anlehnung an [OAS07a]

```
<while>
       <condition><!-- Boolescher Ausdruck --></condition>
       <!-- Auszufuehrende Aktivitaet -->
</while>
```

Um die Aktivität zu implementieren, müssen mehrere Teilaufgaben erledigt werden:

1. Die Spezifikation der Aktivität muss im EMF-Modell der "*Participant Behaviour Description*" `pbd.ecore` angegeben werden.
2. Das Werkzeug zum Einfügen des *While*'s muss zur Palette hinzugefügt werden.
3. Die graphischen Darstellung für die Figur muss festgelegt werden.
4. Das Werkzeug, die Darstellung und die Spezifikation müssen verbunden werden.
5. Einer Option für die Angabe der *While*-Bedingung muss eingerichtet werden.
6. Es muss ermöglicht werden, andere Aktivitäten innerhalb von *While* aufrufen zu können.

4.1.1 Spezifikation der <while>-Aktivität

In der Spezifikation wird festgelegt, wie die *While*-Aktivität aufgebaut ist und das sie vom Typ *Activity* ist. Auch die ihr zugehörige Aktivität `While.activity` ist vom Typ *Activity*. Das heißt alle Elemente die ebenfalls vom Typ *Activitiy* sind, können für `While.activity` eingesetzt werden. Beispiele dafür wären *Sequence* und *Reply*, aber auch *While* selbst. Auf die gleiche Weise kann *While* auch in alle anderen Elemente eingefügt werden, die ein Element vom Typ *Activity* erwarten, zum Beispiel *Sequence* und das noch nicht implementierte *If*.

Die Bedingung `While.condition` für die Aktivität ist vom Typ *Condition*, ein Untertyp von *Expression*. Der Inhalt des <condition>-Element muss später, wie bereits in den Vorüberlegungen in Abschnitt 3.3.1 angekündigt, in einem `CDATA`-Abschnitt gespeichert werden.

Dem XML-Parser wird dadurch mit geteilt, dass der folgende Abschnitt nur als normaler Text betrachtet werden soll. Es können also auch Sonderzeichen genutzt werden, die ansonsten Probleme im XML-Dokument verursachen, für Vergleiche in den angegebenen Bedingungen aber unerlässlich sind. Die wichtigsten Zeichen wären dabei das Kleiner- (<) und Größerzeichen (>), die in XML-Dokumenten Elemente öffnen und schließen, aber auch das Anführungszeichen (") könnte ansonsten Problem hervorrufen.

Die korrekte XML-Code des *While*'s aus Listing 3.1 sieht damit wie folgt aus:

Listing 4.2 Verwendung eines Kleinerzeichens innerhalb eines CDATA-Abschnitts

```
<while>
       <condition><![CDATA[$test.Kleiner < $test.Vergleich]]></condition>
       ...
</while>
```

4.1.2 Option in der Werkzeugpalette

Als zweiter Schritt wird die Option implementiert, mit deren Hilfe der Benutzer später die Aktivität zur Arbeitsfläche hinzufügen kann. Dafür muss das "*Tooling Definition Model*" (`chor.gmftool`) angepasst werden. Im XML Code dieser Datei, siehe Listing 4.3, wird dabei ein neues `<tools>`-Element angelegt. Als untergeordnete Elemente werden dabei ein kleines und ein grosses Icon angegeben, die später, zusammen mit dem Titel und der Beschreibung, das Werkzeug darstellen.

Listing 4.3 Tools-Element-Eintrag für die *While*-Aktivität

```
<tools
    xsi:type="gmftool:CreationTool"
    title="While"
    description="Create new While">
  <smallIcon ... />
  <largeIcon ... />
</tools>
```

Wenn nun die `chor.gmfmap` Map aktualisiert, anschließend daraus das `chor.gmfgen` Modell erzeugt und schließlich der Diagramm-Code neu erstellt wird, ist in der graphischen Oberfläche zum Modellieren der Choreographie das *While*-Werkzeug zu sehen.

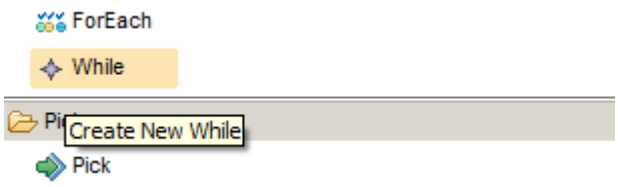

Abbildung 4.1: *While*-Werkzeug zum Einfügen der Aktivität in die graphische Benutzeroberfläche

4.1.3 Graphische Darstellung

Allerdings lässt sich das *While* bisher noch nicht in den *Editor*, die Arbeitsfläche, einfügen. Dafür muss zuerst in der `chor.gmfgraph` definiert werden, aus welchen graphischen Elementen die *While*-Figur besteht. Vom Knoten selbst, über die Beschriftung der Option, bis zur Darstellung der untergeordneten Aktivität kann alles genau definiert werden. In diesem Fall wurde für die Figur selbst ein abgerundetes Rechteck ausgewählt ähnlich wie bei den bereits existierenden Figuren, siehe dazu die folgende Abbildung 4.2.

