Institut für Architektur von Anwendungssystemen (IAAS)
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart

Fachstudie Nr. 48

Evaluierung verschiedener Szenarien zur Anwendung von BPEL und WS-CDL

Ingo Hildebrandt, Marc Wagner, Carsten Stützner

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Stefan Pottinger
begonnen am:	04. Juli 2005
beendet am:	03. Oktober 2005
CR-Klassifikation	D2.2, D2.3, H5.2, J.1, K4.4

Abstract

Die Web Services Choreography Description Language bietet eine neue Möglichkeit, Geschäftsprozesse zwischen einzelnen Personen aus einem globalen Blickwinkel gesehen, zu gestalten. Bereits bekannte Sprachen wie BPEL sind bereits weit verbreitet und stellen ebenfalls Möglichkeiten zur Verfügung, Prozesse formal festzulegen. Dieses Dokument beschäftigt sich mit dem Vergleich beider Konzepte richtet sein Augenmerk vor allem/primär auf die Anwendbarkeit und Möglichkeiten von WS-CDL und die im Moment verfügbaren Tools, die man beim Einsatz nutzen kann.

Zunächst gibt es eine Einführung zum Thema WebServices, die eine Übersicht über verschiedene Vorgehensweisen/Konzepte gibt. Darauf folgend wird näher auf die zu untersuchenden Sprachen BPEL und WS-CDL eingegangen. In beiden Kapiteln gibt es jeweils einen Überblick über grundlegende Eigenschaften der Sprachen und laufende Aktivitäten. Kernkonstrukte werden erläutert um die Struktur eines BPEL beziehungsweise WS-CDL Dokuments verständlich zu machen. Im Abschnitt, der sich mit WS-CDL beschäftigt, werden Stärken und Schwächen von WS-CDL aufgezeigt und auf dem Markt verfügbare WS-CDL Tools werden hinsichtlich Benutzbarkeit, Vor- und Nachteilen, Installation und Erfahrungen im Umgang untersucht. Abschließend wird eine Empfehlung für ein Entwicklungswerkzeug gegeben. Die Beschreibung der untersuchten Szenarien und die sich dabei ergebende Eignung für eine Sprache bildet den Inhalt von Kapitel 5. Jedes Szenario wird natürlichsprachlich formuliert und als Diagramm in UML dargestellt. Die Umsetzung in BPEL und WS-CDL ist ebenso Bestandteil wie die auftretenden Unterschiede

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

XML und Web Services als De-facto-Standards sind aus der Geschäftswelt heute nicht mehr wegzudenken wenn es um die Beschreibung von plattformneutralen Anwendungen geht. Sie ermöglichen mehr denn je die Entwicklung von neuen Geschäftsvorgängen. Jahrelang arbeiteten Unternehmen daran, die Zusammenarbeit zwischen Geschäftspartnern im eigenem Geschäftsbereich, oder darüber hinaus, zu automatisieren um die Produktivität zu steigern und Kosten zu sparen. Bereits jetzt bieten die verschiedenen Web Service Spezifikationen Möglichkeiten, Web Services über Systemgrenzen hinaus zu verbinden und somit geeignete Applikationen zu schaffen. Die Zukunft im E-Business verlangt die Fähigkeit, langfristige Zusammenarbeit zwischen Geschäftspartnern sicherzustellen. Dies ist allerdings nur möglich, wenn klare Schnittstellen definiert sind auf die man, unabhängig von der eingesetzten Technik und der spezifischen Implementierung, aufbauen kann. Hier setzen Sprachen wie BPEL und WS-CDL an. Ihr Ziel ist die Betrachtung der Anbindung von verschiedenen Web Services untereinander und dem Nachrichtenaustausch zwischen den Beteiligten in Geschäftsprozessen. Speziell WS-CDL legt die Schwerpunkte hier auf den Informationsaustausch zwischen zusammenarbeitenden Parteien, ungeachtet der eingesetzten Plattform, des von der Implementierung genutzten Programmiermodells und der Zielumgebung.

In dieser Fachstudie sollen die Funktionalität beider Sprachen verglichen werden und anhand verschiedener Szenarien die Anwendbarkeit evaluiert werden, wobei der Hauptfokus auf WS-CDL liegt. Des weiteren werden Tools zur Bearbeitung von WS-CDL Dokumenten untersucht und deren Vor- und Nachteile bewertet, sowie eine Empfehlung für das bis jetzt beste Tool zu vergeben.

Die mit dem Institut für Architektur von Anwendungssystemen vereinbarte ausführliche Aufgabenstellung ist im nächsten Kapitel aufgeführt.

1.1 Aufgabenstellung

Analyse und Evaluation von Web Services Choreography Description Language (WSCDL) im Vergleich zu Business Process Execution Language for Web Services (BPEL4WS) in verschiedenen Anwendungsszenarien. Die Szenarien sollen sich mit Business-to-Business (B2B) Prozessen beschäftigen, also mit Flows, die partnerübergreifend stattfinden. Nach einer Einarbeitungsphase in WSCDL und BPEL4WS sollen verschiedene konkrete Anwendungsszenarien für Web Services erarbeitet werden, in denen deutlich wird, wann welcher Standard zur Beschreibung der Geschäftsprozesse und der Kommunikation zwischen den Businesspartnern besser geeignet ist. Als Orientierungshilfe wurden die folgenden Szenariotypen besprochen:

1. Partnerinteraktion innerhalb eines Unternehmens (nur ein interner Prozess bzw. mehrere interne Prozesse in gleichen bzw. verschiedenen Unternehmensbereichen – Fokus auf die internen Strukturen (Prozessen)).
2. Nur Partnerinteraktion zwischen verschiedenen Unternehmen(steilen) (ohne Kenntnis der internen Strukturen (Prozesse)).
3. BPEL4WS zur Implementierung von RosettaNet.

Weitere Aufgaben:

- Es soll überprüft werden, ob bereits Tools für WS-CDL verfügbar sind, welchen Entwicklungsstand diese haben und ob diese sinnvoll eingesetzt werden können. Diese(s) Tool(s) soll(en) anhand der vorher erstellten Szenarien erprobt und dabei analysiert werden. Falls es mehrere Tools gibt, sollen diese nach verschiedenen Kriterien bewertet werden (diese werden im Verlauf der Fachstudie noch festgelegt werden (funktionale Anforderungen: Standardkonformität, etc. und nicht-funktionale Anforderungen: Bedienbarkeit, etc.)).
- Eine weitere Aufgabe eine Auflistung der Alleinstellungsmerkmale der beiden Sprachen und, falls vorhanden, das Auffinden und direkte Gegenüberstellen von ähnlichen Funktionen in WS-CDL und BPEL4WS.
- Als Abschluss der Untersuchung sollen die Anwendungsbereiche der beiden Sprachen klar voneinander abgegrenzt werden, es sollen sinnvolle Anwendungsszenarien für WS-CDL hervorgehoben werden und falls Tools verfügbar sind, soll eine Empfehlung für ein oder mehrere Tools erfolgen.

1.1.1 Bearbeiter

Ingo Hildebrandt, info@dvservice.info
Carsten Stützner, stuetzner@gmx.de
Marc Wagner, mw@wagnerwebworks.de

1.1.2 Betreuer

Prof. Dr. Frank Leymann, frank.leymann@informatik.uni-stuttgart.de
Stefan Pottinger, stefan.pottinger@informatik.uni-stuttgart.de
Institut für Architektur von Anwendungssystemen (IAAS)

1.1.3 Zeitraum

04. Juli 2005 - 03. Oktober 2005

2 Definitionen

2.0.4 Orchestration

Definition übersetzt aus Web Services Glossary [10]: Eine **Orchestration** definiert die Sequenz und die Bedingungen, in welcher ein Web Service einen anderen Web Service aufruft, um eine nützliche Funktion zu realisieren. z.B.: Eine **Orchestration** ist das Muster der Interaktionen, denen ein Web Service Agent folgen muss, um sein Ziel zu erreichen.

Adaptiert aus WS-CDL-Barros et al [7]: Eine Orchestration beschreibt sowohl die Kommunikation als auch die **internen Aktionen** des bzw. der Prozesse(s). Weiterhin kann eine Orchestration

auch Kommunikationsaktionen bzw. Abhängigkeiten zwischen Kommunikationsaktionen enthalten, die nicht in einem **behavioral interface** auftauchen. Dies kann immer dann der Fall sein, wenn behavioral interfaces einer externen Partei zugänglich gemacht wurden. In diesem Fall sollen nur Information gezeigt werden, die diese Partei benötigt.

Orchestrations sind auch bekannt unter dem Namen **executable processes**, da ihr Zweck darin besteht, von einer Orchestration Engine ausgeführt zu werden.

Interne Aktionen

Unter dem Begriff „Interne Aktionen“ sind Datentransformationen und Aufrufbeziehungen von interner Software (z.B. Aufruf von Software, die intern benutzt wird, aber nach außen nicht als (Web) Service zugreifbar sind) subsummiert [7].

2.0.5 Behavioral Interface

Adaptiert aus WS-CDL-Barros et al [7]: Ein **behavioral interface**-Modell umfasst die Verhaltensweisen der Interaktionen durch welche ein spezieller Service ein Ziel erreichen kann. Es ergänzt **structural interface** Beschreibungen (wie von WSDL unterstützt).

Ein **behavioral interface** zeichnet die Abhängigkeiten zwischen Interationen auf:

1. Kontroll-Fluß-Abhängigkeiten („control-flow dependencies“, z.B. eine Interaktion muss vor einer anderen stattfinden)
2. Daten-Fluß-Abhängigkeiten („data-flow dependencies“)
3. Zeitabhängigkeiten („time-constraints“)
4. Nachrichtenkorrelation („message-correlation“)
5. Transaktionsabhängigkeiten („transaction dependencies“)
6. etc.

Im Gegensatz zur Choreography beschränkt sich ein behavioral interface auf die Perspektive einer teilnehmenden Partei. Ein behavioral interface umfasst also keine komplette Interaktion (dies würde eine globale Sicht auf beide Parteien erfordern), sondern kann viel mehr als Ansammlung von Kommunikationsaktionen eines Teilnehmers gesehen werden. Wie Choreographies beschreiben behavioral interfaces keine internen Abläufe (wie z.B. interne Datentransformation).

2.0.6 Choreography

Definition übersetzt aus Web Services Glossary [10]:

1. Eine **Choreography** definiert die Sequenz und die Bedingungen unter denen mehrere kooperierende unabhängige Agenten Nachrichten austauschen, um eine Aufgabe auszuführen und dadurch ein Ziel zu erreichen.

2. Web Services Choreography betrifft die Interaktion von Services mit deren Benutzern. Jeder Benutzer eines Web Service ist ein Client dieses Services. Diese Benutzer können ihrerseits andere Web Services, Applikationen oder Menschen sein. Transaktionen zwischen Web Services und deren Clients müssen zur Laufzeit klar definiert sein und können aus mehreren verschiedenen Interaktionen bestehen, deren Zusammensetzung ein vollständige Transaktion ergibt. Diese Komposition, ihre Nachrichtenprotokolle, Interfaces, Sequenzierung und assoziierte Logik wird als **Choreography** betrachtet.

3 BPEL4WS

3.1 Entstehung und Geschichte

Die Entstehung der Business Process Execution Language for Web Services (im weiteren Dokument als BPEL4WS bezeichnet) geht auf die Zusammenführung zweier Standards (XLANG und WSFL) zurück. Im Jahr 2002 wurde von IBM, Microsoft und BEA die Spezifikation in der Version 1.0 veröffentlicht.

Davor wurde von Microsoft die Spezifikation der kalkülbasierten Sprache XLANG für den Microsoft BizTalk Server entwickelt. Diese Sprache beinhaltet vor allem die Ausführungskontrolle von Geschäftsprozessen und die Kontrolle von Interaktionen und Nachrichten zwischen Serviceprovidern. Der Kontrollfluss kann mit Hilfe der folgenden Elemente strukturiert werden: Sequentielle Abfolge, Parallelausführung, Schleifen, Bedingten Verzweigungen und zeit- oder eventbasierten Triggern

Von IBM wurde die Sprache WSFL (Web Service Flow Language) spezifiziert, die den Kontrollfluss der Workflow-Produkte der MQ-Serie von IBM abbildet. Dabei werden zwei Arten der Zusammenstellung und Anordnung von Web Services beschrieben. Einmal die flow models, die die richtige Anordnung der Web Services zum Erreichen von speziellen (Teil-) Geschäftszielen beschreiben, und dann die global models, die die Interaktion der beteiligten Web Services untereinander beschreiben. WSFL erlaubt dabei auf der Basis von XML die Abbildung von ausführbaren Strukturen in gerichteten Graphen, wobei Aktivitäten von Kontroll- und Datenflüssen verbunden werden. Ein Alleinstellungsmerkmal von WSFL ist die "Dead-Path-Eliminationßur Beseitigung von SSackgassenïn der Struktur.

Im Moment liegt BPEL4WS als Spezifikation in der Version 1.1 vom April 2003 vor, bei der auch SAP und Siebel Systems mitgewirkt haben. Diese Version wurde an OASIS (Organization for the Advancement of Structured Information Standards) zur Standardisierung übergeben.

In Bearbeitung ist dabei die 2.0-Release, die jedoch inkompatibel zur Version 1.1 ist. Im September 2004 wurde die Umbenennung in WS-BPEL 2.0 beschlossen, um die Nähe zu den anderen Standards rund um Web Services wie z.B. WS-Adressing, WS-Security, etc. zu zeigen. Ein erster Arbeitsentwurf der neuen Release wurde im April 2004 veröffentlicht.

3.2 Grundlagen

BPEL4WS ist wie seine beiden Vorgänger eine Sprache, die das Verhalten von Geschäftsprozessen beschreiben soll. Als Grundlage werden dazu Web Services verwendet, weiterhin basiert BPEL4WS auf den Standards XMLSchema, XPath und WS-Adressing. BPEL4WS ist dazu geeignet einen neuen Web Service aus einer Menge bereits vorhandender Web Services (auf Basis von WSDL) zusammenzusetzen.

Dabei können Geschäftsprozesse auf zwei verschiedene Arten beschrieben und instanziiert werden:

1. Ausführbare Geschäftsprozesse (Executable Business Processes) beschreiben direkt das Verhalten eines Geschäftspartners während einer Interaktion.
2. Geschäftsprotokolle (Business Protocols) dagegen beschreiben das gegenseitig sichtbare

Verhalten beim Austausch von Nachrichten von allen Beteiligten. Die Prozessbeschreibungen dieser Protokolle werden abstrakte Prozesse (Abstract Processes) genannt.

3.3 Konzepte

BPEL4WS ist blockstrukturiert, was zur Folge hat, dass bei der Definition von lokalen Umgebungen (Scopes) für eine Aktivität lokale Variablen eingeführt werden können. Mit den Scopes können außerdem Fehlerbehandlung (Fault Handler), Kompensationsbehandlung (Compensation Handler) und Ereignisbehandlung (Event Handler) zugeordnet werden.

Das heisst, dass Daten für einzelne Prozessinstanzen direkt in der lokalen Umgebung und damit in der Ablauflogik referenziert werden können. Das sind dann beispielsweise Nachrichten und Statusinformationen der beteiligten Web Services.

Das Fault Handling ist für die Behandlung von möglicherweise auftretenden Fehlern zuständig und wird auch direkt im Scope referenziert.

Wenn eine zuvor durchgeführte Aktivität nicht mehr rückgängig gemacht werden kann, kommt der Compensation Handler zum tragen. Dieser ist dann für die Kompensation zuständig. Wann dieser aktiv wird und welchen Gültigkeitsberiech dieser hat, ist von der Umgebung (Scope) abhängig.

Partner Link Types definieren die Beziehung zwischen zwei Diensten mit Hilfe von Rollen (Roles), dabei bezieht sich eine Rolle auf einen PortType in WSDL. Auf diesen Partner Link Types basierend, können Partner Links definiert und in Aktivitäten verwendet werden.

Für die Synchronisation von Aktivitäten stehen Links als Hilfsmittel bereit. Nach der Definition eines Links kann dieser dann in einer Aktivität entweder als Quelle (Source) oder als Ziel (Target) verwendet werden.

3.4 Aktivitäten

Ein Geschäftsprozess wird in BPEL4WS beschrieben, indem innerhalb einer XML-Datei mit der Endung bpel Aktivitäten, die einzelne Teile des Prozesses beschreiben, aneinandergereiht werden. Nachfolgend sind die einzelnen in BPEL4WS möglichen Aktivitäten aufgelistet:

3.4.1 <receive>, <reply>

Warten auf Nachricht (<receive>) und Antwort auf eine Nachricht (<reply>). Die Kombination beider Aktivitäten ermöglicht eine request - response Operation für einen WSDL portType des Prozesses.

3.4.2 <invoke>

Aufruf eines Web Services.

3.4.3 <assign>

Updaten von Variablen mit neuen Daten.

3.4.4 <throw> ,<catch>

Interne Fehler eines Geschäftsprozesses signalisieren (throw) bzw. abfangen (catch).

3.4.5 <terminate>

Explizites Beenden des Prozesses.

3.4.6 <wait>

Der Prozess wartet eine definierte Zeit.

3.4.7 <empty>

Leere Aktivität, wird verwendet um Fehler abzufangen oder um Aktivitäten eines Geschäftsprozesses zu synchronisieren.

3.4.8 <sequence>

Sequentielle Ausführung der nachfolgenden Aktivitäten.

3.4.9 <switch>

Aktivität zur Modellierung von Verzweigungen. Kann eine oder mehrere Bedingungen (<case condition="boolscher Ausdruck») und optional einen alternativen Zweig (<otherwise>) enthalten.

3.4.10 <while>

Mittels <while> wird die betroffene Aktivität so lange wiederholt ausgeführt, bis der boolsche Ausdruck nicht mehr erfüllt wird.

3.4.11 <pick>

Diese Aktivität wartet auf das Auftreten eines Ereignisses aus einer Menge von Ereignissen und führt danach die entsprechend angeführte Aktivtät aus.

3.4.12 <flow>

Das <flow> Konstrukt ermöglicht die parallele Verarbeitung von einer oder mehreren Aktivitäten. Mittels dem <link> Konstrukt können direkt oder indirekt eingebettete Aktivitäten synchronisiert werden.

3.4.13 <scope>

Das <scope> Konstrukt ermöglicht die Verbindung einer eingebetteten Aktion mit Variablen, einer Fehlerbehandlung und Kompensationsbehandlung. Zur Koordination wird hierbei das BusinessAgreement Protokoll der WS-Transaction-Spezifikation verwendet.