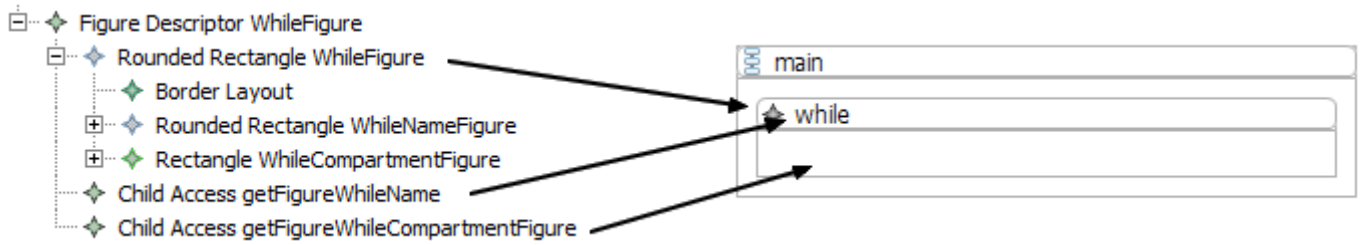

Abbildung 4.2: Links: "*While*-Figure Descriptor", rechts: "*While*-Figur" des Editors

In der `chor.gmfgraph` werden ebenfalls noch die Zugriffspunkte definiert, durch die später der Name und die Kind-Aktivität passend eingefügt werden können:

Listing 4.4 Weitere Definitionen für die *While*-Figur in der `chor.gmfgraph`

```
<nodes
    name="While"
    figure="WhileFigure"/>
      ...
<compartments
    name="WhileActivityCompartment"
    figure="WhileFigure"
    accessor="//@figures.1/@descriptors.13/@accessors.1"/>
      ...
<labels
    name="WhileName"
    figure="WhileFigure"
    accessor="//@figures.1/@descriptors.13/@accessors.0"/>
```

4.1.4 Verknüpfung von Werkzeug, Darstellung und Spezifikation

Als letzter Schritt müssen im *Mapping Model* `chor.gmfmap` die Darstellung aus dem *Graphical Definition Model* und das neue Werkzeug aus dem *Tooling Definition Model*, zusammen mit dem *Domain Model* `chor.ecore` verknüpft werden. Das *Mapping Model* greift dabei auf das *Participant Behaviour Description* Modell `pbd.ecore` zurück, in dem die einzelnen Aspekte der *While*-Aktivität bereits in Abschnitt 4.1.1 definiert wurden.

Zunächst wird die untergeordnete Kind-Aktivität ignoriert und die Aktivität lediglich als mögliche *Child Reference* der *Sequence*-Aktivität definiert, wie in Abbildung 4.3 zu sehen ist.

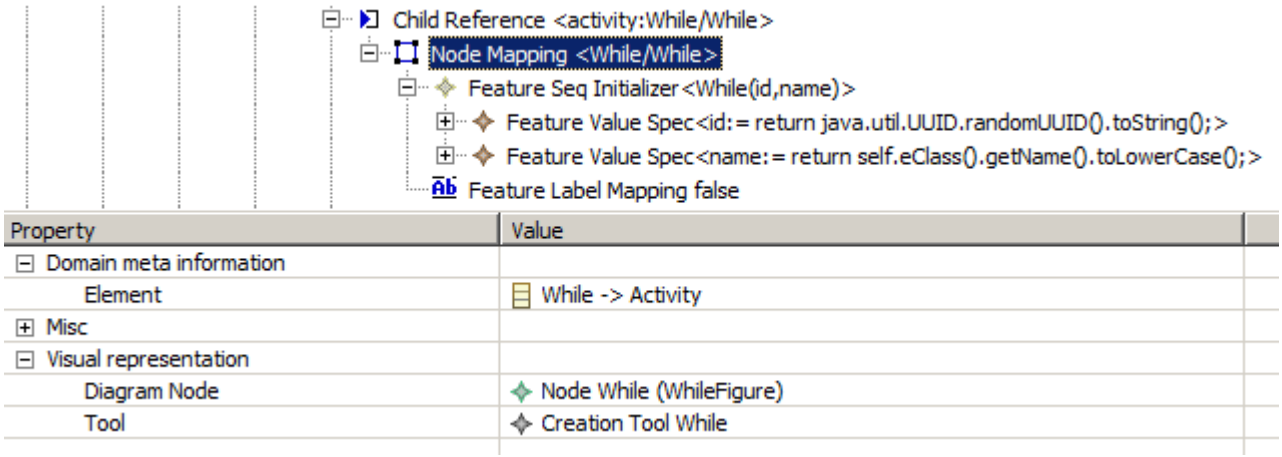

Abbildung 4.3: *While* als mögliches Kind für *Sequence*

Wärend eine *While*-Instanz erstellt wird, werden zwei Werte für das Element gesetzt: `id`, zur eindeutigen Identifizierung des Elements, und `name`, der im Editor angezeigt wird.

Wenn nun wieder, wie in Abschnitt 4.1.2, das `chor.gmfgen` Modell und anschließend der Diagramm-Code neu erstellt werden, kann die *While*-Aktivität in den Editor eingefügt werden. Sieht man sich anschließend das chor-Modell des Diagramm's an, sieht man dort die *While*-Aktivität als XML-Element:

Listing 4.5 *While*-Aktivität im chor-Diagramm Modell

```
<activity xsi:type="pbd:While" name="while" id="2e6247d9-0897-4ec5-ba19-4ec7906ce71e">
  <condition body="ACED0005740008746573742O3C2033"/>
</activity>
```

Anschließend kann über die Option "ChorDiagramEditor" > "Export to BPEL4Chor" der BPEL4Chor Prozess aus dem Diagramm exportiert werden, wo sich die *While*-Aktivität dann auch zum jetzigen Zeitpunkt schon wieder findet:

Listing 4.6 *While*-Aktivität im BPEL4Chor-Dokument

```
<while name="while" wsu:Id="2e6247d9-0897-4ec5-ba19-4ec7906ce71e">
  <condition><![CDATA[$test <= 10]]></condition>
</while>
```

4.1.5 Definitionsmöglichkeit für die Bedingung

Als Nächsten soll nun ermöglicht werden, die Bedingung des *While*-Element's, in einem neuen Tab der *Property View*, anzugeben. In der `plugin.xml` wird hierfür zu Beginn der Tab definiert:

Listing 4.7 Definition des *Condition* Tab's

```
<propertyTab
      category="extra"
      id="property.tab.ConditionTab"
      label="Condition">
</propertyTab>
```

Als zweites wird dann die *Property Section* definiert:

Listing 4.8 Definition der *Property Section*

```
<propertySection
  class="org.eclipse.bpel4chor.property.tabs.sections.WhileConditionSection"
  enablesFor="1"
  filter="org.eclipse.bpel4chor.property.tabs.sections.filter.WhileTypeFilter"
  id="property.section.WhileConditionSection"
  tab="property.tab.ConditionTab">
    <input
      type="org.eclipse.bpel4chor.model.chor.diagram.navigator.ChorAbstractNavigatorItem">
    </input>
</propertySection>
```

Dabei wird ein "Filter" angegeben, der dafür sorgt, dass die *Property Section* nur für *While*-Elemente erstellt wird. Das zweite wichtige Attribut ist die angebene "Klasse" `class`, die sich darum kümmert, dass der in der *Textarea* eingebene Text im zugeordneten Element gespeichert und im Falle einer späteren Änderung auch wieder davon geladen wird. Der erstellte Tab und die zugehörige *Property Section* sind in der folgenden Abbildung zu sehen:

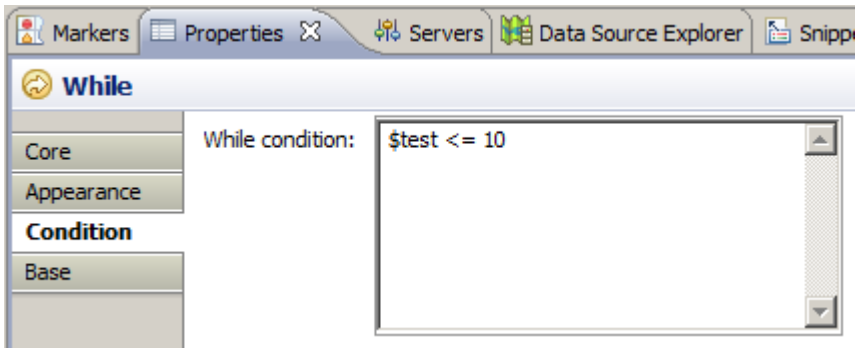

Abbildung 4.4: *Condition*-Tab in den *Properties* eines *While*-Elements

In Abschnitt 4.3.4 auf Seite 30 wird genauer darauf eingegangen, wie eine solche *Property Section* funktioniert.

4.1.6 Weitere Aktivitäten als Eltern- und Kindelement von `<while>`

Zum Abschluss der Implementierung des *While*-Features muss ermöglicht werden, andere Aktivitäten als Kind von *While* anzugeben. Dafür wird im *Compartment Mapping* angegeben, in welchem Abschnitt der *While*-Figur die Kinder angezeigt werden. Danach muss für jede mögliche Aktivität eine *Child Reference* definiert werden, die angibt, welches *Feature* das jeweilige Kind implementiert, in welchem *Compartment* es angezeigt werden soll und um welches Kind es sich dabei handelt.

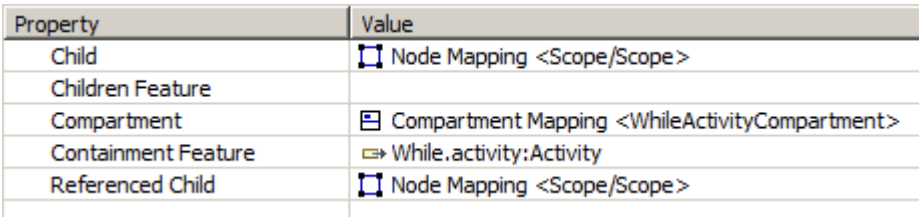

Abbildung 4.5: *Scope - Child Reference* für die *While*-Aktivität. Diese *Child Reference* ermöglicht das Einfügen von *Scope* in ein *While*.

Da bisher keine anderen Aktivitäten, abgesehen von *Sequence*, existieren, die eine *While*-Aktivität als Kind akzeptieren, ist die Implementierung damit abgeschlossen. Wenn solche *Structured Activities*[1] bereits implementiert wären, müssten auch *Child References* (Abbildung 4.5) dafür angelegt werden, um *While* innerhalb solcher Aktivitäten nutzen zu können.

[1] vgl. Abschnitt 2.1 auf Seite 11

4.2 Implementierung der <if>-Aktivität

Listing 4.9 Aufbau einer <if>-Aktivität in Anlehnung an [OAS07a]

```
<if>
        <condition><!-- Boolescher Ausdruck --></condition>
        <!-- Auszufuehrende Aktivitaet -->

        <elseif>*
                <condition><!-- Boolescher Ausdruck --></condition>
                <!-- Auszufuehrende Aktivitaet -->
        </elseif>

        <else>?
                <!-- Auszufuehrende Aktivitaet -->
        </else>
</if>
```

4.2.1 Spezifikation

Der Basisaufbau der *If*-Aktivität ähnelt sehr dem der *While*-Aktivität, innerhalb des Elements ist zuerst eine *Bedingung* und dann eine auszuführende *Aktivität*, siehe hierfür Listing 4.9.

Zusätzlich dazu können im Falle des *If*'s noch beliebig viele *Elseif*-Elemente und ein optionales *Else*-Element folgen. Die *Elseif*-Elemente bestehen immer aus einer *Bedingung* und einer *Aktivität*, wie auch schon *If* und *While*. Das *Else*-Element hingegen enthält nur eine Aktivität. Es wird ausgeführt, wenn weder das *If* noch eines der *Elseif*-Elementen erfüllt wurde.

Wie auch schon bei der *While*-Aktivität müssen die *Condition*-Elemente auch bei *If* und *Elseif* als <![CDATA[ ]]>-Abschnitte gespeichert werden um Probleme mit Sonderzeichen zu vermeiden.