3.4.14 <compensate>

<compensate> wird verwendet, wenn eine in einen <scope> eingebettete Aktion erfolgreich durchgeführt wurde, diese Aktion aber kompensiert werden muss. Dieses Konstrukt kann nur von einem anderen compensation handler oder von einem fault handler aufgerufen werden.

3.5 Beispiel

Zur Verdeutlichung des Einsatzes der Konzepte und Aktivitäten folgt hier das Beispiel aus der BPEL4WS Spezifikation Release 1.1:

Wie in Abbildung 1 dargestellt, beginnt der Prozess mit dem Eingang einer Bestellung (Receive Purchase Order). Nach dem Erhalt der Bestellung starten drei Teilprozesse. Der erste Prozess dient der Kalkulation des Preises, der zweite Prozess der Lieferungsplanung der Produkte und der dritte Prozess der Produktionsplanung der in der Bestellung angegebenen Güter. Kommen die drei Prozesse zu einem erfolgreichen Ergebnis, gilt die Fakturierung als erfolgreich bzw. der Gesamtprozess wird beendet und der Kunde wird über diesen Erfolg informiert.

3.5.1 Einsatz der Konzepte

Im folgenden Code-Ausschnitt wird die Verwendung der Konzepte in diesem Geschäftsprozess dargestellt.

```
<process name="purchaseOrderProcess">
                <variables>
                <variable name="PO" messageType="POMessage"/>
                ...
                </variables>
                <faultHandlers>
                <catch faultName="cannotCompleteOrder" faultVariable="POFault">
                <reply partner="customer" portType="purchaseOrderPT"
                operation="sendPurchaseOrder" variable="POFault"
                faultName="cannotCompleteOrder"/>
                </catch>
                </faultHandlers>    ...
```

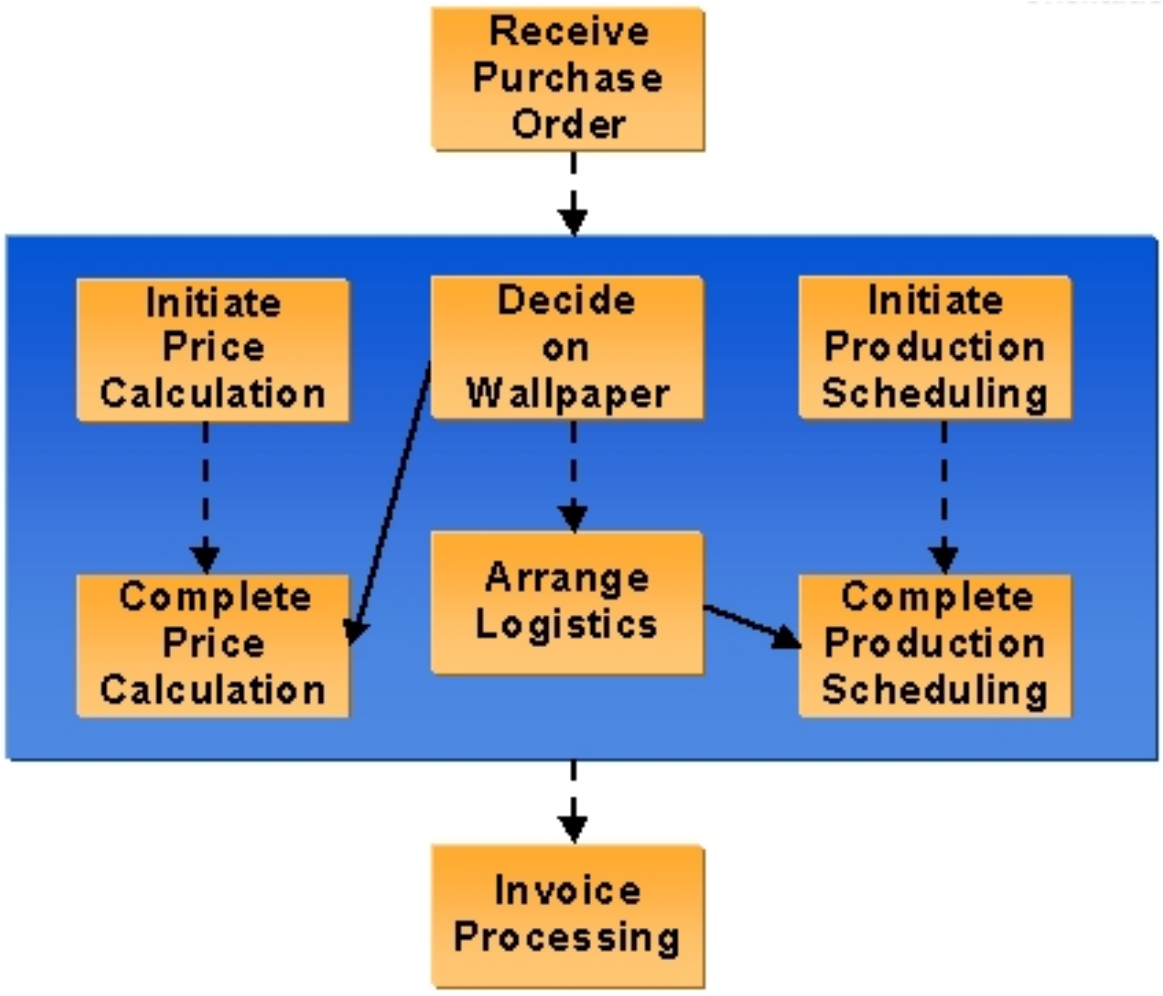

Abbildung 1: BPEL Beispiel

Wie im vorangegangen Codeausschnitt dargestellt, wird für den Geschäftsprozess eine Variable und ein Fault Handler definiert. Die Variable ermöglicht es den Teilprozessen, untereinander Daten auszutauschen oder Nachrichten zu senden. Der Fault Handler macht es dabei möglich, den Kunden beim Auftreten eines Fehlers innerhalb eines Teilprozesses über diesen Fehler zu informieren.

3.5.2 Aufbau des Prozesses

Abbildung 2 zeigt den Aufbau des Prozesses. Dabei wird innerhalb des Tags <prozess> die Aktivität <sequence> eingefügt. Das hat zur Folge, dass nach dem Eingang einer Bestellung eines Kunden (<receive.../>) die Aktivität <flow> angesprochen wird und der Kunde über die Aktivität <reply> über deren Ausgang informiert wird. Innerhalb der Aktivität <flow> werden nun die zur Kalkulation, Lieferungsplanung und Produktionsplanung benötigten Web Services angesprochen.

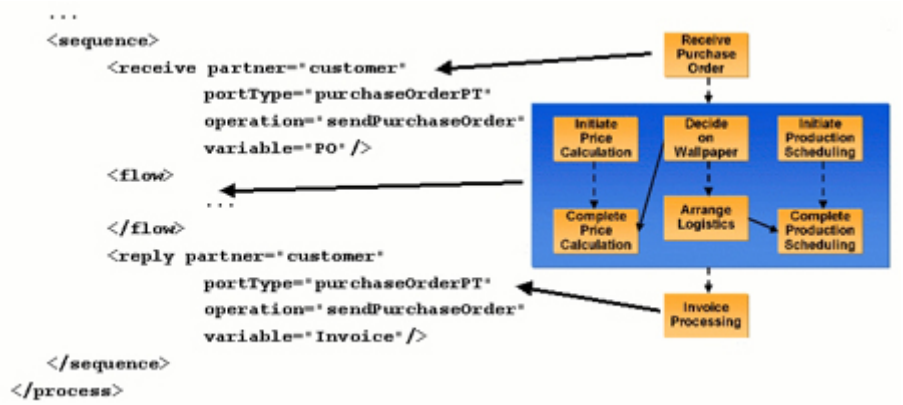

Abbildung 2: BPEL Beispiel - Prozess

3.5.3 Zusammenfassung

BPEL4WS ist eine Spezifikationssprache, die dazu verwendet wird, aus bestehenden Web Services neue Geschäftsprozesse zu generieren, wobei die neu entstehenden Geschäftsprozesse wieder die Grundlage für neue Geschäftsprozesse sein können.

Ein Blick in die Zukunft lohnt sich bei dieser Sprache auf jeden Fall, da sie bereits jetzt von den grossen Softwareherstellern wie IBM, Microsoft, Oracle, etc. unterstützt wird und ja im Moment bei OASIS zur Standardisierung liegt. So wird sicher in der Zukunft noch das Interesse weiterer Hersteller für diese fundierte Modellierungssprache wachsen.

Auch an vorgefertigten Beispielen und Werkzeugen mangelt es nicht, für die gängigsten Szenarien kann man im Internet Beispiele finden. Unter http://de.wikipedia.org/wiki/BPEL#Weblinks finden sich eine Auflistung von BPEL Engines mit denen BPEL-Code ausgeführt werden kann.

3.6 BPEL-Tools

In dieser Fachstudie steht **WS-CDL** im Fokus der Betrachtung, weshalb keine BPEL-Tools evaluiert wurden. Trotzdem soll hier eine kurze Auflisting einiger Tools erfolgen, um die später besprochenen WS-CDL-Tools hierzu in Relation betrachten zu können. Der geneigt Leser sei auf die, ebenfalls am Insititut für Architektur von Anwendungssystemen enstandenen, Fachstudie Nr. 38 [25] verwiesen in der BPEL-Workflow Modellierungstools genauer untersucht wurden.

Grober Überblick:

- **IBM** - WebSphere Studio Application Developer Integration Edition
- **Oracle** - BPEL PM Designer
- **IDS Scheer** - ARIS Toolset (ARIS for SAP NetWeaver)
- **BEA** - Weblogic Intergration (ab Version 9.0)
- etc

Es ist also festzuhalten, dass eine Anzahl nahmhafter Firmen, wie z.B. IBM und Oracle, Tools speziell für BPEL bzw. zumindest mit Unterstützung für BPEL am Markt haben.

4 WS-CDL

Die Web Services Choreography Description Language (WS-CDL) stellt den eigentlich Kernuntersuchungspunkt dieser Arbeit dar. Wie auf der folgenden Abbildung 3 gut zu erkennen ist, handelt es sich bei WS-CDL um eine noch relativ junge Sprache, die erst seit dem Jahre 2003 existiert.

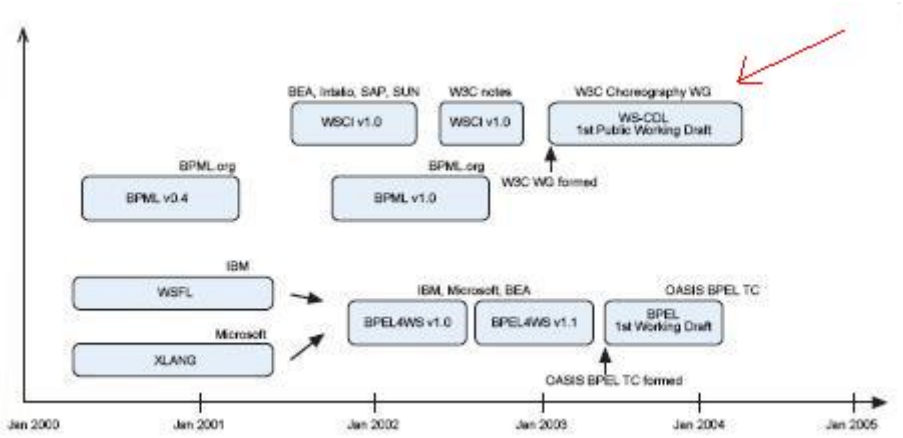

Abbildung 3: Evolution of the standards

4.1 Point of View

siehe hierzu Kapitel 6.1

4.2 Kernkonstrukte

4.2.1 package

Ein (Choreographie-) Package bündelt eine Reihe von WS-CDL Typdefinitionen und stellt den Namespace für die Definitionen zur Verfügung. In einem solchen Paket muss zumindest eine Choreographie definiert werden, die maximale Anzahl ist nicht festgelegt. Weiterhin können Typdefinitionen aus anderen Choreographie-Packages eingebunden werden.

4.2.2 roleType

Rollentypen definieren Rollen für Geschäftsprozesse. Ein Rollentyp zählt die verschiedenen, von außen beobachtbaren, Verhaltensweisen einer Geschäftspartei auf, die in Zusammenarbeit mit anderen Parteien auftreten.

4.2.3 relationshipType

Ein Relationship Type legt die Beziehungen zwischen jeweils zwei Rollentypen fest, die für eine erfolgreiche Zusammenarbeit notwendig sind. Zudem kann das Verhalten der einzelnen Rollentypen in einer solchen Geschäftsbeziehung definiert werden. Wird dies unterlassen, so werden automatisch alle möglichen Verhaltensweisen eines Rollentyps verbindlich akzeptiert.

4.2.4 participantType

Ein Participant Type identifiziert all die Rollentypen, die logisch zusammengehörig sind und gemeinsam implementiert werden sollten. Im Hinblick auf das beobachtbare Verhalten bilden die Rollentypen eine Einheit und es ist notwendig, dass sie von derselben logischen Entität oder Organisation implementiert werden.

4.2.5 channelType

Dieses Konstrukt realisiert die Schnittstelle, über die Informationen bei Zusammenarbeit zwischen Parteien ausgetauscht werden können. Es legt fest, wo, wie und welche Informationen verarbeitet werden und welcher Rollentyp das Ziel des Informationsflusses ist.

4.2.6 informationType

Information Types beschreiben die Art der Information, die in einer Choreographie benutzt werden. Laut Spezifikation soll durch diese Abstraktion vermieden werden, direkt die Datentypen zu referenzieren.

4.2.7 variableDefinitions / variable

Variablen speichern Informationen über sichtbare Objekte in einer Zusammenarbeit. Man kann hierbei 3 Typen unterscheiden: **Information Exchange Capturing Variables**, die als Resultat eines Informationsaustauschs befüllt werden oder zu Beginn eines solchen zu befüllen sind und den Informationsaustausch an sich festhalten; **State Capturing Variables** stellen Zustandsvariablen dar, die das Ergebnis eines Informationsaustauschs selbst speichern, sowie **Channel Capturing Variables**, die z.B. Informationen bezüglich einer URL, die das Ziel einer Nachricht darstellt oder notwendigen Sicherheitsrichtlinien, die zu erfüllen sind, enthalten.

4.2.8 token

Tokens stellen bestimmte Teile einer Variable dar, die zur Nutzung durch eine Choreographie notwendig sind. So kann eine Variable "Buchung"beispielsweise sämtliche Buchungsinformationen wie Reisetermin, Zielort, Preis, usw. enthalten und ein Token Preisnur den gewünschten Datenteil.

4.2.9 tokenLocator

Definiert den Bereich, in dem ein Token zu finden ist und stellt somit einen Abfragemechanismus für diese Informationsausschnitte dar.

4.2.10 choreography

Eine Choreographie ist das Herzstück der WS-CDL Technik. Sie regelt den Nachrichtenaustausch zwischen interagierenden Parteien sowie einzelne Aktivitäten, welche die Geschäftspartner zu tätigen haben (siehe Abbildung 4).

4.2.11 interaction

Interaktionen bilden die Basis einer Choreographiezusammenstellung. Das Ergebnis einer Interaktion sind ausgetauschte Informationen sowie eine mögliche Synchronisierung des beobachtbaren Informationsaustauschs. Mehrere Interaktionen werden kombiniert um eine Choreographie zu schaffen, die dann in verschiedenen Geschäftsumfeldern genutzt werden kann.

4.2.12 workunit

Eine Workunit kann Bedingungen beschreiben, die notwendig sind, um den Fortschritt in einer Choreographie zu gewährleisten und somit mit der aktuellen Arbeit fortfahren zu können. Workunits stellen zum Beispiel auch die Konsistenz der Zusammenarbeit von Geschäftspartnern sicher. So können Variableninformationen geschützt werden, so dass sie nur unter bestimmten Voraussetzungen bearbeitet werden können. Anwendungen können durch Workunits in Verbindung mit Exception Blocks aus fehlerhaften Zuständen wiederhergestellt werden oder durch Finalizer Blocks Aktionen ausgeführt werden auch wenn eine Choreographie bereits abgeschlossen ist.

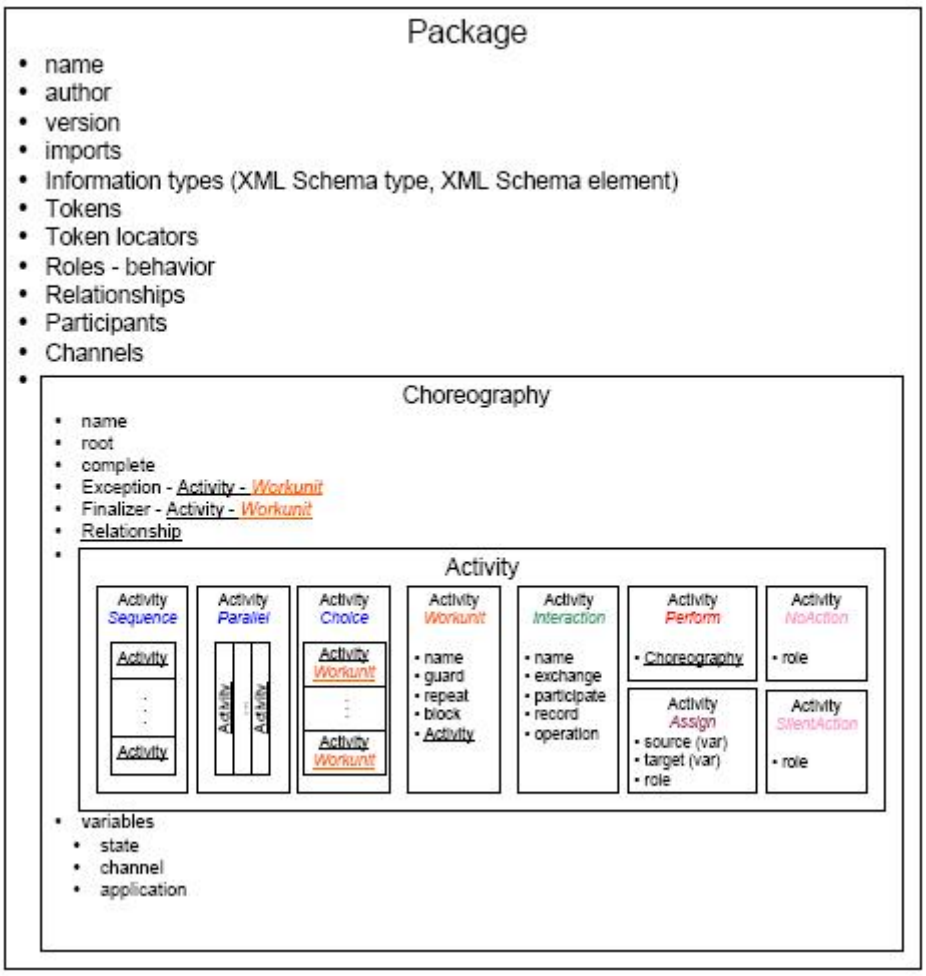

Abbildung 4: WS-CDL Choreography

4.2.13 exceptionBlock

In Exception Blocks werden Fehlersituationen behandelt, die durch Fehlervariablen in Interaktionen ausgelöst werden. Eine Exception kann aktiv ausgelöst werden, wenn bestimmte Bedingungen erfüllt werden müssen oder sie werden durch Constraints, wie etwa Zeitüberschreitung bei Warten auf eine Antwort, automatisch aktiviert.

4.2.14 finalizerBlock / finalizer

Nach der erfolgreichen Beendigung einer Choreographie ist es eventuell nötig, weitere Aktionen durchzuführen, um Ergebnisse der vollendeten Aktivitäten zu bestätigen, rückgängig zu machen oder zu ändern. Um diese Änderungen durchzuführen, können Finalizer Blocks definiert werden, die nach Beenden einer Choreographie aufgerufen werden. Mit finalize können bereits geschlossene Choreographien zu einem definierten Ende gebracht werden, indem direkt Finalizer Blocks aktiviert werden.

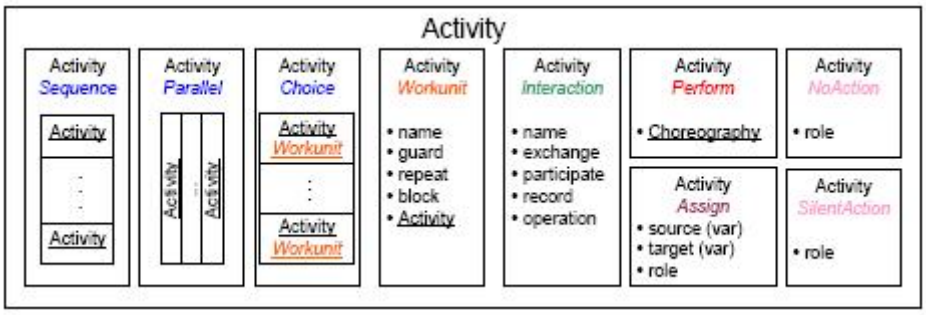

Abbildung 5: WS-CDL Activity

4.3 WS-CDL: Out of Scope

Laut der WS-CDL Charter[8] gibt es im Wesentlichen drei Aspekte, die sich außerhalb des Scopes von WS-CDL befinden:

1. **Qualities** - Transaktionen, Sicherheit, Zuverlässigkeit, Erreichbarkeit und andere Qualitäten[1] sind zwar mehr oder weniger eng verbunden mit Web Service Choreography, aber es ist nicht Ziel von WS-CDL bzw. der entsprechenden Working Group Mechanismen zu definieren.

2. **Mappings to Programming Languages** - Web Services sind aus Interfaces zu Applikationen zusammengesetzt, die wiederum in verschiedenen Programmiersprachen geschrieben sein können. WS-CDL bzw. die entsprechende Working Group wird keine Mappings WS-CDL -> Programmiersprache definieren.

3. **Graphical notation** - Ein (wichtiger) Aspekt der Modellierung von Choreographies ist die graphische Repräsentation. Die Working Group wird keine solche graphische Notation definieren.

Das Fehlen einer graphischen Notation ist wohl als Hauptschwachpunkt in Sachen Benutzbarkeit einzustufen. Ohne eine klar definierte Grundlage für eine graphische Repräsentation ist das graphische Editieren und damit das Bereitstellen von benutzerfreundlichen und intuitiv bedienbaren Werkzeugen praktisch unmöglich, da sich die Funktionalität praktisch auf eine erweiterte Text- bzw. XML-Verarbeitung beschränkt. Damit ist es für WS-CDL als Sprache sehr schwierig sich mit etablierten Web Service Sprachen wie BPEL4WS [1] und den entsprechenden Tools (wie z.B. IBM Modeler Business Integration [13]) zu messen. Diesen Punkt werden wir auch bei der Betrachtung der vorhandenen WS-CDL-Tools beim Kapitel pi4soa noch einmal aufgreifen müssen.

[1] Quality of Services

4.4 Weitere Schwächen in WS-CDL

1. Fehlende Trennung von Meta-Model und Syntax und damit Fehlen einer strikten Trennung von Syntax und Semantik. „Lack of separation between meta-model and syntax. The WS-CDL specification tries to simultaneously define a meta-model for service choreography and an XML syntax. As a result, there is no strict separation between semantic and syntactic aspects.“ - WS-CDL-Barros et al [7]

2. Fehlende Unterstützung für spezielle Kategorie von Use Cases, insbedonder für **one-to-many**-Beziehungen. „Lack of direct support for certain categories of use cases. „...“ In particular, while many service choreography use cases involve **one-to-many** or multi-transmission interactions, the language does not provide direct support for these scenarios.“ - WSCDL-Barros et al [7]

3. Fehlen einer umfassenden formalen Basis. Obwohl von diversen Experten ausgesagt wird, daß WS-CDL insbesondere auf dem pi-Calculus [14] basiert, wie das genaue Mapping darauf aussieht, ist jedoch relativ offen. „Lack of comprehensive formal grounding.“ - WSCDL-Barros et al [7]

4. Im Gegensatz zu BPEL ist WS-CDL bis jetzt kein formaler Standard, sondern immer noch ein „Working Draft“. Dies mag auch ein Grund dafür sein, daß es bis jetzt keine am Markt erhältlichen Tools speziell hierfür gibt.

5. aus 4. ergibt sich, daß kein Herstellersupport geboten wird und das sich WS-CDL deshalb schwer tun dürfte in der Business Welt Fuß zu fassen

6. aus 4. ergibt sich weiterhin, daß die vorliegende Version der Spezifikation an vielen Stellen unvollständig, schwammig, offen oder fehlerbehaftet ist.

7. Die Einarbeitung in WS-CDL ist auch aufgrund mangelnder Beispiele bzw. Patterns relativ schwierig.

4.5 Veränderung in WS-CDL

Seit dem Beginn dieser Fachstudie im Juli 2005 hat sich an WS-CDL inhaltlich nichts geändert, da kein neuer Working Draft bzw. keine neue Release-Version von WS-CDL herausgegeben wurde. Trotzdem ist anhand einiger Fakten zu erkennen, daß an der Entwicklung und Verbesserung gearbeitet wird:

1. Laut Informationen aus den pi4soa Release Notes [11]: „The CDL namespace is now 'http://www.w3.org/2005/08/ws-chor/cdl' in line with the Candidate Recommendation version of the specification.“ - der Namespace wurde also erneuert, das genannte Dokument konnte jedoch noch nicht auf den Seiten des W3C[2] gefunden werden.

4.6 WS-CDL Tools

Es sind verschiedene Tools für WS-CDL im Umlauf, wobei festzuhalten ist, dass nahezu alle Tools noch im Anfangsstadium sind. Im Gegensatz zu den in Kapitel 3.6 genannten BPEL-Workflow

[2] http://www.w3c.org

Modellierungstools sind die meisten WS-CDL-Tools keine kommerziellen Produkt nahmhafter (Software-)Hersteller sondern Open-Source-Produkte (Community Produkte), es ist also mit wenig bis keinem Support bzw. nur mit Community-Support zu rechen. Für unsere Betrachtungen waren zwei Tools (**WSCDL-Eclipse** und **pi4soa**) von besonderem Interesse. Diese sollen hier eingeführt und bewertet werden.

4.6.1 WSCDL-Eclipse

WS-CDL Eclipse ist eine auf Eclipse basierende Entwicklungsumgebung, mit der man WS-CDL Dokumente erstellen und diese in einer Art graphischen Simulation ablaufen lassen kann. Das Tool ist unter der GNU General Public License (GPL) veröffentlicht.

Voraussetzungen:

1. Sun Java 2 SDK, Standard Edition, version 1.4.2
2. (oder Sun Java 2 Java 2 Platform Standard Edition 5.0 - bevorzugt!)
3. Eclipse version 3.0.1+
4. Eclipse Graphical Editing Framework (GEF) 3.0.1+

Wir benutzten Eclipse 3.0 mit GEF 3.1 und Java Runtime 1.5.0.

Installation

Das Tool ist ein Eclipse-Plugin und befindet sich in einem gepackten Zip-Archiv. Man entpackt das Archiv direkt in das plugins-Verzeichnis der Eclipse-Installation und kann nach Neustart von Eclipse direkt die Funktionalität nutzen. Um mit WS-CDL Eclipse eine Simulation durchzuführen ist zudem das Graphical Editor Framework (GEF) notwendig, welches separat installiert werden muss. Das GEF bekommt man im Downloadbereich der Eclipse Project Homepage [6]. Dieses Plugin ist befindet sich ebenfalls in einem Archiv, welches ins Installationsverzeichnis von Eclipse entpackt werden muss.

Erfahrungen mit dem Tool

WS-CDL Eclipse bietet die Möglichkeit, ein neues WS-CDL Projekt oder eine neue WS-CDL Datei anzulegen. Dabei zeigen sich auch gleich die Stärken dieser Entwicklungsumgebung. Legt man eine neue WS-CDL Datei an, so kann man Schritt für Schritt die einzelnen Typdefinitionen eines Dokuments abarbeiten, denn für jeden Teil, seien es nun roleTypes, channelTypes, Choreographien, usw. steht eine Eingabemaske zur Verfügung, die ausgefüllt werden kann. Jegliche Attribute, die es für ein Element gibt, sind in diesem Assistent vorgesehen und müssen nun nur noch mit entsprechenden Eingabewerten gefüllt werden. Man hat jederzeit die Möglichkeit im Assistenten einen Schritt vor oder zurück zu springen und später erneut dahin zurückzukehren. Die Eingabe vereinfachen würde eine Erkennungsmerkmal welches anzeigt, ob ein Element beziehungsweise Attribut optional ist (also auch ganz weggelassen werden kann). Diese könnte zum Beispiel durch Fettdruck des selbigen geschehen. Hat man alle Schritte abgearbeitet, kann man den Assistenten beenden. Es ist ebenfalls möglich auch ohne Ausfüllen fortzufahren und nur eine leeres WS-CDL Dokument anzulegen, welches nur den Paketname und den XML-Namespace enthält.

Nach Abschluss findet man ein wohlgeformtes XML bzw. eben WS-CDL Dokument vor, welches der WS-CDL Spezifikation entspricht.

Der Assistent ist zwar sehr hilfreich, jedoch wird schnell deutlich, dass man schon eine sehr genaue Vorstellung des Webservices, insbesondere über einzelne Rollen und deren Zusammenarbeit, haben muss bevor man überhaupt beginnt. Dies ist erforderlich, da man diesen Assistenten bzw. einen ähnlichen Dialog, der einen beim Anlegen der Dokumentstruktur (WS-CDL spezifische Elemente und Attribute) unterstützt, im Nachhinein nicht mehr aufrufen kann. Nach Anlegen des Dokuments und somit nach Abschluss des Assistenten bietet WS-CDL Eclipse also nur die Möglichkeiten eines reinen XML-Editors, wenn man von der Möglichkeit der Simulation absieht.

Die Arbeit mit diesem Plugin gestaltete sich demnach auch nicht einfacher als mit jeglichem anderen XML-Editor. Vielmehr taucht zusätzlich das Problem auf, dass Fehler in der Dokumentenstruktur, seien es zum Beispiel Verstöße gegen Wohlgeformtheitsregeln oder gegen unzulässige Einträge in Attributen, erst beim Speichern der Datei in einem kurzen Dialogfenster angezeigt werden. Hierbei wird einem immer der erste Fehler angezeigt, der im Dokument vorhanden ist, weitere Fehler sind nicht ersichtlich. So lassen sich auch während der Bearbeitung keine Fehler im Dokument erkennen. Auch lässt der Hinweisdialog oft keine Rückschlüsse auf den wirklichen Fehler zu, da es weder eine Positionsangabe bezüglich Zeilen- und Spaltenzahl gibt noch einen Hinweis, ob ein gegen die Wohlgeformtheitsregeln, die WS-CDL Spezifikation oder andere Voraussetzungen verstoßen wurde.

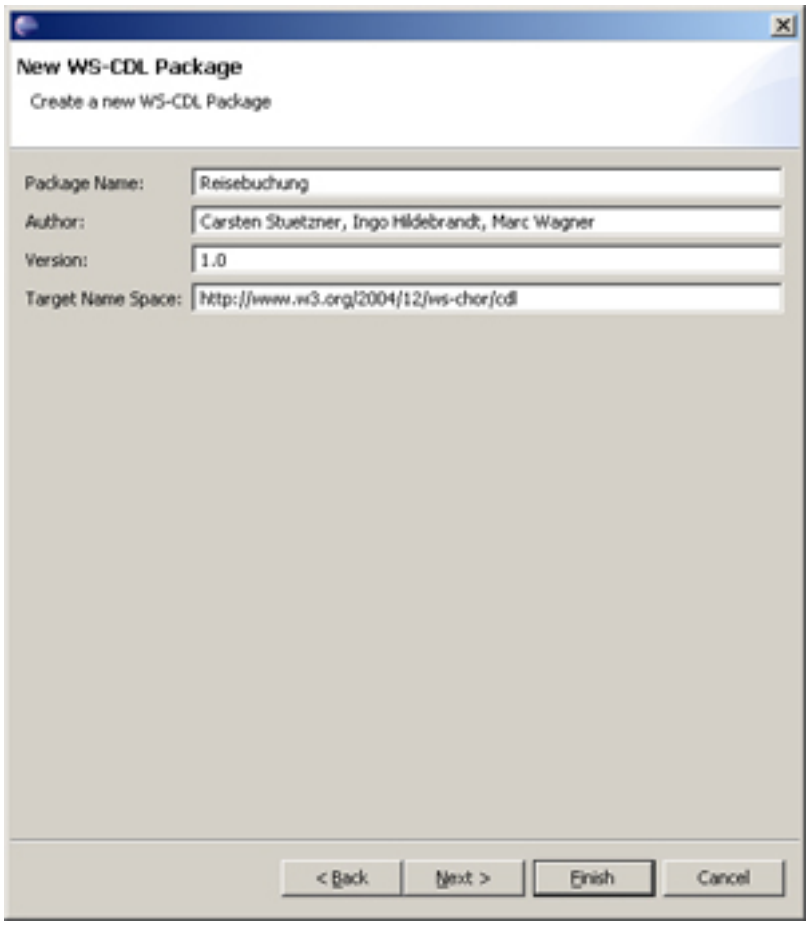

Abbildung 6: WS-CDL Eclipse Dialog (1)

Da das WS-CDL Eclipse Tool Dokumente erst dann akzeptiert, wenn sie vollkommen korrekt sind, lässt sich während der Arbeit kein Überblick über den bei XML-Dokumenten typischen baumartigen Aufbau des Dokuments gewinnen. Dieses Outline-Feature, was Eclipse standardmäßig bietet ist einfach nicht verfügbar. Die Struktur lässt sich also nur durch saubere Einrückung erahnen.

Verzichten muss man leider auch auf Tag Completion, also auf ein automatisches Vorschlagen oder Vervollständigen der XML-Elementen beziehungsweise der XML-Attribute. Dies ist vor allem bei längerem Arbeiten eine Erleichterung und wäre gerade für die WS-CDL spezifischen Elemente und Attribute eine große Hilfe. Leider macht das Tool zunächst keinen Unterschied ob man einen WS-CDL-Tag eingibt oder irgendeinen anderen XML-Tag.

Hat man letztendlich ein korrektes WS-CDL Dokument erstellt so erhält man leider trotzdem eine Fehlermeldung mit Inhalt null". Ein Modell konnte somit auch mit Dokumenten, die vom Plugin selbst als korrektes WS-CDL Dokument validiert wurden, nicht erzeugt werden. In der eingesetzten Entwicklungsumgebung gelang es nicht, eine Choreographie zu simulieren, geschweige denn graphisch darzustellen, jegliche Versuche (auf verschiedenen Computern mit verschiedenen Eclipse Versionen) schlugen fehl. Somit lässt sich über diese Features im Rahmen dieser Fachstudie keine Aussage treffen.

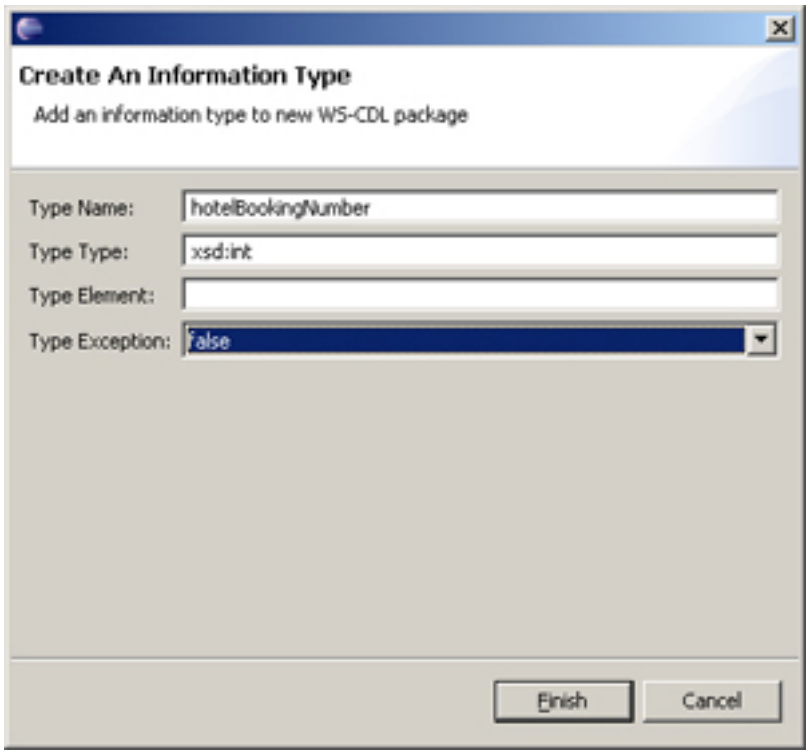

Abbildung 7: WS-CDL Eclipse Dialog (2)

Abschliessende Betrachtungen

Die Ansätze des WS-CDL Eclipse Projektes sind vielversprechend. Der Assistent zu Beginn bietet eine gute Möglichkeit ein WS-CDL Dokument zu erstellen. Jedoch wäre es wünschenswert dieses Feature auch nach erstmaligen Abschluss nutzen zu können, indem man beispielsweise einzelne Schritte des Assistenten zum Einfügen neuer Typen beliebig oft ausführen könnte. So wäre das Manko der nicht vorhanden Tag Completion nicht so gravierend. Ersetzen kann diese Möglichkeit das Fehlen noch lange nicht. Man tut sich mit einem ausgereiften XML-Editor wie «oXygen/> XML Editor"doch einiges leichter, da es hier das Vervollständigen von XML-Elementen und XML-Attributen gibt sobald jene einmal bekannt sind. Sehr hilfreich wäre in einem WS-CDL spezifischen Editor die Kenntnis der verfügbaren WS-CDL Elemente und Attribute.