4.2.2 Option in der Werkzeugpalette

Beim Einfügen der Optionen in die Werkzeugpalette wird nur ein Eintrag erstellt. Später werden alle drei Elemente (*If*, *Elseif* und *Else*) mit dieser Option verbunden, da eindeutig ist, an welche Stelle ein Element eingefügt werden muss. Außerhalb von <if>-Elementen können nur <if>-Elemente eingefügt werden. Das *If* selbst wiederum besteht aus 3 Figurteilen. Wird ein <if> innerhalb des <if>s eingefügt, so wird das neue *If* als `If.acitivity` in den ersten Abschnitt eingefügt. Mögliche <elseif>-Elemente kommen in den zweiten Abschnitt, das optionale <else>-Element in den Dritten. Wählt man die Option aus und versucht sie in den Editor einzufügen, bietet GMF die, an dieser Stelle noch zur Verfügung stehenden, Optionen in einem kleinem Dropdown an.

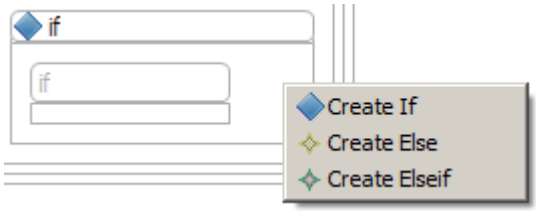

Abbildung 4.6: Dropdown zum Einfügen der verschiedenen *If*-Optionen

4.2.3 Graphische Darstellung

Wie bereits erwähnt, benötigt die *If*-Figur, im Gegensatz zur Figur von *While*, drei Abschnitte, in die Kinder eingefügt werden können. Die Abschnitte für *Elseif* und *Else* sind jedoch am Anfang nicht sichtbar.

Abbildung 4.7: Neu erstelltes *If* ohne Unterelemente

Erst wenn über die Werkzeugpalette *Elseif* und *Else* selbst eingefügt werden, sind sie auch im Editor zusehen. Siehe dazu Abbildung 4.8

Abbildung 4.8: *If* mit *Elseif* und *Else* als Unterelementen

Die Aktivitäten der einzelnen Elemente werden anschließend in die zugehörigen untergeordneten Rechtecke der jeweiligen *If*, *Elseif* und *Else* Figur eingefügt.

Abbildung 4.9: *If* mit *Elseif* und *Else* als Unterelementen und eingefügten Aktivitäten

4.2.4 Mapping-Eintrag für *If*

Der Eintrag für das *If* im *Mapping Model* `chor.gmfmap` ist um einiges Umfangreicher, als der Eintrag des *While*'s. Zunächst muss wie auch schon beim *While* das Werkzeug zusammen mit der Figur und dem *Domain* Element verbunden werden. Danach werden ebenfalls die *Child References* für die jeweiligen Elemente als `If.activity` angelegt. Beim *If* fehlen nun aber noch *Elseif* und *Else*.

Für diese beiden Elemente muss jeweils wieder ein *Compartment Mapping* angegeben werden, dann der Aufbau der jeweiligen Elements und die zugehörigen *Child References*. Wichtig hierbei ist, dass für *Elseif* und *Else*, im Gegensatz zu *If*, keine *Child Reference* für die anderen Elemente, die Aktivitäten enthalten, erstellt werden. `<elseif>` und `<else>` können also nur innerhalb von `<if>` genutzt werden und nicht direkt als Kind von `<while>` auftreten.

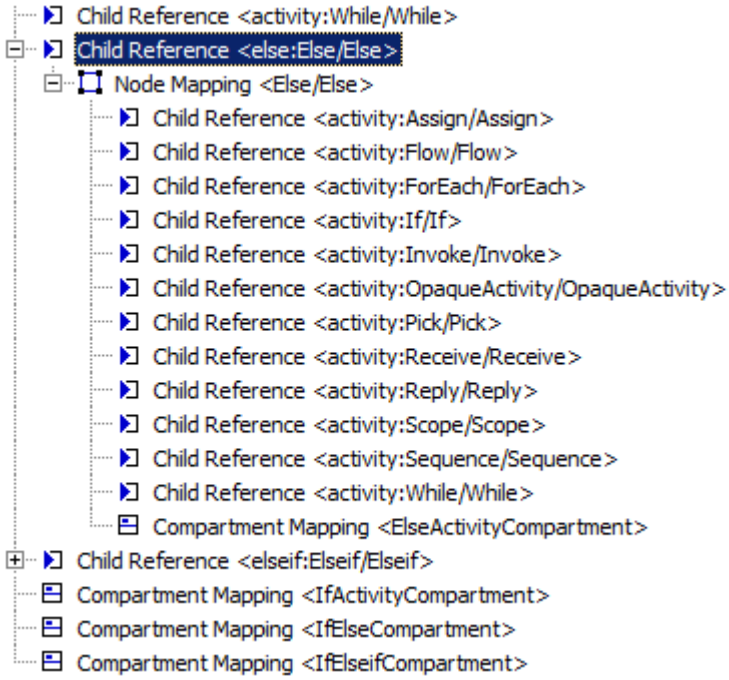

Abbildung 4.10: Definition für die *Child Reference Else*

4.2.5 *Chor Diagram*- und *BPEL4Chor*-Einträge für *If*

Dadurch, das beim Erstellen eines *If*'s nicht gleich *Elseif* und *Else* mit erstellt und eingefügt werden kann, wird der *Diagram* Code und auch der daraus erzeugte *BPEL4Chor* Code so einfach wie möglich gehalten. Siehe hierzu die zwei folgenden Listings die das *If* nur mit *Condition* und einer Aktivität, also ohne *Elseif* und *Else*, zeigen:

Listing 4.10 *If*-Aktivität in der *Chor Diagram* Datei ohne *Elseif* und *Else* aus Abbildung 4.7 auf Seite 26

```
<activity xsi:type="pbd:If" name="if" id="b53c08f7-d1e5-4190-b079-1b2c888b693f">
  <condition body="ACED0005740005656D707479"/>
  <activity xsi:type="pbd:Receive" name="receive"
      id="811c162d-8d68-45ea-ad80-48ea93f39826"/>
</activity>
```