Insbesondere sollte die Möglichkeit der Fehlerverfolgung bereits während der Bearbeitung nicht fehlen. Hier gibt es weiteren Entwicklungsbedarf. Verstöße gegen Wohlgeformtheitsregeln und gegen die WS-CDL Spezifikation sollten sofort im passenden View (Problems") angezeigt werden, wie man es aus der Programmiererfahrung mit Eclipse kennt. Natürlich ist eine Überarbeitung im Bereich der grafischen Darstellung und Simulation notwendig, so dass diese überhaupt lauffähig ist.

Vorschlag: Nutzung eines anderen XML- bzw. WS-CDL Editors bis das Dokument ein gültiges WS-CDL XML Dokument darstellt. Danach kann man die Funktionalität der Simulation nutzen, sobald diese einwandfrei arbeitet.

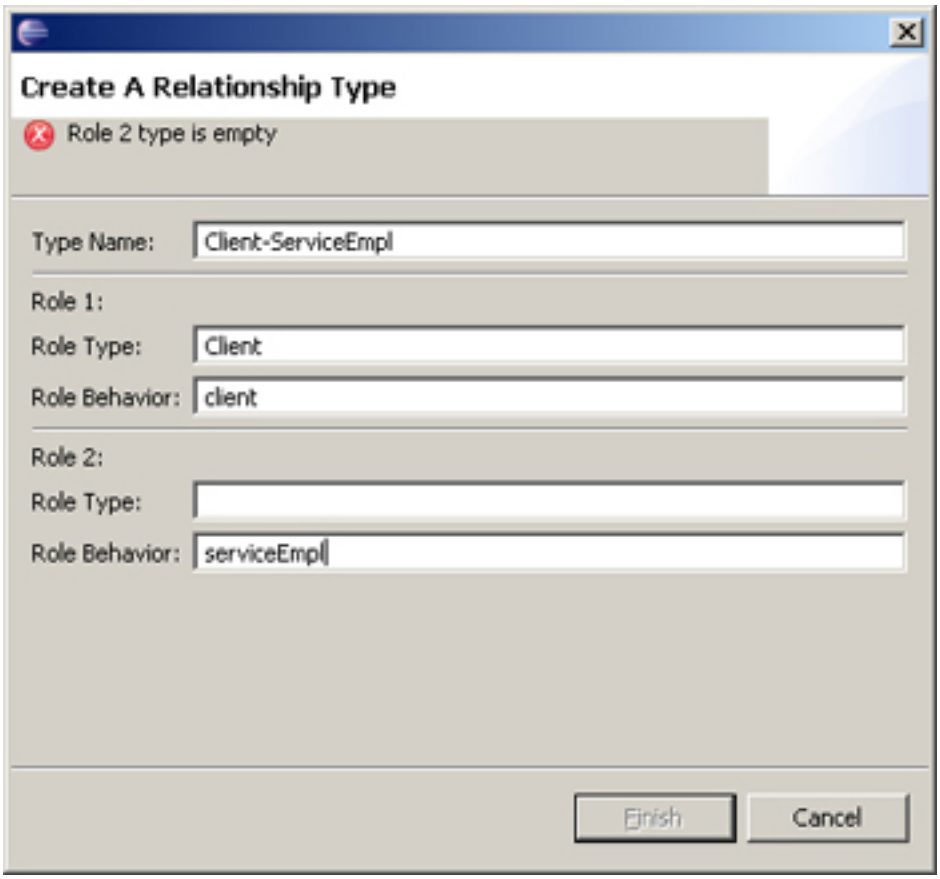

Abbildung 8: WS-CDL Eclipse Dialog (3)

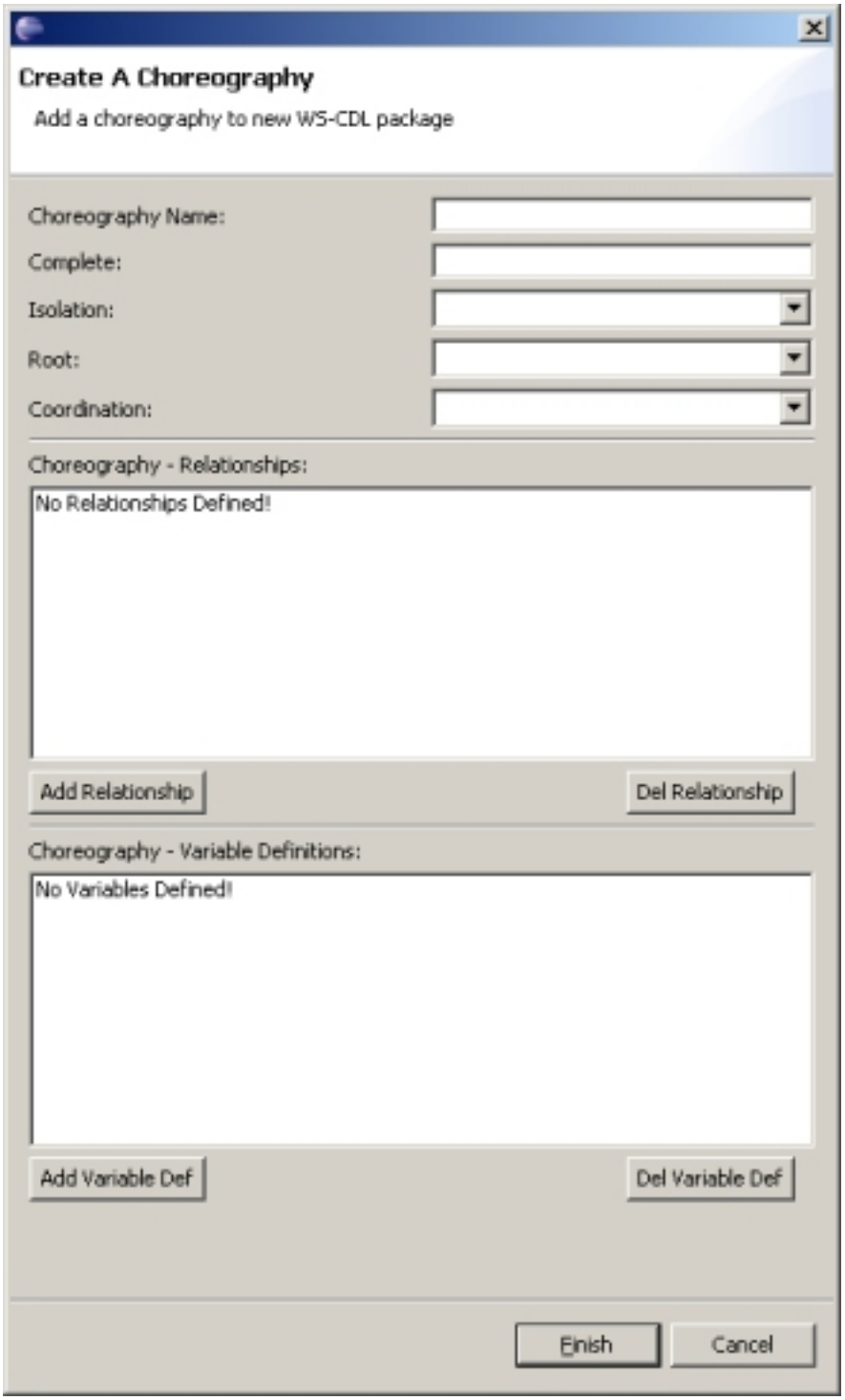

Abbildung 9: WS-CDL Eclipse Dialog (4)

4.6.2 pi4soa

Das von Pi4 Technologies hergestellte Tool Pi Calculus for Service Oriented Archicteture (pi4soa) ist laut Auskunft der Hersteller-Webseite (http://www.pi4tech.com/) die erste Implementierung der W3C Webservice Choreography Description Language (WS-CDL). Auf der Homepage sind nur wenige Informationen zum eigentlichen Tool vorhanden und die eigentlichen Dateien des Tools sind dort auch nicht zu finden. Das eigentlich Tool und die Release Notes sind auf der Seite http://www.pi4soa.org zu finden, welche auf die SourceForge-Platform und das dort befindliche Projekt (http://sourceforge.net/projects/pi4soa/) verweist.

Eine neue Version des Tools wird in regelmäßigem Abstand released. Zum Anfang der Fachstudie war noch die Version 1.1.0 vom 04. Juli 2005 aktuell, diese wurde jedoch am 02. September 2005 durch die neue Version 1.2.0 ersetzt. Laut der Projektseite befindet sich das Tool, trotz des neuen Releases, immer noch im Alphastadium, ist also eher als fehlerbehaftet und nicht voll einsatzfähig einzustufen. Das diese Einschätzung durchaus zutrifft erfährt der Benutzer bereits bei der Installation.

Vorbereitung vor der Installation

Da pi4soa kein eigenständiges Programm ist, sondern lediglich aus einer Sammlung von Plugins für die Eclipse-Plattform [3] besteht, muss auf dem Rechner auf dem pi4soa benutzt werden soll eine aktuelle Version eben dieser installiert sein. Da sich die „Installation“ von Eclipse auf das Entpacken des entsprechenden Archivs beschränkt, sei der geneigte Leser bei möglicherweise auftretenden Probleme auf die Eclipse-Homepage [3] verwiesen.

Installation - Version 1.2.0

Die Installation der Software gestaltet sich, nachdem man die erforderliche Software downgeloaded hat, relativ einfach und besteht im Wesentlich aus einfachem Entpacken von Zip-Dateien und Kopieren der entpackten Dateien in die dafür vorgesehenen Ordner:

1. GEF entpacken und die Ordner in das Eclipse-Plugins-Verzeichnis kopieren (/yourpath/eclipse/plugins/)
2. EMF entpacken und die Ordner in das Eclipse-Plugins-Verzeichnis kopieren (/yourpath/eclipse/plugins/)
3. pi4soa entpacken und die Ordner in das Eclipse-Plugins-Verzeichnis kopieren (/yourpath/eclipse/plugins/)

Nachdem dies erledigt ist, kann überprüft werden, ob die Eclipse-Plattform die Plugins richtig erkannt hat. Hierzu muss die Eclipse-Plattform gestartet werden und dann unter „Help“ -> „About Eclipse“ -> „Plugin-Details“ (**Abbildung 10**) nachgeschaut werden, ob die entsprechenden pi4soa-Plugins (**Abbildung 11**) vorhanden sind. Ist dies der Fall sollte mit der Arbeit begonnen werden können.

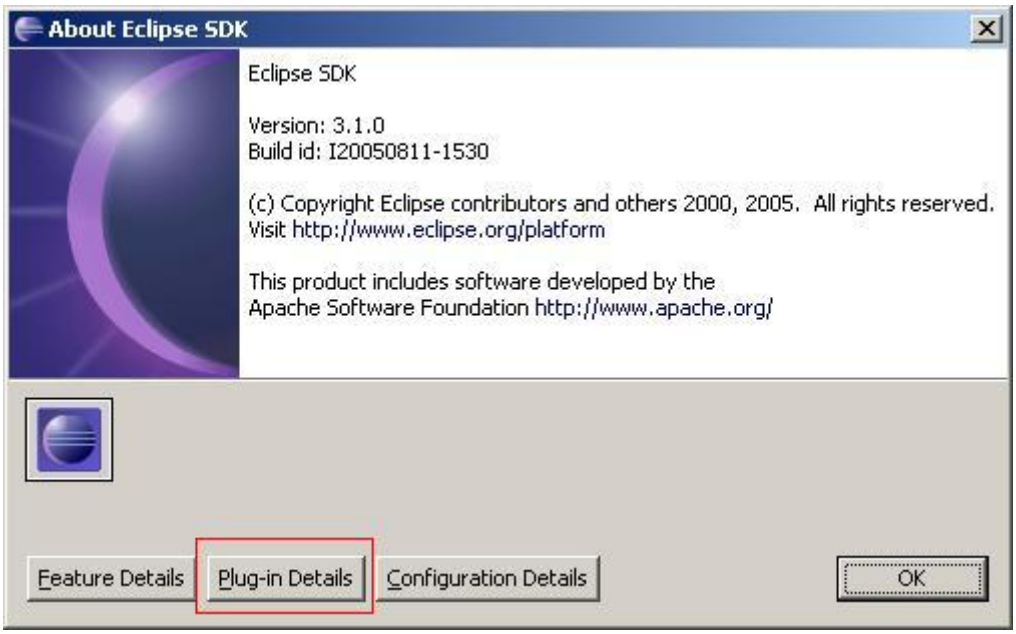

Abbildung 10: Überprüfen der pi4soa-Installation (1)

4.6.3 Probleme nach der Installation

Bei unserer Evaluierung des Tools traten nach der oben genannten Installation des Tools diverse Fehler auf. Es konnte z.B. keine Choreography Description angewählt werden, da sofort ein Fehler auftrat (siehe **Abbildung 12**). Dieser Fehler war nicht ganz nachvollziehbar, da es sich bei der Evaluationsumgebung um eine „frische“ Installation von Eclipse handelte. Über die Plugin-Developer-Ansicht konnte zwar die Information gewonnen werden, dass es sich um einen Fehler im Paket „org.eclipse.jface“ (siehe **Abbildung 13**) handelt, dies war aber nur bedingt hilfreich, da über diesen Fehler auf der Homepage von pi4soa nichts zu finden war.

Erst am **23. September 2005** wurde der „Patch1“ [12] zur Verfügung gestellt der diesen (vermutlich aber auch diverse andere) Fehler behob. Damit war es dann endlich möglich die Funktionalität **Choreography Description** (siehe Abbildung 14) zu benutzen.

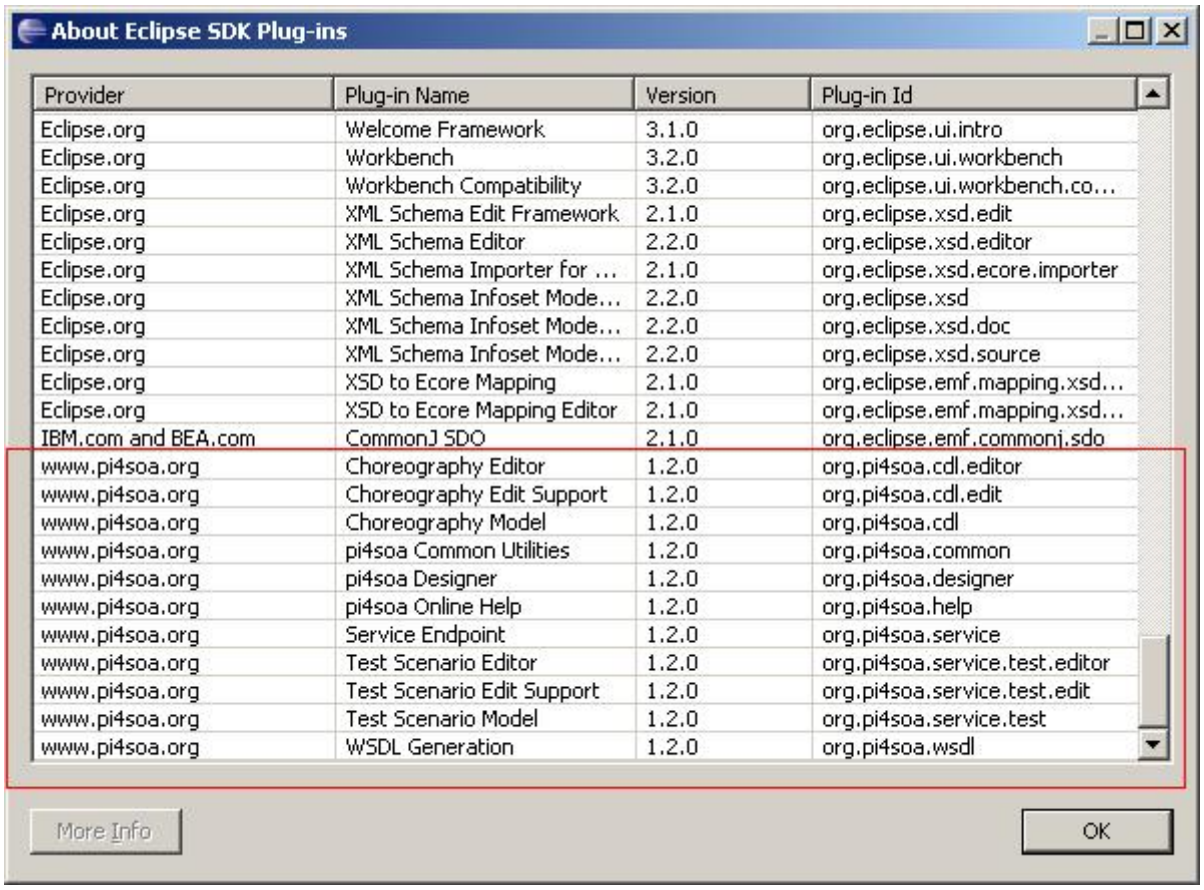

Abbildung 11: Überprüfen der pi4soa-Installation (2)