Listing 4.11 *If*-Aktivität in der exportierten *BPEL4Chor*-Datei ohne *Elseif* und *Else* aus Abbildung 4.7

```
<if name="if" wsu:Id="b53c08f7-d1e5-4190-b079-1b2c888b693f">
  <condition><![CDATA[$test = 1]]></condition>
  <receive name="receive" wsu:Id="811c162d-8d68-45ea-ad80-48ea93f39826"/>
</if>
```

Wird noch ein *Elseif* und ein *Else* angegeben, kann die volle *If*-Funktionalität gezeigt und genutzt werden:

Listing 4.12 *If*-Aktivität in der *Chor Diagram* Datei mit *Elseif* und *Else* und Kind-Aktivitäten aus Abbildung 4.9 auf Seite 26

```
<activity xsi:type="pbd:If" name="if" id="b53c08f7-d1e5-4190-b079-1b2c888b693f">
  <condition body="ACED000574000A6E6F74456D7074794966"/>
  <activity xsi:type="pbd:Receive" name="receive"
      id="811c162d-8d68-45ea-ad80-48ea93f39826"/>
  <elseif>
    <condition body="ACED000574000E6E6F74456D707479456C73656966"/>
    <activity xsi:type="pbd:Receive" name="receive"
        id="81b39750-51d8-4f96-b068-144f12fee0c6"/>
  </elseif>
  <else>
    <activity xsi:type="pbd:Receive" name="receive"
        id="74baea27-3378-45af-8510-b8e2a1721db7"/>
  </else>
</activity>
```

Listing 4.13 *If*-Aktivität in der exportierten *BPEL4Chor*-Datei mit *Elseif* und *Else* und Kind-Aktivitäten aus Abbildung 4.9

```
<if name="if" wsu:Id="b53c08f7-d1e5-4190-b079-1b2c888b693f">
  <condition><![CDATA[$notEmptyIf]]></condition>
  <receive name="receive" wsu:Id="811c162d-8d68-45ea-ad80-48ea93f39826"/>
  <elseif>
    <condition><![CDATA[$notEmptyElseif]]></condition>
    <receive name="receive" wsu:Id="81b39750-51d8-4f96-b068-144f12fee0c6"/>
  </elseif>
  <else>
    <receive name="receive" wsu:Id="74baea27-3378-45af-8510-b8e2a1721db7"/>
  </else>
</if>
```

4.2.6 *Condition*-Option für *If* und *Elseif*

Die Eingabe der *Condition* für die *If*-Aktivität und das *Elseif*-Elemente erfolgt, genau wie bei der *While*-Aktivität, über einen zusätzlichen Tab in der *Property View*. Der Tab aus Listing 4.7 auf Seite 23 kann dabei wieder verwendet werden. Die Property Section hingegen muss zweimal neu definiert werden, um den Filter und die Klasse entsprechend für das *If* und *Elseif* anzupassen.

Damit ist nun auch die Implementierung von *If*, *Elseif* und *Else* abgeschlossen.

4.3 Implementierung der <assign>-Aktivität

Listing 4.14 Aufbau einer <assign>-Aktivität in Anlehnung an [OAS07a]

```
<assign validate="yes|no"?>
  <copy ignoreMissingFromData="yes|no"? keepSrcElementName="yes|no"?>
    <from variable="<!-- Variablenname -->"/>
    <to variable="<!-- Variablenname -->"/>
  </copy>+
</assign>
```

4.3.1 Spezifikation

Die *Assign*-Aktivität wird benutzt um Wert von einer Variable in eine Andere zu kopieren. Das optionale Attribut `validate`, mit dem Standardwert `no`, ermöglicht es, mit Hilfe der XML-Definition zu überprüfen, ob der neue Inhalt in der Zielvariable gespeichert werden darf. Wenn dies nicht der Fall ist, wird ein Fehler geworfen und alle Ziel-Elemente müssen den Startwert von Beginn der *Assign*-Aktivität annehmen – das *Assign* verhält sich also atomar. Das heißt, wenn nicht alle Kopiervorgänge ausgeführt werden können, wird kein Vorgang ausgeführt.

Die zwei booleschen Attribute `ignoreMissingFromData` und `keepSrcElementName` für das *Copy*-Element sind ebenfalls optional. Der Wert ist standardmässig für beide Attribute `no`. Im Falle von `ignoreMissingFromData` werden mögliche Fehlermeldungen unterdrückt, wenn das zu kopierende Element nicht existiert. Mit Hilfe von `keepSrcElementName` kann der Name des Ziel-Elements überschrieben werden, er wird dann mit dem Namen des zu kopierenden Element's ersetzt.

4.3.2 Option in der Werkzeugpalette

Wie auch schon bei der *If*-Implementierung werden *Assign* und *Copy* mit dem gleichen Werkzeug eingefügt. Wie auch schon beim *If* ergibt sich aus dem Kontext, ob an der Stelle ein *Assign* oder ein *Copy* eingefügt werden soll, da *Copy*-Elemente nur innerhalb von *Assign*-Aktivitäten auftreten können, gleichzeitig aber die *Assign*-Aktivität nur *Copy*-Elemente als Unterelemente haben kann.

4.3.3 Graphische Darstellung

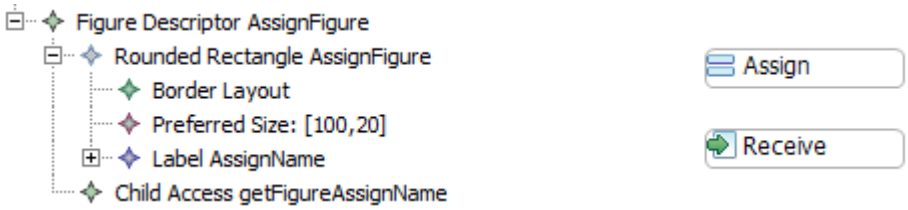

Abbildung 4.11: *Assign*-Figurbeschreibung links; rechts: Gegenüberstellung von *Assign* und *Receive*

Die Darstellung der *Assign*-Aktivität ist stark an die der *Receive*-Aktivität angelehnt. Es handelt sich dabei um ein einfaches abgerundetes Rechteck. Die *Copy*-Elemente finden in der graphischen Darstellung keine Repräsentation. Die Optionen der *Copy*-Elemente können über das *Assign*-Element angegeben und geändert werden.

4.3.4 Erklärung der *Property Section*

Die *Property Sections* für das *Assign*-Element und die ihm untergeordneten *Copy*-Elemente sind etwas aufwendiger, als die *Property Section* von *While* und *If*, weshalb an dieser Stelle genauer darauf eingegangen wird. Für das *Copy* werden zwei Eingabemöglichkeiten für die Ausgangs- und Zielvariable benötigt. Außerdem sollen die 2 optionalen Attribute `ignoreMissingFromData` und `keepSrcElementName` über Checkboxen angesteuert werden können, siehe dazu Abbildung 4.12:

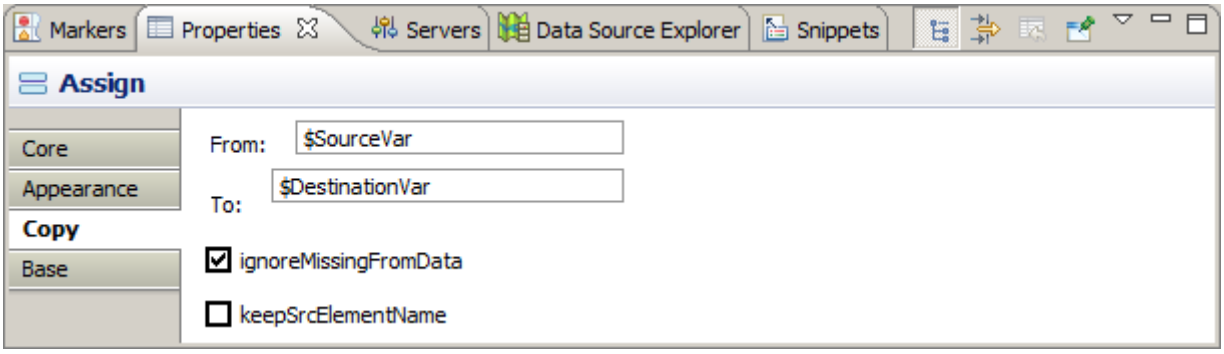

Abbildung 4.12: *Property Section* für ein *Copy*-Element

Dafür müssen in der zugehörigen `AssignCopySection.java` zunächst die Kontroll-Elemente erstellt werden, siehe Listing 4.15. Die hierfür benutzten *Buttons*, *Labels* und *Textareas* sind aus der bekannten Java-Bibliothek *Standard Widget Toolkit* (SWT)[2].

Listing 4.15 Erstellung der Kontroll-Elemente

```
public void createControls(Composite parent, TabbedPropertySheetPage
    aTabbedPropertySheetPage)
{
            ...
            assignCopyFromLabel = getWidgetFactory().createCLabel(composite, "From:");
            assignCopyFromTextarea = getWidgetFactory().createText(composite, "");
            ...
            ignoreMissingFromDataCheckBox = getWidgetFactory().createButton(composite,
                   PbdPackage.eINSTANCE.getCopy_IgnoreMissingFromData().getName(),
                       SWT.CHECK);
            ...
            configureLayout();
}
```

In der aufgerufenen Methode `configureLayout()` (Listing 4.16) wird anschließend festgelegt, wie die einzelnen Elemente auf der Fläche angeordnet sind, an welchen Elementen sie sich ausrichten und wie viel Abstand sie zu einander haben:

[2] http://www.eclipse.org/swt/

Listing 4.16 Konfiguration des Layout's

```
private void configureLayout()
{
    FormData formData = new FormData();
    formData.left = new FormAttachment(0,ITabbedPropertyConstants.HSPACE);
    formData.top = new FormAttachment(0,ITabbedPropertyConstants.VSPACE);
    assignCopyFromLabel.setLayoutData(formData);

    formData = new FormData();
    formData.left = new
        FormAttachment(assignCopyFromLabel,ITabbedPropertyConstants.HSPACE);
    formData.right = new FormAttachment(20,ITabbedPropertyConstants.HSPACE);
    formData.top = new FormAttachment(0,ITabbedPropertyConstants.VSPACE);
    assignCopyFromTextarea.setLayoutData(formData);
    ...
}
```

Bis zum jetzigen Zeitpunkt ist aber alles noch ohne Funktion. Daher muss als letzter Schritt festgelegt werden, dass, bei einer Texteingabe oder einem Klicken auf die Checkbox, die Daten aus der Anzeige auch in das Chor Diagramm und Chor Modell übernommen werden. Dafür wird, beim Laden der *Property Section*, der Wert aus dem Element ausgelesen und in das SWT-Element eingefügt. Siehe Listing 4.17

Listing 4.17 Laden der Werte aus dem Modell in die SWT-Elemente

```
public void refresh()
{
    ...
    assignCopyFromTextarea.setEnabled(true);
    assignCopyFromTextarea.setText("");
    ignoreMissingFromDataCheckBox.setSelection(false);
    ...
    if (assign.getCopy() != null)
    {
        if (assign.getCopy().get(0) != null)
        {
            if (assign.getCopy().get(0).getFrom() != null)
            {
                 if (assign.getCopy().get(0).getFrom().getVariable() != null)
                 {
                    assignCopyFromTextarea.setText(
                        assign.getCopy().get(0).getFrom().getVariable().toString()
                    );
                }
            }
            ...

            if (assign.getCopy().get(0).getIgnoreMissingFromData() != null &&
                 assign.getCopy().get(0).getIgnoreMissingFromData().equals(Boolean.YES))
            {
                ignoreMissingFromDataCheckBox.setSelection(true);
            }
            ...
        }
    }
    addListeners();
}
```