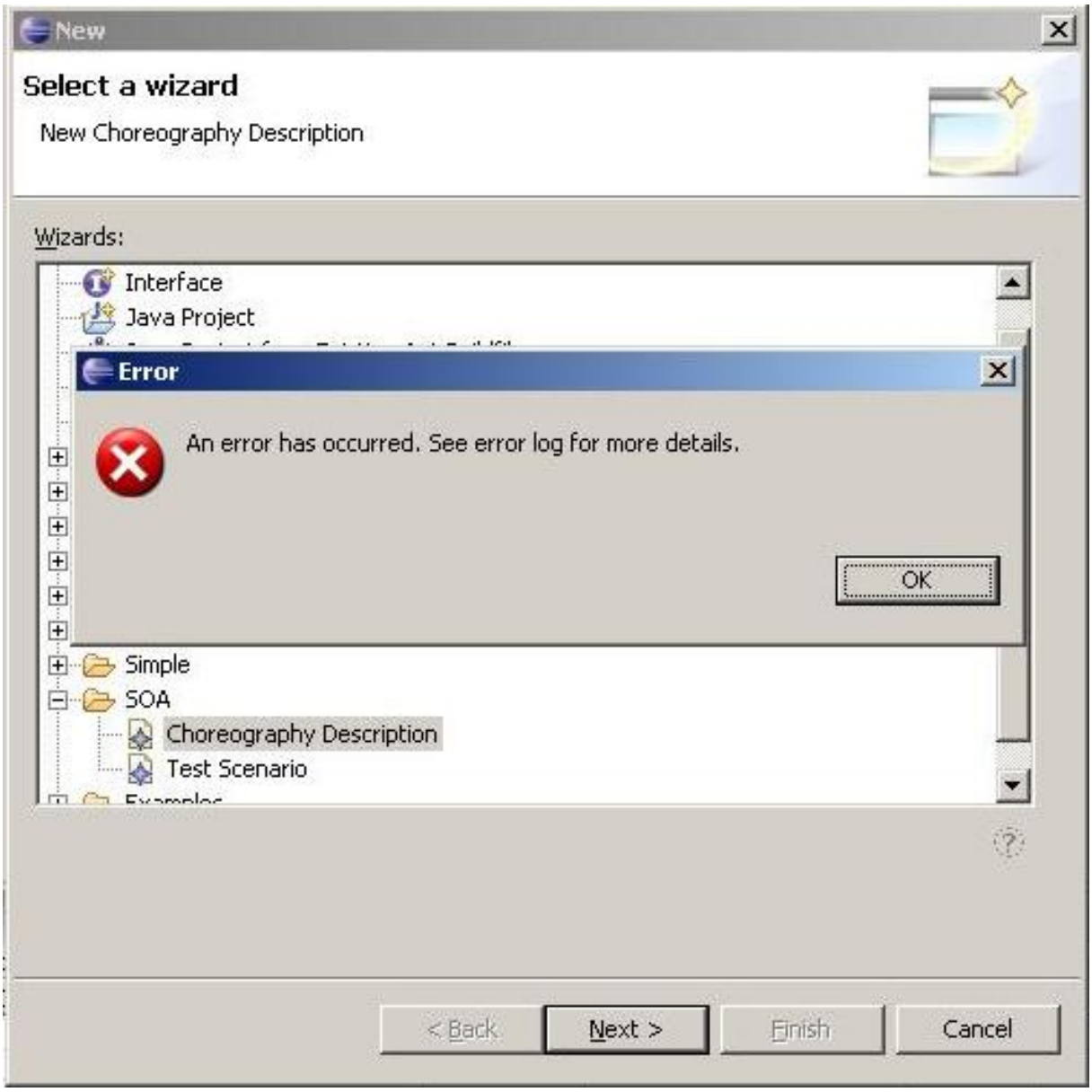

Abbildung 12: pi4soa Error 1

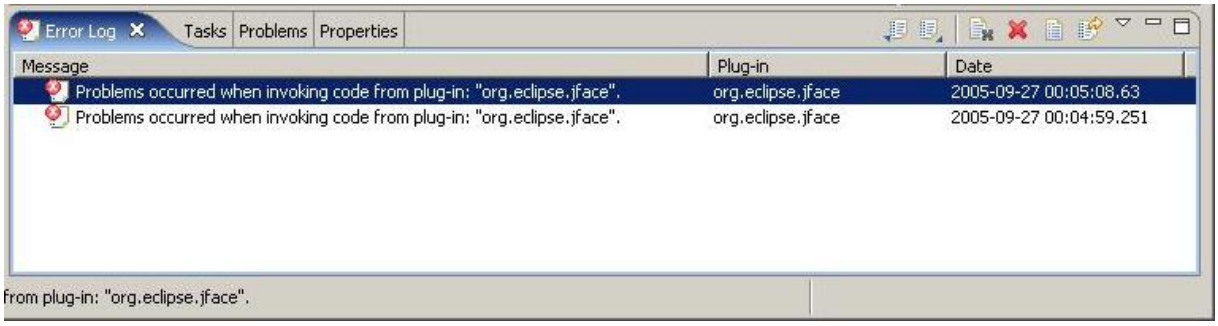

Abbildung 13: pi4soa Error 2

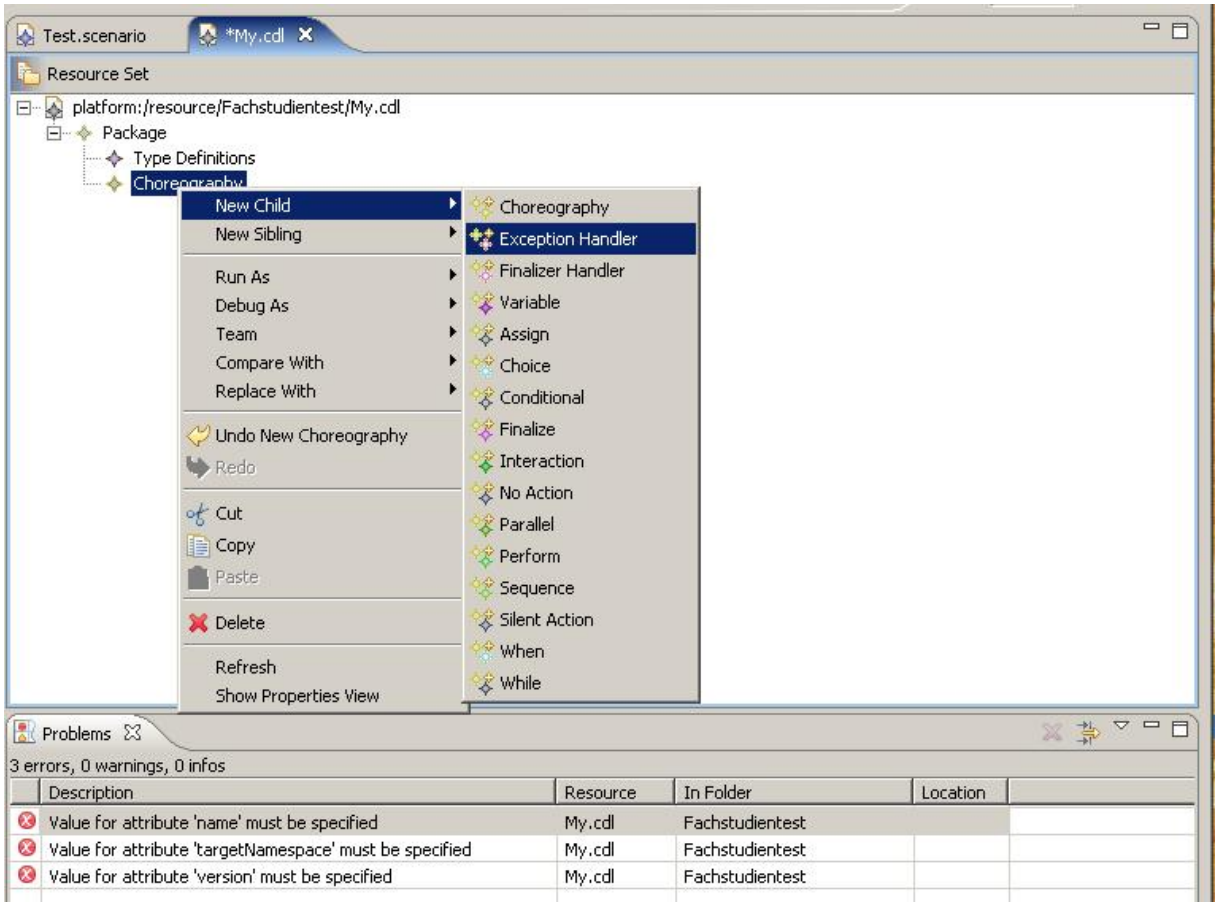

Abbildung 14: Funktion „Choreography Description“

Benutzung

Nach erfolgreicher Installation stehen sowohl eine neue **Perspektive SOA** als auch neue Funktionalitäten unter „SOA“ („New“ -> „File“ -> „Other“) bereit (siehe Abbildungen 15 und 16).

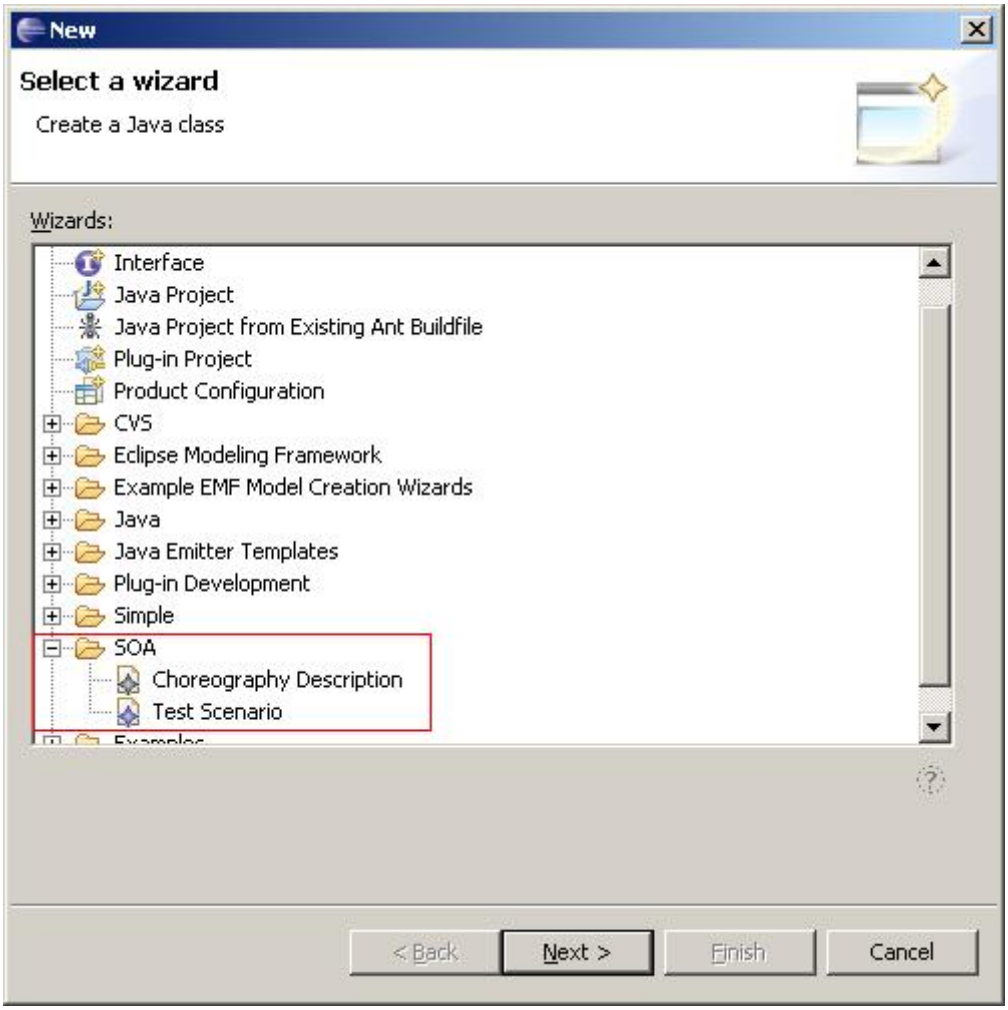

Abbildung 15: pi4soa

In pi4soa gibt es zwei verschiedene Möglichkeiten, ein WS-CDL Dokument zu erstellen. Diese beiden Ansichten können auch parallel genutzt werden, dabei muss nur beachtet werden, dass bei jedem Wechsel von der einen zur anderen Ansicht gespeichert werden muss.

Modeleditor

Die erste Ansicht ist der CDL Model Editor. In diesem kann in Baumform das Dokument aufgebaut werden. Dabei bietet der Editor jedoch nur wenig Unterstützung. Beim Rechtsklick auf ein Element werden nur die möglichen untergeordneten Elemente angezeigt und können ausgewählt werden. Die einzige Erleichterung dabei ist das Ausschließen von Elementen, die an der ausgewählten Stelle nicht angelegt werden können. Die Eigenschaften müssen dann bei den Properties von Hand eingegeben werden. Dabei folgt keine Gültigkeitsprüfung und es wird dabei

leider auch nicht angezeigt, ob die Attribute optional oder zwingend sind. Dabei ist man auf das Nachschlagen in der Spezifikation von WS-CDL angewiesen. Es handelt sich dabei also um eine Art besseren XML-Editor mit rudimentärer WS-CDL-Unterstützung. Trotz der Unzulänglichkeiten lässt sich ein komplettes WS-CDL-Dokument in dieser Ansicht erstellen, ohne in die andere Ansicht wechseln zu müssen. Bequem ist es jedoch nicht.

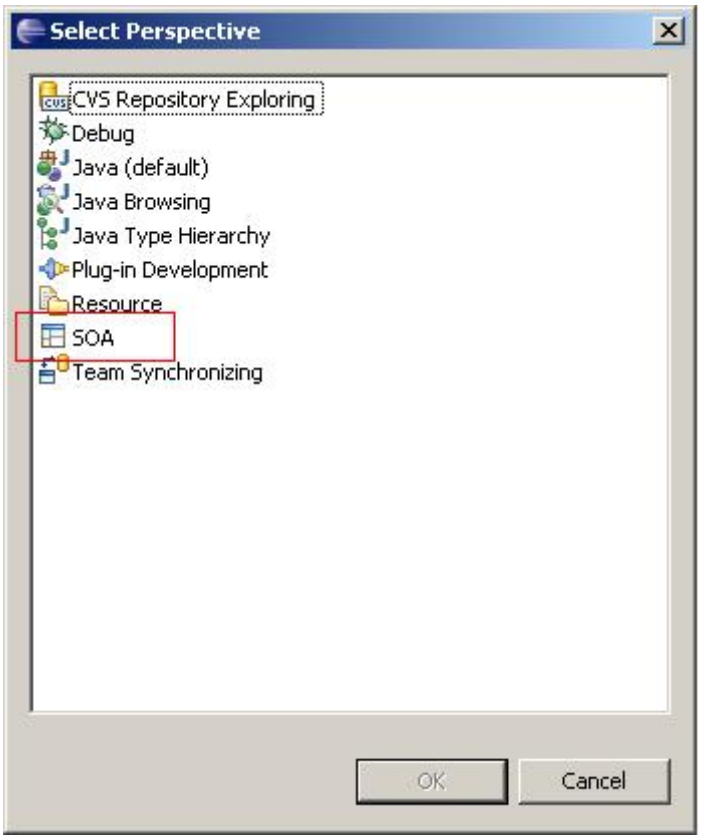

Abbildung 16: SOA Perspective

4.6.4 Graphische Notation

Obwohl eine graphische Notation und somit ein graphischer Editor außerhalb des Scopes von WS-CDL liegt, haben sich die Entwickler von pi4soa daran gemacht eben dieses selber zu realisieren. Es ist in pi4soa möglich, zumindest Teile einer Choreography, in verschiedenen Ansichten graphisch zu modellieren. Andere Teile müssen wiederum in den normalen interaktiven Dialogen erzeugt und befüllt werden(siehe hierzu die folgenden Abbildungen 17, 18, 19).

Im Choreography Description Designer geht das Erstellen von WS-CDL-Dokumenten leichter von der Hand. Der Designer teilt sich in 3 Unteransichten auf, die im Hauptfenster von Eclipse über Tabs erreichbar sind. Dabei sind zwei dieser Ansichten graphische Editoren, eine Ansicht ermöglicht das Editieren einer Baumstruktur.

In der Participants, Roles und Relationsships-Ansicht werden die grundlegenden Verhältnisse des Geschäftsprozesses festgelegt. Hier können die Partner, ihre Interaktionen und die dabei eingenommenen Rollen und Verhaltensweisen graphisch relativ einfach erstellt werden. Dabei entsteht

recht schnell eine gute Übersicht über den Geschäftsprozess. Trotz der einfachen Bedienung gibt es auch in dieser Ansicht einige Nachteile. Die Anordnung der Elemente kann nicht frei gewählt werden, obwohl das Programm das durch die entsprechende Anzeige eines Mauszeigers suggeriert. Alle Elemente werden automatisch und nicht immer sinnvoll angeordnet. Ein weiterer Nachteil ist, dass man zum Eintragen der Details diese von der Bedienung recht angenehme Ansicht recht schnell wieder verlassen muss.

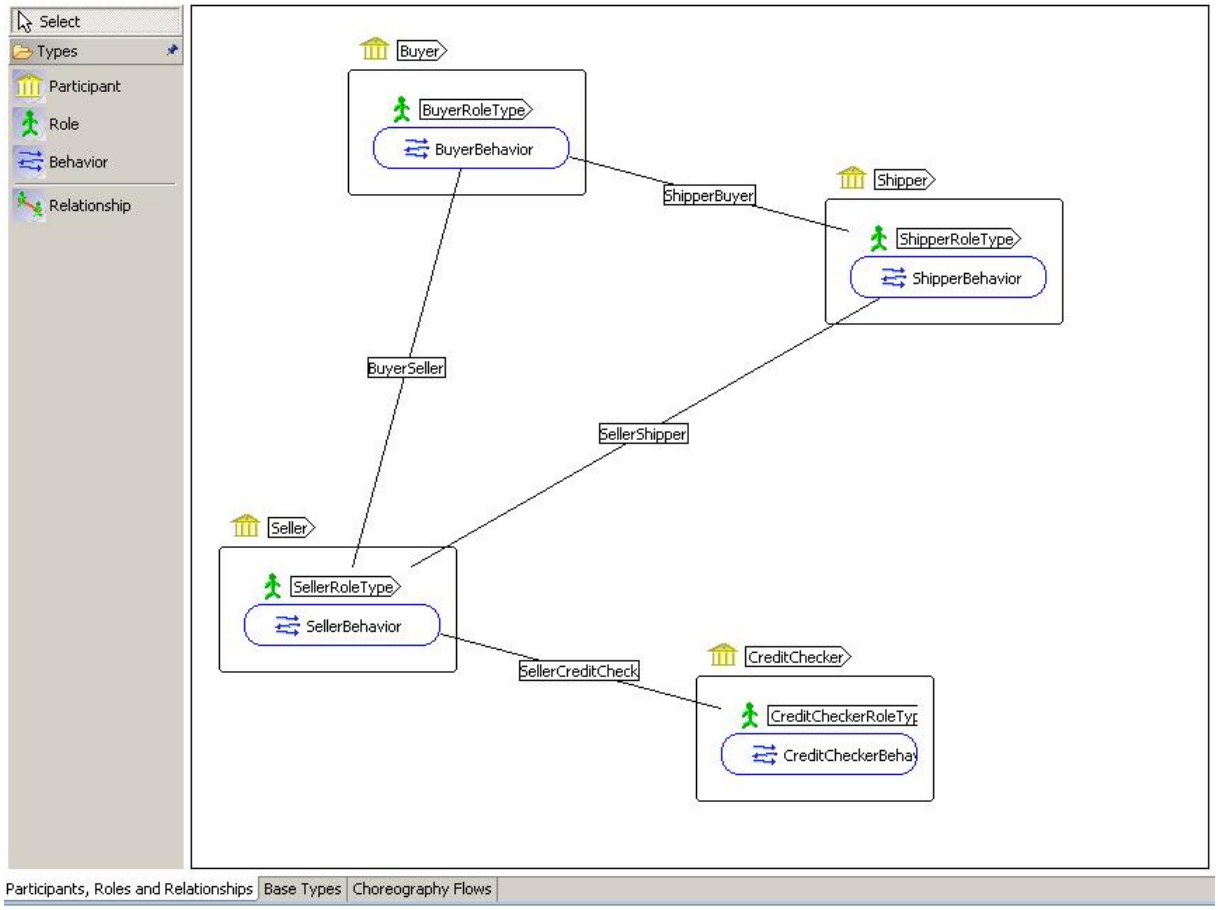

Abbildung 17: Graphische Notation pi4soa (1)

Details kann man dann besser in der Base Types-Ansicht eingeben. Die Bedienung ist dabei der Editor-Ansicht recht ähnlich, es wird eine Baumstruktur bearbeitet. Statt per Rechtsklick kann man dabei direkt die Elemente in die Baumstruktur ziehen und muss dann die Details über die Properties eingeben.

Der Ablauf des Geschäftsprozesses kann dann über die Choreography Flows-Ansicht modelliert werden. Hier ist es möglich, in einer graphischen baumartigen Darstellung den Ablauf zu gestalten. Dabei die verschiedenen Auswahlmöglichkeiten nebeneinander und die sequentiellen Abfolgen untereinander dargestellt. Darüber hinaus wird auch die Kommunikation der verschiedenen Teilnehmer über Channels graphisch visualisiert. In den Properties können weitere Eintragungen zu den einzelnen Attributen der Elemente vorgenommen werden.

Liegt dann ein Geschäftsprozess innerhalb von pi4soa als WS-CDL-Dokument vor, dann fehlt leider die Möglichkeit, dieses gegen die WS-CDL-Spezifikation zu prüfen. Beim Versuch Doku-

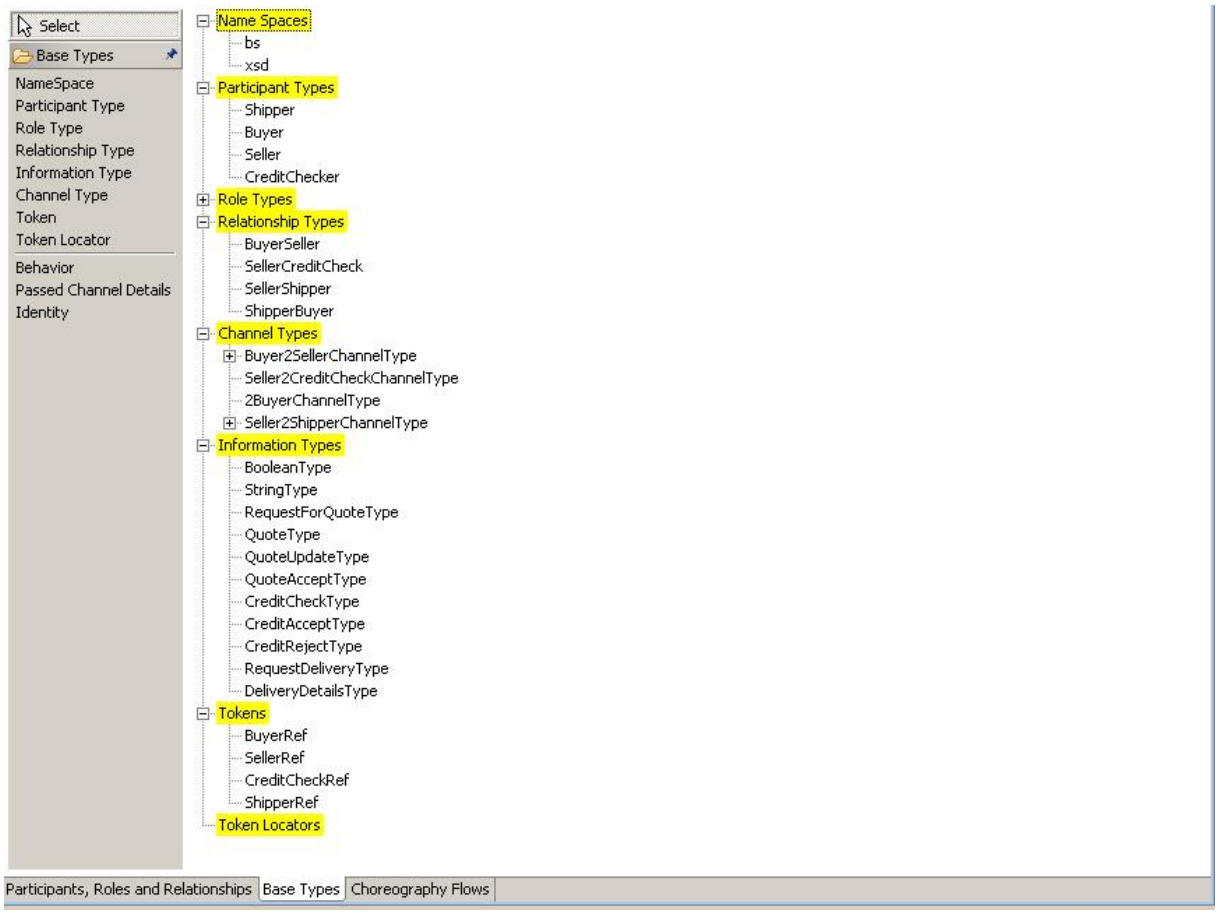

Abbildung 18: Graphische Notation pi4soa (2)

mente mit WS-CDL Eclipse auszutauschen, stellte sich heraus, dass sich zumindest pi4soa nicht konkret an den WS-CDL-Standard hielt. Doch dazu mehr im nächsten Abschnitt. Im Gegensatz dazu ist es möglich, ein oder mehrere Testszenarien zu entwickeln. Dazu gibt es einen eigenen Editor, den Test Scenario Editor. In diesem können verschiedene Gruppen von Ereignissen erstellt werden, die Nachrichten enthalten, die zwischen den einzelnen Teilnehmern ausgetauscht werden. So kann der korrekte oder inkorrekte Ablauf des Geschäftsprozesses simuliert werden. Test man nun eines dieser Szenarien, dann wird in der Eclipse-Console ein Protokoll des Prozesses ausgegeben. Dabei werden Fehler markiert. Es stellt sich jedoch als etwas schwierig heraus dieses Protokoll zu analysieren.

Stabilität

Bei der Benutzung von pi4soa 1.2.0 patch1 vom 23.09.2005 unter Eclipse 3.1.0 mit GEF 3.1 und emf 2.1.0 unter Windows XP traten keine Stabilitätsprobleme auf. Weder Eclipse noch Windows stürzten dabei ab. Das Gestalten eigener einfacher Beispiele in allen Ansichten sowie das Öffnen, Anpassen und Testen der Beispiele, die von den Entwicklern von pi4soa auf ihrer Homepage http://www.pi4tech.com im Downloadbereich (zugänglich nach Registrierung) gelang ohne Probleme. Kleinere Bugs traten bei der Aktualisierung der Baumansicht im Editor auf: Wenn man Attribute über die Properties in der falschenReihenfolge verändert, wurden diese

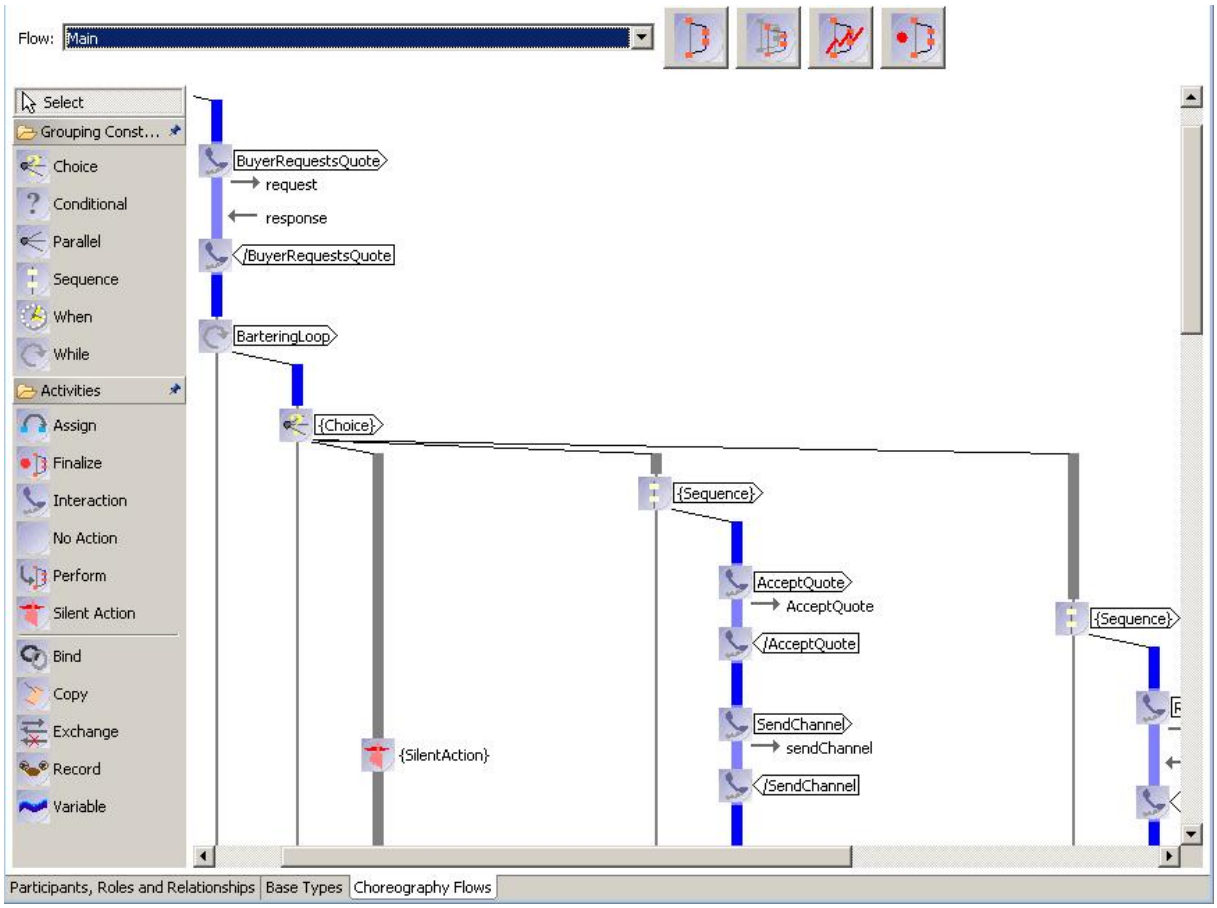

Abbildung 19: Graphische Notation pi4soa (3)

in der Baumansicht nicht mehr angezeigt. Durch neuerliche Eingabe ließ sich dieses Problem umgehen.

Abschliessende Betrachtungen

Der positive Eindruck, den pi4soa bei der Arbeit mit der grafischen Darstellung hinterlässt, wird durch die auffallende Nichtkonformität mit der WS-CDL Spezifikation geschmälert. So verwenden die Entwickler von pi4soa eigene Konstrukte, welche die Element-Tags und Attribute von WS-CDL ersetzen. Somit sind WS-CDL Dokumente, die mit pi4soa erstellt wurden nicht standardkonform und können beispielsweise in WSCDL-Eclipse nicht validiert werden.

Als Beispiel soll folgender Codeschnipsel angeführt werden: Die konforme Zeile

```
<informationType name="RequestForQuoteType" type="RequestForQuote" element=""/>
```

wird vom Tool folgendermaßen umgesetzt

```
<informationTypes name="RequestForQuoteType" typeName="bs:RequestForQuote" elementName=""/>
```

Es ist ersichtlich, dass eigene Elemente und Attributnamen die WS-CDL vorgaben substituieren.

So wird auch ein WS-CDL Package durch die Entwickler von pi4soa redefiniert, indem ein Namespace eingebunden wird. Auch das Element >typeDefinitions> ist in WS-CDL Standard gänzlich unbekannt. Ein solches Dokument kann nur durch das pi4soa-Tool erkannt werden.

```
<org.pi4soa.cdl:Package ...>
  <typeDefinitions>
...
</typeDefinitions>
</org.pi4soa.cdl:Package>
```

Auffallend ist ebenfalls das Fehlen einer reinen XML-Ansicht, bei welcher man das WS-CDL Dokument im Textformat betrachten kann. Zumindest konnte bei der Arbeit mit dem Tool keine Möglichkeit gefunden werden, auf eine solche Ansicht umzuschalten. Lediglich ein Export in ein normales XML-Dokument ist möglich, jedoch fehlen dort dann Strukturformatierungen wie Zeilenumbrüche, welche die bekannte Baumstruktur nachbilden. Man erhält Elemente und Attribute als Fließtext hintereinander gereiht. Somit muss das exportierte Dokument nachbearbeitet werden, um die Elemente passend zu strukturieren.

4.6.5 Tool-Empfehlung

Bei Nutzung beider untersuchter Tools konnte nicht zweifelsfrei geklärt werden, ob es sich bei einem Dokument wirklich um ein WS-CDL konformes, das heißt der verfügbaren Spezifikation entsprechendes, Dokument handelt. Da, wie bereits erwähnt, pi4soa erzeugte Dokumente toolspezifische Konstrukte beinhalten und scheinbar standardkonforme Dokumente in WSCDL-Eclipse zwar als solche validiert werden aber nicht ausgeführt (graphisch dargestellt bzw. simuliert) werden können, bleibt unklar, ob dies bei letzterem an der Konformität oder wirklich an Fehlern im Tool liegt, oder ob jemals mit einem korrekten Dokument gearbeitet wurde.

Aufgrund der gesammelten Erfahrungen im Umgang mit den beiden WS-CDL-Tools können wir nur einen Schluß ziehen, nämlich dass pi4soa zum jetzigen Zeitpunkt das am weitesten entwickelte Tool ist und damit unsere Empfehlung erhält.

4.7 Ausblick WS-CDL

Im Moment bietet WS-CDL nur die Spezifikation einer formalen Definitionssprache zur Schnittstellenbeschreibung von Web Services. Die Entwicklungsarbeit an WS-CDL wird zurzeit nur von einer kleinen Gruppe vorangetrieben. Noch fehlt die Unterstützung durch große Softwarehersteller. Für die allgemeine Akzeptanz innerhalb der Web Service Gemeinde ist es notwendig, dass nicht nur die Entwicklung der Sprache an sich weitergeht, sondern dass auch formalisierte Beispiele und Patterns (wiederverwendbare Lösungsansatze für bekannte und wohldefinierte Probleme bzw. Szenarien), die bisher fast gänzlich fehlen, ausgearbeitet werden. Nur so wird WS-CDL auch für eine breitere Masse verständlich und damit attraktiv. Zielführend wäre die Entwicklung einer offiziellen und formal spezifizierten graphischen Notation, deren Fehlen nicht nur im Rahmen dieser Fachstudie als Hauptschwachpunkt angesehen wird.

5 Szenarien

Ohne konkrete praxisnahe Beispiele lassen sich so formale Definitionssprachen wie BPEL4WS und WS-CDL nur schlecht verstehen und bewerten. Deshalb ist es üblich, dazu verschiedene Szenarien einzusetzen, um das Verhalten in der Praxis und ein Vergleich der verschiedenen Möglichkeiten und Einsatzgebiete zu untersuchen.

Dabei sollen die Szenarien in deutlich verschieden sein, um damit ein möglichst breites Anwendungsgebiet abzudecken. Ebenso sollen altgediente Praxisbeispiele zum Zug kommen, an denen sich schon BPEL4WS beweisen musste. Ein wichtiges Kriterium ist die Reichweite"des Szenarios. Handelt es sich um einen internen Geschäftsprozess oder um eine Interaktion mehrer Partner aus unterschiedlichen Unternehmen. Wichtig ist auch noch, ob dabei mehrere Web Services beteiligt sind oder es sich nur um einen einzelnen Web Service handelt, der modelliert werden soll.

Dabei darf man jedoch nicht vergessen, dass die Szenarien nur beispielhaft für einen bestimmten Anwendungsbereich stehen und es sich nicht um fertige Beispiele handelt, die ausführbar sind oder so direkt als Vorlage eingesetzt werden können.

Zu der Beschreibung eines einzelnen Szenarios gehört die natürlichsprachliche Beschreibung der beteiligten Partner, der Abläufe und der Kommunikation. Zur besseren Visualisierung werden die Szenarien auch als Ablaufdiagramme dargestellt. Schließlich sollen die Szenarien sowohl in BPEL4WS und WS-CDL umgesetzt, d.h. beschrieben werden, falls dies möglich ist.

Wie der Titel dieser Fachstudie bereits thematisiert sollte der Teil der Szenarien den Kernpunkt dieser Fachstudie darstellen. Dies ist nur bedingt umsetzbar, da eine direkte Entwicklung von Szenarien ohne eine Einarbeitung in die Technologien/Sprachen selbst keinen Sinn macht und diese im Umfang nicht zu unterschätzen ist.

5.1 Ansätze zum Erzeugen bzw. Finden von Szenarien

Es ist nicht leicht als „Neuling“ in einer Domäne Szenarien aus dem Nichts zu erschaffen. Deshalb wurden zuerst die Kerndokumente (Spezifikationen) der beiden zu betrachtenden Sprachen betrachtet und die darin enthaltenen Szenarien bzw. Beispiele (soweit vorhanden) zu rate gezogen. Innerhalb des Bereichs Web Services konnten wir weiterhin nur auf eine sehr kleine Menge an Standardbeispielen (Reisebuchung in verschiedenen Varianten, Kreditgenehmigung, generelles Buyer-Seller-Szenario) zurückgreifen.

Ein anderer Ansatzpunkt war das „Crossover“-Prinzip: Man nehme ein Szenario, dass in einer der beiden Sprachen bereits vorhanden ist und versuche dieses ohne Änderung der Betrachtung in der anderen Sprache umzusetzen. Dieser Ansatz ist deshalb nur teilweise praktikabel, da die Betrachtungsweisen von WS-CDL und BPEL4WS sehr verschieden sind.

Ein dritter Ansatzpunkt war das formulieren von „Real-Life“-Beispielen und der Versuch diese umzusetzen.

Eine Mischung aus diesen drei Ansatzpunkten führte dann schließlich zu den nachfolgend aufgeführten Szenarien, anhand derer Stärken und Schwächen der Sprachen herausgearbeitet wurden.

5.2 Granularität der Szenarien / Sichten auf Szenarien

Im Verlauf der Fachstudie wurde deutlich, dass diverse Szenarien je nach Grad der Abstraktion bzw. des Betrachtungswinkels generell mit beiden Sprachen umsetzbar sind (siehe hierzu auch 6.1).

Wenn man die verfügbaren bzw. ausdenkbaren Beispiele bzw. Szenarien immer weiter abstrahiert landet man immer wieder bei einem abstrakten Top-Level-Modell, dass sich zwischen zwei Partnern abspielt, die wiederum wieder mit anderen Partnern verbunden sein können (siehe Abbildung 20).

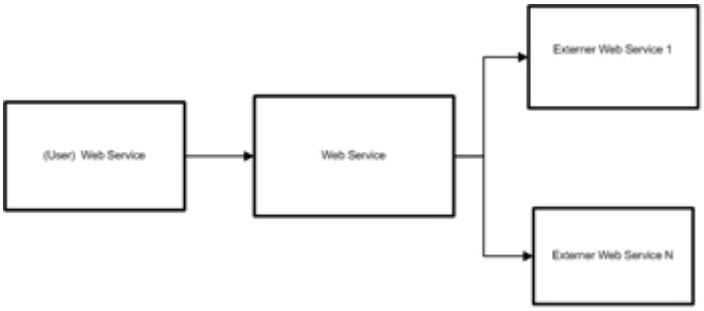

Abbildung 20: Höchste Abstraktion

5.3 Szenario 1 - Reisebuchung

5.3.1 Einleitung

Das erste Szenario „Reise buchen“ ist eines der Standardbeispiele für die Betrachtung von Webservices, abzulesen u.a. an der Nennung als Use Case in den Requirements von WS-CDL. Zu Beginn der Betrachtung wurde ein relativ komplexes Szenario in teilformalisierter Form erstellt (siehe **Erstes Herantasten**), dieses wurde im Laufe der Fachstudie jedoch in ein simpleres Szenario überführt, um die Komplexität zu verringern und die Umsetzung zu erleichtern (Einige vor-, zwischen- und nachgelagerte Prozesse wie die Hotelauswahl, Flugauswahl, Rechnungsstellung, Abrechnung, etc wurden abstrahiert, siehe hierzu auch im Anhang 7.1). Ein weiterer Grund für die Verringerung der Komplexität dieses Szenarios war, dass dieses Beispiel auch schon als BPEL4WS-Implementierung vorliegt (Fachstudie Nr. 38 [25]).

5.3.2 Beschreibung des Szenarios

Es handelt sich bei diesem Szenario um eine sehr einfache Reisebuchung, die mit der realen Buchung in einem Reisebüro relativ wenig zu tun hat. Basierend auf den Eingaben eines Benutzers von einem Reisebüro für einen angegebenen Zeitraum ein Hotelzimmer am Zielort gebucht. Gleichzeitig wird am Anreisetag ein Flug zum Zielort und am Rückreisetag ein Rückflug gebucht. Wenn beide Vorgänge erfolgreich sind, dann erhält der Benutzer eine Bestätigung und bekommt den Gesamtpreis der beiden Buchungen mitgeteilt.

5.3.3 Ablauf

Zu Beginn erwartet der Travel-Prozess (travel.wsdl, travelInterface.wsdl, BPEL: bookingIntegration.bpel) die Eingabe einer Personalnummer (BPEL), sowie ein Datum für die An- und Abreise (BPEL und WS-CDL). Mit diesen Informationen wird durch einen externen Webservice (hotel.wsdl) ein Hotelzimmer für den angegebenen Zeitraum gebucht. Nach erfolgreicher Hotelbuchung gibt der Webservice den Preis zurück. Dann wird über einen zweiten externen Webservice (flight.wsdl) ein Flug am An- und Abreisedatum gebucht. Auch hier bei Erfolg der Preis zurückgegeben. Waren die Hotelbuchung und die Flugbuchung erfolgreich, wird der Gesamtpreis beider Buchungen berechnet und der Benutzer erhält eine Bestätigung mit der Ausgabe des Preises. Falls bei der Buchung des Fluges ein Fehler auftritt (z.B. kein Flug verfügbar), wird die Hotelbuchung storniert und der Vorgang abgebrochen (BPEL). Wird der gesamte Prozess innerhalb von zehn Minuten nicht beendet, dann tritt ein Timeout auf. Eine bis dahin getätigte Buchung des Hotelzimmers, bzw. Fluges wird automatisch storniert (BPEL).

5.3.4 BPEL

siehe hierzu Kapitel 7.2.2 Anhang.

5.3.5 WS-CDL

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://www.w3.org/2005/08/ws-chor/cdl"
xmlns:tns="http://fachstudie.ingoal.info/TravelService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
author="Fachstudie" name="TravelService"
targetNamespace="http://fachstudie.ingoal.info/TravelService" version="1.0">
<informationType name="TravelData" type="xsd:string"/>
<informationType name="PriceHotel" type="xsd:int"/>
<informationType name="PriceAirline" type="xsd:int"/>
<informationType name="String" type="xsd:string"/>
<informationType name="Integer" type="xsd:int"/>
<informationType name="URI" type="xsd:anyURI"/>
<token informationType="String" name="sendBillt"/>
<token informationType="URI" name="hotelRef"/>
<token informationType="URI" name="airlineRef"/>
<token informationType="String" name="offerConfirmation"/>
<token informationType="TravelData" name="TravelRequest"/>
<roleType name="Client">
<behavior name="client"/>
</roleType>
<roleType name="ServiceEmpHotel">
<behavior name="serviceEmpHotel"/>
</roleType>
<roleType name="ServiceEmpAirline">
<behavior name="serviceEmpAirline"/>
</roleType>
<roleType name="ServiceEmpTravel">
<behavior name="serviceEmpTravel"/>
</roleType>
<relationshipType name="ServiceEmpTravel-ServiceEmpHotel">
<roleType typeRef="ServiceEmpTravel"/>
<roleType typeRef="ServiceEmpHotel"/>
</relationshipType>
<relationshipType name="ServiceEmpTravel-ServiceEmpAirline">
<roleType typeRef="ServiceEmpTravel"/>
<roleType typeRef="ServiceEmpAirline"/>
</relationshipType>
<relationshipType name="ServiceEmpTravel-Client">
<roleType typeRef="ServiceEmpTravel"/>
<roleType typeRef="Client"/>
</relationshipType>
<channelType name="TravelChannel" usage="shared">
<roleType typeRef="ServiceEmpTravel"/>
<reference>
<token name="TravelRequest"/>
</reference>
</channelType>
<channelType name="OfferConfirmationChannel">
<roleType typeRef="Client"/>
<reference>
<token name="offerConfirmation"/>
```

```
</reference>
</channelType>
<channelType name="sendBillChannel">
<roleType typeRef="Client"/>
<reference>
<token name="sendBillt"/>
</reference>
</channelType>
<channelType name="AirlineChannel">
<roleType typeRef="ServiceEmpAirline"/>
<reference>
<token name="airlineRef"/>
</reference>
</channelType>
<channelType name="HotelChannel">
<roleType typeRef="ServiceEmpHotel"/>
<reference>
<token name="hotelRef"/>
</reference>
</channelType>
<choreography name="TravelBooking" root="true">
<relationship type="ServiceEmpTravel-ServiceEmpHotel"/>
<relationship type="ServiceEmpTravel-ServiceEmpAirline"/>
<relationship type="ServiceEmpTravel-Client"/>
<variableDefinitions>
<variable channelType="sendBillChannel" name="sendBillChannel"/>
<variable channelType="OfferConfirmationChannel" name="offerConfirmationChannel"/>
<variable channelType="HotelChannel" name="hotelChannel"/>
<variable channelType="AirlineChannel" name="airlineChannel"/>
<variable channelType="TravelChannel" name="travelChannel"/>
<variable channelType="AirlineChannel" name="flightConfirmation"/>
<variable channelType="HotelChannel" name="roomConfirmation"/>
<variable channelType="sendBillChannel" name="bill"/>
<variable informationType="TravelData"
name="travelRequest" roleTypes="Client ServiceEmpTravel"/>
<variable channelType="AirlineChannel" name="flightRequest"/>
<variable channelType="HotelChannel" name="roomRequest"/>
<variable informationType="TravelData" name="TravelOffer"/>
<variable channelType="OfferConfirmationChannel"
name="offerConfirmation" roleTypes="Client ServiceEmpTravel"/>
</variableDefinitions>
<sequence>
<interaction channelVariable="travelChannel" initiate="true"
name="getTravelOffer" operation="handleOfferRequest">
<participate fromRoleTypeRef="Client"
relationshipType="ServiceEmpTravel-Client" toRoleTypeRef="ServiceEmpTravel"/>
<exchange action="request" informationType="TravelData" name="travelDataRequest">
<send variable="cdl:getVariable('travelRequest','','')"/>
<receive variable="cdl:getVariable('travelRequest','','')"/>
</exchange>
<exchange action="respond" informationType="TravelData" name="travelDataRespond">
<send variable="cdl:getVariable('TravelOffer','','')"/>
<receive variable="cdl:getVariable('TravelOffer','','')"/>
</exchange>
```

```
</interaction>
<interaction channelVariable="hotelChannel"
name="requestHotel" operation="handleHotelRoom">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-ServiceEmpHotel"
toRoleTypeRef="ServiceEmpHotel"/>
<exchange action="request" name="roomRequest">
<send variable="cdl:getVariable('roomRequest','','')"/>
<receive variable="cdl:getVariable('roomRequest','','')"/>
</exchange>
</interaction>
<interaction channelVariable="hotelChannel"
name="confirmationHotel" operation="confirmHotelRoom">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-ServiceEmpHotel"
toRoleTypeRef="ServiceEmpHotel"/>
<exchange action="respond" name="roomResponse">
<send variable="cdl:getVariable('roomConfirmation','','')"/>
<receive variable="cdl:getVariable('roomConfirmation','','')"/>
</exchange>
</interaction>
<interaction channelVariable="airlineChannel"
name="requestFlight" operation="handleFlight">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-ServiceEmpAirline"
toRoleTypeRef="ServiceEmpAirline"/>
<exchange action="request" name="flightRequest">
<send variable="cdl:getVariable('flightRequest','','')"/>
<receive variable="cdl:getVariable('flightRequest','','')"/>
</exchange>
</interaction>
<interaction channelVariable="airlineChannel"
name="confirmationFlight" operation="confirmFlight">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-ServiceEmpAirline"
toRoleTypeRef="ServiceEmpAirline"/>
<exchange action="respond" name="flightRespond">
<send variable="cdl:getVariable('flightConfirmation','','')"/>
<receive variable="cdl:getVariable('flightConfirmation','','')"/>
</exchange>
</interaction>
<interaction channelVariable="offerConfirmationChannel"
name="confirmTravelOffer" operation="handleOfferConfirmation">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-Client" toRoleTypeRef="Client"/>
<exchange action="respond" name="offerConfirmation">
<send variable="cdl:getVariable('offerConfirmation','','')"/>
<receive variable="cdl:getVariable('offerConfirmation','','')"/>
</exchange>
</interaction>
<interaction channelVariable="sendBillChannel" name="sendBill"
operation="handleBilling">
<participate fromRoleTypeRef="ServiceEmpTravel"
relationshipType="ServiceEmpTravel-Client" toRoleTypeRef="Client"/>
```

```
<exchange action="respond" name="writeSendBill">
<send variable="cdl:getVariable('bill','','')"/>
<receive variable="cdl:getVariable('bill','','')"/>
</exchange>
</interaction>
</sequence>
</choreography>
</package>
```

5.3.6 Bemerkungen

Der abgebildete Code zeigt die Umsetzung des Szenarios Reise buchenïn WS-CDL. Das Beispiel wurde den Anforderungen entsprechend abstrahiert. Folglich wurden hauptsächlich Grundelemente der Sprache verwendet, weiterführende Möglichkeiten wie Anbindungen an WSDL Interfaces, Information Alignment und tiefgehende Detaillierung blieben damit ungenutzt. Auch waren hierfür keinerlei Beispiele zu finden. Eine eventuelle Umsetzung hätte das Beispiel völlig unüberschaubar gemacht. Das Bestreben war, eine möglichst leicht verständliche Choreographie zu modellieren. Verzichtet wurde auf die Einbindung von Exceptions und FinalizerBlocks, da diese Konstrukte zugegebenermaßen recht komplex erschienen und in der verfügbaren Zeit nicht zufrieden stellend umgesetzt werden konnten. Im Code ist eine Sequenz von Interaktionen, welche die Nachrichtenübermittlung darstellen, zu finden. Diese Sequenz legt die Reihenfolge des Ablaufs der nacheinander auszuführenden Interaktionen fest. Natürlich ist der dargestellte Code noch optimierbar. Die Anfrage an das Hotel und an die Fluggesellschaft könnte parallel erfolgen, welches durch das Element <parallel> umsetzbar wäre. Auch kann direkt bei Absage beziehungsweise Nicht-Antworten (da eine Absage nicht implementiert ist) einer der beiden Services direkt eine Absage an den Anfrager erfolgen. Die Antwort des zweiten Services wäre dann redundant, da beide Anfragen erfolgreich sein müssen. Genau hier ist dann der Einsatz von Exceptions sinnvoll, um etwaige Fehlerzustände abzufangen.

5.4 Szenario 2 - Kreditgenehmigung

5.4.1 Abstract

Das Szenario „Kreditgenehmigung“ enstand angelehnt an die Übungen der Vorlesung Workflow-Management [Leymann, 2005] und die BPEL-Spezifikation [Leymann et al, 2005] und wurde für die Zwecke dieser Fachstudie etwas modifiziert.

5.4.2 Ablauf

Mit jeder Kundenanfrage wird eine neue Prozessinstanz erzeugt. Bei einer Anfrage erhält der Prozess über die receive-Aktivität persönliche Kundeninformationen und die Summe des Kreditantrags. Ist diese Summe kleiner als $10000, wird die invoke-Aktivität „LoanAssessor“ aufgerufen und das Kundenrisiko wird ermittelt. Bei einem Risiko „high“ wird die invoke-Aktivität „LoanApprover“ aufgerufen. Ist das Risiko „low“ wird der Antwortvariablen mit der assign-Aktivität „AssignYesToAccept“ der Ausdruck „Yes“ zugewiesen. Der Kredit ist damit automatisch genehmigt und der Kunde bekommt über die reply-Aktivität eine positive Antwortnachricht. Ist die Kreditsumme größer oder gleich $10000, wird die invoke-Aktivität „LoanApprover“ sofort aufgerufen, ohne den Umweg über den LoanAssessor zu gehen. Der LoanApprover untersucht den Antrag. Dem Ergebnis entsprechend bekommt der Kunde über die reply-Aktivität eine Antwort auf den Kreditantrag. Der LoanApprover ist ein asynchroner Dienst. Ist die Antwort vom Approver nach einer bestimmten Zeit noch nicht verfügbar, bekommt der Kunde ein Antwort, dass die Prüfung nicht in der maximal vorgesehenen Zeit möglich war. Tritt im Prozess ein Fehler auf, wird dieser im faultHandler von einem catchAll-Block abgefangen. Anschließend wird eine reply-Aktivität aufgerufen und der Kunde erhält eine Fehlernachricht als Antwort. (BPEL)

5.4.3 BPEL

- siehe hierzu Kapitel 7.2.1 im Anhang

5.4.4 WS-CDL

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://www.w3.org/2005/08/ws-chor/cdl"
xmlns:tns="http://fachstudie.ingoal.info/CreditDecider"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="CreditDecider"
targetNamespace="fachstudie.ingoal.info/creditDecider" version="1.0">
<informationType name="CreditRequest" type="xsd:String"/>
<informationType name="CreditResponse" type="xsd:String"/>
<informationType name="CreditApprovalRequest" type="xsd:String"/>
<informationType name="CreditApprovalResponse" type="xsd:String"/>
<token informationType="CreditRequest" name="loanToken"/>
<token informationType="CreditApprovalRequest" name="expertToken"/>
<token informationType="CreditResponse" name="approveToken"/>
<roleType name="Client">
<behavior name="client"/>
</roleType>
<roleType name="Employee">
```

```
<behavior name="employee"/>
<behavior name="creditRequester"/>
</roleType>
<roleType name="Expert">
<behavior name="expert"/>
</roleType>
<relationshipType name="Employee-Expert">
<roleType behavior="creditRequester" typeRef="Employee"/>
<roleType behavior="expert" typeRef="Expert"/>
</relationshipType>
<relationshipType name="Client-Employee">
<roleType typeRef="Client"/>
<roleType behavior="employee" typeRef="Employee"/>
</relationshipType>
<participantType name="Bank">
<roleType typeRef="Expert"/>
<roleType typeRef="Employee"/>
</participantType>
<channelType action="request" name="requestLoanChannel">
<roleType behavior="employee" typeRef="Employee"/>
<reference>
<token name="loanToken"/>
</reference>
</channelType>
<channelType action="request" name="requestExpertHelp">
<roleType typeRef="Expert"/>
<reference>
<token name="expertToken"/>
</reference>
</channelType>
<channelType action="respond" name="responseLoanChannel">
<roleType typeRef="Client"/>
<reference>
<token name="approveToken"/>
</reference>
</channelType>
<choreography name="CreditRequest" root="true">
<relationship type="Employee-Expert"/>
<relationship type="Client-Employee"/>
<variableDefinitions>
<variable channelType="requestLoanChannel" name="requestChannel"/>
<variable channelType="requestExpertHelp" name="expertChannel"/>
<variable channelType="responseLoanChannel" name="responseChannel"/>
<variable informationType="CreditRequest" name="loanRequest"/>
<variable informationType="CreditResponse" name="loanResponse"/>
<variable informationType="CreditApprovalRequest" name="expertRequest"/>
<variable informationType="CreditApprovalResponse" name="expertResponse"/>
</variableDefinitions>
<sequence>
<interaction channelVariable="requestChannel"
name="getLoan" operation="handleLoanRequest">
<participate fromRoleTypeRef="Client"
relationshipType="Client-Employee" toRoleTypeRef="Employee"/>
<exchange action="request" informationType="CreditRequest" name="loanRequest">
```

```
<send variable="cdl:getVariable('loanRequest','','')"/>
<receive/>
</exchange>
</interaction>
<choice>
<interaction channelVariable="responseChannel"
name="approveLoan" operation="handleLoanResonse">
<participate fromRoleTypeRef="Employee"
relationshipType="Client-Employee" toRoleTypeRef="Client"/>
<exchange action="respond"
informationType="CreditResponse" name="loanResponse">
<send/>
<receive variable="cdl:getVariable('loanResponse','','')"/>
</exchange>
</interaction>
<sequence>
<interaction channelVariable="expertChannel"
name="askExpert" operation="handleEmployeeRequest">
<participate fromRoleTypeRef="Employee"
relationshipType="Employee-Expert" toRoleTypeRef="Expert"/>
<exchange action="request" informationType="CreditApprovalRequest"
name="getApproval">
<send variable="cdl:getVariable('expertRequest','','')"/>
<receive/>
</exchange>
<exchange action="respond" informationType="CreditApprovalResponse"
name="sendApproval">
<send/>
<receive variable="cdl:getVariable('expertResponse','','')"/>
</exchange>
</interaction>
<interaction channelVariable="responseChannel" name="loanReply"
operation="handleLoanReply">
<participate fromRoleTypeRef="Employee"
relationshipType="Client-Employee" toRoleTypeRef="Client"/>
<exchange action="respond" informationType="CreditResponse"
name="loanResponse">
<send/>
<receive variable="cdl:getVariable('loanResponse','','')"/>
</exchange>
</interaction>
</sequence>
</choice>
</sequence>
</choreography>
</package>
```

5.4.5 Bemerkungen

Damit eine Umsetzung in WS-CDL überhaupt möglich ist, musste, wie bei den meisten Beispielen, eine kleine Änderung erfolgen. Prozesse, wie sie in BPEL umgesetzt werden, sind nicht direkt in WS-CDL umsetzbar, da interne Prozesse, bei denen kein Informationsaustausch zwischen

Parteien stattfindet, nicht darstellbar sind. So ist beispielsweise eine Modellierung des Kunden notwendig, der eine Anfrage an den entsprechenden Sachbearbeiter bei der Bank stellt. Die internen Abläufe der Bank wurden auf zwei zu modellierende Partner abstrahiert. So entscheidet der Sachbearbeiter selbst über die Kreditvergabe bis zu einer vereinbarenden Summe. Ab dieser ist ein Kreditexperte hinzuzuziehen. Da solche Bedingungen in WS-CDL nicht direkt umsetzbar sind und nur mit Hilfskonstrukten zu realisieren, wurde darauf verzichtet. Eine denkbare Lösung wäre der Einsatz einer Exception, welche die Antwortvariable solange schützt (<block> - Element in Verbindung mit dem <guard>, der unter der bestimmten Bedingung ausgelöst wird), bis der Kreditexperte eine Entscheidung gefällt hat oder ein Timout aufgetreten ist. Erst danach ist die Variable verfügbar und auf die Anfrage des Antragstellers kann geantwortet werden. Die Umsetzung scheiterte hier ebenfalls an der scheinbar nicht trivial lösbaren Komplexität.

5.5 Szenario 3 - Buyer/Seller [26/27]

5.5.1 Abstract

Das BuyerSeller-Szenario stammt aus dem Beispiel-Paket der pi4soa-Entwickler und stellt ein einfaches Beispiel für einen Geschäftsprozess mit mehreren beteiligten Partnern dar. Beteiligt sind dabei als Teilnehmer der Käufer (Buyer), der Verkäufer (Seller), der Versender (Shipper) und die Bank (Credit Checker). Interaktionen finden dabei zwischen dem Kunden und dem Verkäufer, zwischen dem Verkäufer und dem Versender, zwischen dem Verkäufer und der Bank und zwischen dem Versender und dem Kunden statt.

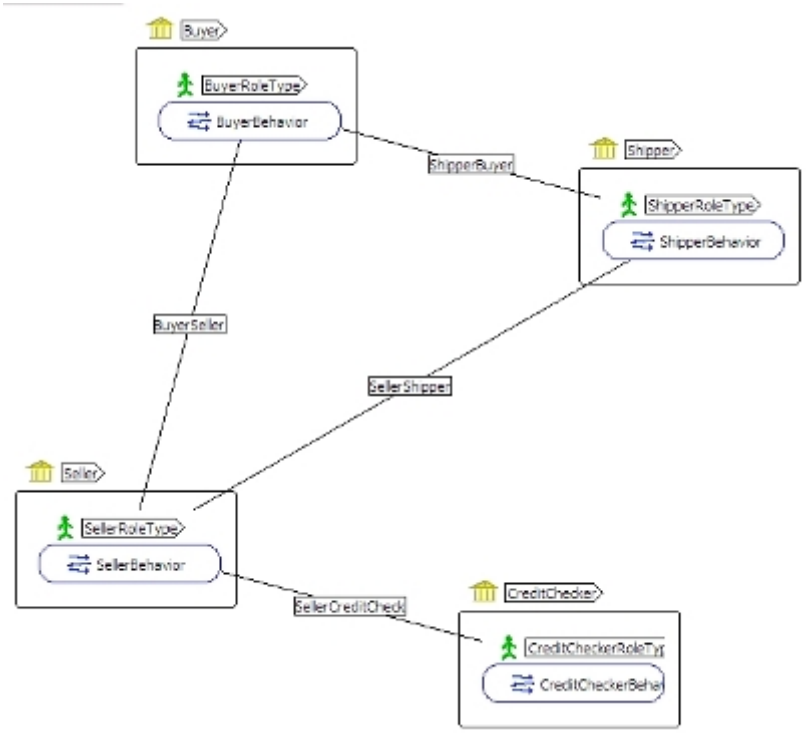

Abbildung 21: Participants, Roles and Relationships

5.5.2 Ablauf

Auch der Ablauf des Geschäftsprozesses lässt sich einfach beschreiben. Zuerst stellt der Käufer eine Anfrage nach einem Produkt an den Verkäufer und macht ihm ein Preisangebot. Dieser sendet ihm als Antwort ein Angebot mit einem Preis für das Produkt zu. Jetzt kann der Käufer entweder das Preisangebot des Verkäufers annehmen oder nachverhandeln. Dann wird so lange nachverhandelt, bis der Käufer entweder den Preis akzeptiert oder den Kauf abbricht. Nimmt der Käufer das Preisangebot an, dann sendet er eine Bestellung an den Verkäufer. Dieser überprüft bei der Bank die Kreditwürdigkeit des Käufers. Wenn dieser Kreditwürdig ist, dann sendet er einen Auftrag zum Versenden des Produkts an den Versender. Dieser verschickt das Produkt an den Kunden.

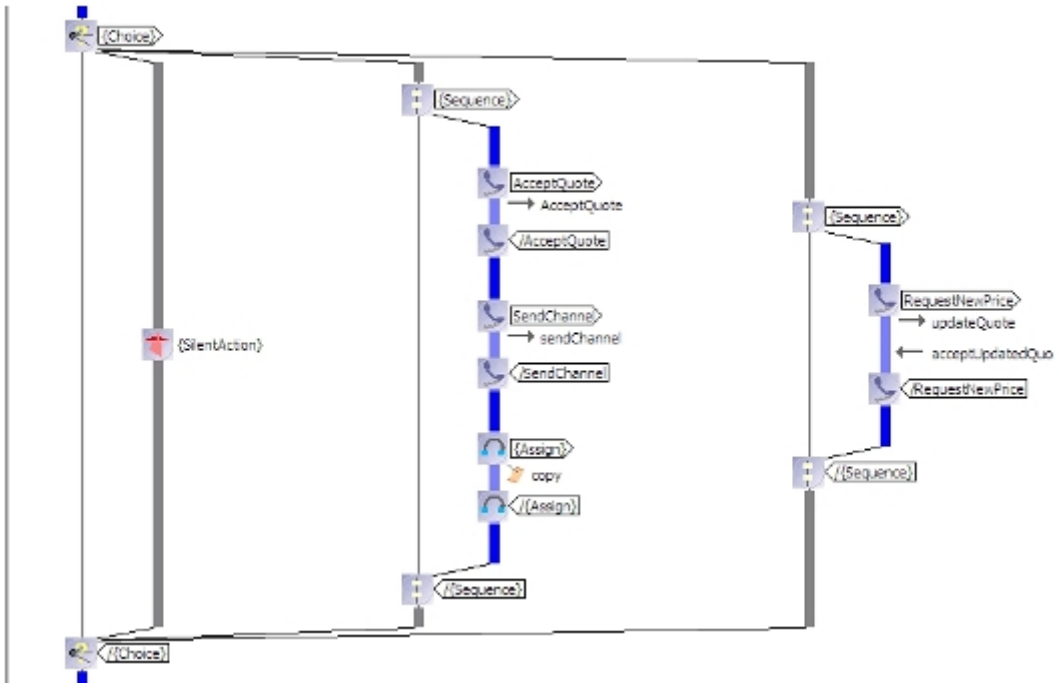

Abbildung 22: Ausschnitt aus den Choreography Flows

5.5.3 Schematische Darstellung

In einer schematischen Darstellung werden die einzelnen Interaktionen zwischen den Partnern deutlich (siehe Abbildung 23, 24, 25 auf den folgenden Seiten).

5.5.4 BPEL

Dieses Beispiel ist in nicht umgesetzt, da der Schwerpunkt auf der Suche nach geeigneten Beispielen und deren möglich Umsetzung in WS-CDL liegt. In BPEL4WS ist dieses Beispiel aus der Sicht eines beliebigen Teilnehmers problemlos umsetzbar.

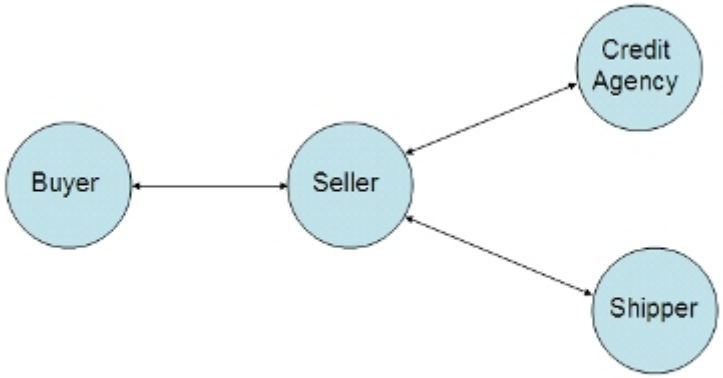

Abbildung 23: Schematische Darstellung der Interaktionen

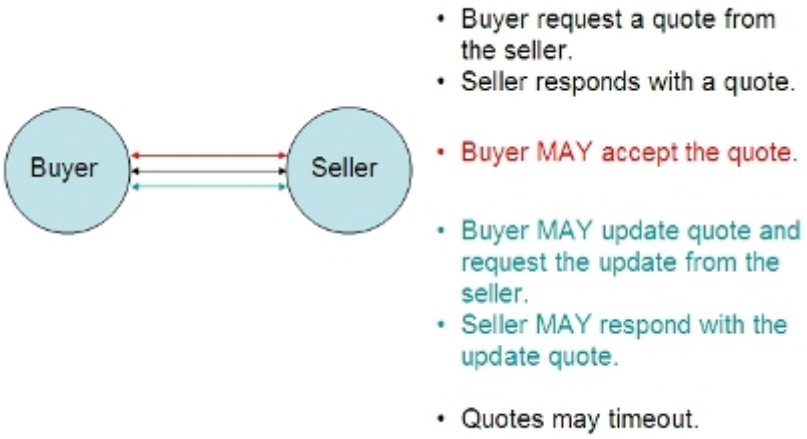

Abbildung 24: Interaktion zwischen Käufer und Verkäufer

5.5.5 WS-CDL

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://www.w3.org/2005/08/ws-chor/cdl"
xmlns:bs="http://www.pi4tech.com/cdl/BuyerSellerExample-1"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" author="Steve Ross-Talbot"
name="BuyerSellerCDL" targetNamespace="http://www.pi4tech.com/cdl/BuyerSeller" version="1.1">
<informationType name="BooleanType" type="xsd:boolean"/>
<informationType name="StringType" type="xsd:string"/>
<informationType name="RequestForQuoteType" type="bs:RequestForQuote"/>
<informationType name="QuoteType" type="bs:Quote"/>
<informationType name="QuoteUpdateType" type="bs:QuoteUpdate"/>
<informationType name="QuoteAcceptType" type="bs:QuoteAccept"/>
<informationType name="CreditCheckType" type="bs:CreditCheckRequest"/>
<informationType name="CreditAcceptType" type="bs:CreditAccept"/>
<informationType name="CreditRejectType" type="bs:CreditReject"/>
```

Abbildung 25: Abschluss der Interaktionen

```
<informationType name="RequestDeliveryType" type="bs:RequestForDelivery"/>
<informationType name="DeliveryDetailsType" type="bs:DeliveryDetails"/>
<token informationType="StringType" name="BuyerRef"/>
<token informationType="StringType" name="SellerRef"/>
<token informationType="StringType" name="CreditCheckRef"/>
<token informationType="StringType" name="ShipperRef"/>
<roleType name="BuyerRoleType">
<behavior name="BuyerBehavior"/>
</roleType>
<roleType name="SellerRoleType">
<behavior name="SellerBehavior"/>
</roleType>
<roleType name="CreditCheckerRoleType">
<behavior name="CreditCheckerBehavior"/>
</roleType>
<roleType name="ShipperRoleType">
<behavior name="ShipperBehavior"/>
</roleType>
<relationshipType name="BuyerSeller">
<roleType typeRef="BuyerRoleType"/>
<roleType typeRef="SellerRoleType"/>
</relationshipType>
<relationshipType name="SellerCreditCheck">
<roleType typeRef="SellerRoleType"/>
<roleType typeRef="CreditCheckerRoleType"/>
</relationshipType>
<relationshipType name="SellerShipper">
<roleType typeRef="SellerRoleType"/>
<roleType typeRef="ShipperRoleType"/>
</relationshipType>
<relationshipType name="ShipperBuyer">
<roleType typeRef="ShipperRoleType"/>
<roleType typeRef="BuyerRoleType"/>
</relationshipType>
```

```
<participantType name="Shipper">
<roleType typeRef="ShipperRoleType"/>
</participantType>
<participantType name="Buyer">
<roleType typeRef="BuyerRoleType"/>
</participantType>
<participantType name="Seller">
<roleType typeRef="SellerRoleType"/>
</participantType>
<participantType name="CreditChecker">
<roleType typeRef="CreditCheckerRoleType"/>
</participantType>
<channelType name="Buyer2SellerChannelType">
<passing channel="2BuyerChannelType" new="true"/>
<roleType typeRef="SellerRoleType"/>
<reference>
<token name="SellerRef"/>
</reference>
</channelType>
<channelType name="Seller2CreditCheckChannelType">
<roleType typeRef="CreditCheckerRoleType"/>
<reference>
<token name="CreditCheckRef"/>
</reference>
</channelType>
<channelType action="request" name="2BuyerChannelType">
<roleType typeRef="BuyerRoleType"/>
<reference>
<token name="BuyerRef"/>
</reference>
</channelType>
<channelType name="Seller2ShipperChannelType">
<passing channel="2BuyerChannelType"/>
<roleType typeRef="ShipperRoleType"/>
<reference>
<token name="ShipperRef"/>
</reference>
</channelType>
<choreography name="Main" root="true">
<relationship type="BuyerSeller"/>
<relationship type="SellerCreditCheck"/>
<relationship type="SellerShipper"/>
<relationship type="ShipperBuyer"/>
<variableDefinitions>
<variable channelType="Buyer2SellerChannelType" name="Buyer2SellerC"
roleTypes="BuyerRoleType"/>
<variable channelType="Seller2ShipperChannelType" name="Seller2ShipperC"
roleTypes="SellerRoleType"/>
<variable channelType="Seller2CreditCheckChannelType" name="Seller2CreditChkC"
roleTypes="SellerRoleType"/>
<variable channelType="2BuyerChannelType" name="DeliveryDetailsC"
roleTypes="BuyerRoleType SellerRoleType ShipperRoleType"/>
<variable informationType="BooleanType" name="barteringDone"
roleTypes="BuyerRoleType SellerRoleType"/>
```

```
</variableDefinitions>
<sequence>
<interaction channelVariable="Buyer2SellerC" initiate="true"
name="BuyerRequestsQuote" operation="requestForQuote">
<description type="documentation">Buyer requests a Quote - this is the initiator</description>
<participate fromRoleTypeRef="BuyerRoleType" relationshipType="BuyerSeller"
toRoleTypeRef="SellerRoleType"/>
<exchange action="request" informationType="RequestForQuoteType"
name="request">
<send/>
<receive/>
</exchange>
<exchange action="respond" informationType="QuoteType" name="response">
<send/>
<receive/>
</exchange>
</interaction>
<workunit name="BarteringLoop" repeat="barteringDone = false">
<description type="documentation">Repeat until bartering has been completed</description>
<choice>
<silentAction roleType="BuyerRoleType"/>
<sequence>
<interaction channelVariable="Buyer2SellerC" name="AcceptQuote"
operation="quoteAccept">
<description type="documentation">Buyer accepts the quote and engages in the act of buying</descript
<participate fromRoleTypeRef="BuyerRoleType" relationshipType="BuyerSeller"
toRoleTypeRef="SellerRoleType"/>
<exchange action="request" informationType="QuoteAcceptType"
name="AcceptQuote">
<send/>
<receive/>
</exchange>
</interaction>
<interaction channelVariable="Buyer2SellerC" name="SendChannel"
operation="sendChannel">
<description type="documentation">Buyer send channel to seller to enable callback behavior</descript
<participate fromRoleTypeRef="BuyerRoleType" relationshipType="BuyerSeller"
toRoleTypeRef="SellerRoleType"/>
<exchange action="request" channelType="2BuyerChannelType" name="sendChannel">
<send variable="cdl:getVariable('DeliveryDetailsC','','')"/>
<receive variable="cdl:getVariable('DeliveryDetailsC','','')"/>
</exchange>
</interaction>
<assign roleType="BuyerRoleType">
<copy name="copy">
<source expression="