Mit `addListeners()` werden an die jeweiligen Elemente *Listener* hinzugefügt. Diese *Listeners* werden anschließend bei jedem Tastendruck aufgerufen und übernehmen die Änderungen aus den `SWT`-Elementen zurück in die `AssignCopySection` Klasse. Von dort aus werden die Werte dann beim Speichern in die *Choreographie*-Datei `default.chor` kopiert:

Listing 4.18 *Assign*-Aktivität mit *Copy*-Element und den angegebenen Attributen im Chor Modell

```
<activity xsi:type="pbd:Assign" name="assign"
    id="dde8812b-9732-4a0d-ac2d-fa60cd4c62a6">
  <copy ignoreMissingFromData="yes">
    <from variable="$SourceVar"/>
    <to variable="$DestinationVar"/>
  </copy>
</activity>
```

Wenn nun über "*Export as BPEL4Chor*"die `BPEL4Chor`-Datei erstellt wird, sind dort die in der *Property Section* angegebenen Werte zu finden: in diesem Fall die Namen der zu kopierenden Variable "`$SourceVar`" und der Zielvariable "`$DestinationVar`".

Die angewählte Checkbox "`ignoreMissingFromData`" tritt in Form des gleichnamigen Attributes mit dem Wert "yes" auf. Die nicht ausgewählte Option "`keepSrcElementName`" wird nicht ausgegeben, siehe Listing 4.19, da "no" der Standardwert für das Attribut ist.

Listing 4.19 *Assign*-Aktivität mit *Copy*-Element und den angegebenen Attributen im exportierten BPEL4Chor-Prozess

```
<assign name="assign" wsu:Id="dde8812b-9732-4a0d-ac2d-fa60cd4c62a6">
  <copy ignoreMissingFromData="yes">
    <from variable="$SourceVar"/>
    <to variable="$DestinationVar"/>
  </copy>
</assign>
```

Damit ist auch die Implementierung der *Assign*-Aktivität abgeschlossen.

5 Zusammenfassung und Ausblick

Mit den jetzt neu zur Verfügung stehenden Aktivitäten, kann die TaxiApp aus Abschnitt 3.1 erweitert werden.

5.1 Das Taxi Szenario erweitert - Version 2

Ein Beispiel dafür wäre Punkt 3, an dem die freien Taxis der Umgebung abgefragt werden:

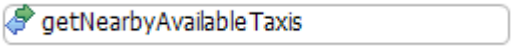

Abbildung 5.1: Abfrage "freier Taxis aus der Nähe" aus TaxiApp Version 1

Wenn keine Taxis in der Nähe frei sind, können zum Beispiel freie Taxis aus der weiteren Umgebung angefragt werden. Sind auch in dieser Kategorie keine Taxis verfügbar, kann entschieden werden solange zu warten, bis ein Taxi frei wird.

Die von den jeweiligen Möglichkeiten zurück gegebenen Taxis werden dann in eine `Taxis` Variable kopiert, damit für den folgenden Verlauf über die gleiche Variable iteriert werden kann. Für diese Änderungen werden allerdings alle 3 neu implementierten Aktivitäten (*If*, *While* und *Assign*) benötigt. Die vollständige geänderte Version der TaxiApp ist in Abbildung 5.3 auf der nächsten Seite zu sehen. Der geänderte Abschnitt des *Taxi-Service-Providers* sieht dabei wie folgt aus:

Abbildung 5.2: Abfrage "freier Taxis" aus TaxiApp Version 2

Abbildung 5.3: Zweite Version der Taxi App

Ähnliche Änderungen können auch an anderen Stellen vorgenommen werden. Wenn zum Beispiel am Ende kein Taxi den Transport bestätigt, kann ein anderer *MessageLink* genutzt werden, um einfacher zwischen einer erfolgreichen und fehlgeschlagenen Anfrage zu unterscheiden.

5.2 Neue entstandene Fehler

Während der Implementierung der Features wurden noch zusätzliche Fehler festgestellt. Einzelne Punkte können zusätzlich geändert werden um dem Benutzer die Bedienung zu erleichtern.

5.2.1 Darstellungsprobleme nach dem Einfügen von *Flow* innerhalb von *While*

Für die Implementierung der *While*-Aktivität wurde die Figur der *Flow*-Aktivität wieder verwendet. Der graphische Fehler der beim Verschachteln von mehreren *Flow*-Aktivitäten auftritt, erscheint daher auch, wenn ein *Flow* innerhalb einer *While*-Aktivität eingefügt wird. Siehe dazu die folgende Abbildung:

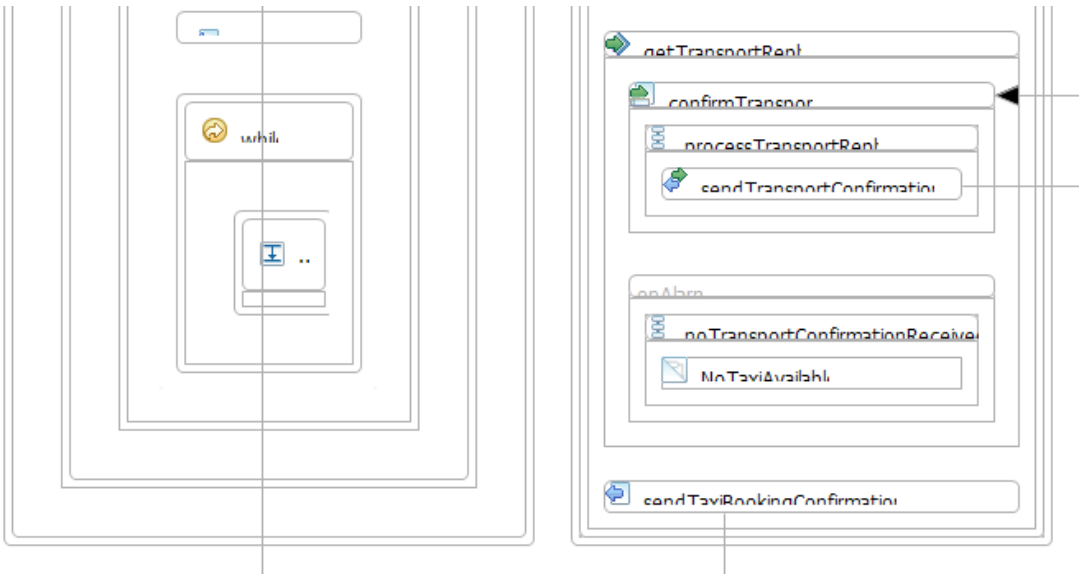

Abbildung 5.4: Darstellungsprobleme nach dem Einfügen von *Flow* innerhalb einer *While*-Aktivität

Der Fehler kann verhindert werden, wenn die *Flow*-Aktivität nicht direkt in das *While* eingefügt wird, sondern z.B. innerhalb einer *Sequence*:

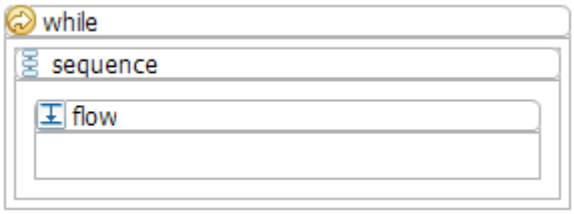

Abbildung 5.5: "Workaround" für das Darstellungsproblem von verschachtelten *Flow* und *While*-Aktivitäten

5.2.2 Nur ein *Copy* pro *Assign* möglich

Die derzeitige Implementierung von *Assign* erlaubt es nicht, mehrere *Copy*-Elemente in ein *Assign*-Element einzufügen. Wird über die Choreographie-Datei selbst die Änderung vorgenommen, so kann immer nur das erste *Copy*-Element in der *Property Section* des *Assign*-Elements bearbeitet werden.

5.2.3 *Elseif*-Elemente werden nicht nebeneinander dargestellt

Wie auch schon die *OnMessage*-Elemente, werden auch die einem *If* untergeordneten *Elseif*-Elemente nicht nebeneinander, sondern untereinander dargestellt:

Abbildung 5.6: *If*-Aktivität mit mehreren *Elseif*-Elementen

5.2.4 Einfügen von Elementen über das Kontextmenü

Der letzte Punkt an dieser Stelle betrifft die Benutzerfreundlichkeit beim Modellieren der Choreographie. Im Gegensatz zum BPEL Designer können im Choreographiewerkzeug untergeordnete Elemente nicht über das Kontextmenü (Rechtsklick mit der Maus auf das jeweilige Element) eingefügt werden. Dies sollte sowohl bei *Assign* > *Copy* als auch *If* > *Elseif* und *If* > *Else* ermöglicht werden. Dadurch wird auch das parallele Arbeiten in BPEL Designer und Choreographiewerkzeug angenehmer, weil der Benutzer nicht zwei verschiedene Arbeitsweisen beherrschen muss.

Literaturverzeichnis

[BSM+04] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose. *Eclipse Modeling Framework*. Addison-Wesley, 2004. (Zitiert auf Seite 12)

[DKLW07] G. Decker, O. Kopp, F. Leymann, M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *Proceedings of the IEEE 2007 International Conference on Web Services (ICWS)*. 2007. URL http://bpt.hpi.uni-potsdam.de/pub/Public/GeroDecker/icws2007-BPEL4Chor.pdf. (Zitiert auf den Seiten 9 und 11)

[EF14a] T. Eclipse-Foundation. Eclipse - BPEL Designer Project, 2014. URL http://www.eclipse.org/bpel/. (Zitiert auf Seite 9)

[EF14b] T. Eclipse-Foundation. Eclipse Modeling Framework (EMF), 2014. URL http://projects.eclipse.org/projects/modeling.emf. (Zitiert auf Seite 12)

[EF14c] T. Eclipse-Foundation. Graphical Modeling Project, 2014. URL http://projects.eclipse.org/projects/modeling.gmp. (Zitiert auf Seite 12)

[Hag11] R. Hagin. *Enabling integration and aggregation of context information into WS-BPEL processes*. Diplomarbeit, Universitaet Stuttgart, 2011. URL http://elib.uni-stuttgart.de/opus/volltexte/2011/6623. (Zitiert auf Seite 13)

[LK12] F. Leymann, D. Karastoyanova. Chapter 12: BPEL. Vorlesungsunterlagen von Services and Service Composition, 2012. (Zitiert auf Seite 11)

[OAS07a] OASIS. Web Services Business Process Execution Language Version 2.0, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. (Zitiert auf den Seiten 7, 11, 19, 25 und 29)

[OAS07b] OASIS. Web Services Business Process Execution Language Version 2.0 - Specification, 2007. URL http://docs.oasis-open.org/wsbpel/2.0/OS/process/abstract/ws-bpel_abstract_common_base.xsd. (Zitiert auf Seite 11)

[Son13] O. Sonnauer. *Modellierung von Scientific Workflows mit Choreographien*. Diplomarbeit, Universitaet Stuttgart, 2013. URL http://elib.uni-stuttgart.de/opus/volltexte/2013/8504/. (Zitiert auf den Seiten 9, 13 und 14)

[W3C05] W3C. Web Services Choreography Description Language Version 1.0, 2005. URL http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/. (Zitiert auf Seite 9)

[WAS+13] A. Weiss, V. Andrikopoulos, S. G. Saez, D. Karastoyanova, K. Vukojevic-Haupt. Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies. Technischer Bericht, Institut fuer Architektur von Anwendungssystemen - Universitaet Stuttgart, 2013. (Zitiert auf Seite 16)

Alle URLs wurden zuletzt am 07. 01. 2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift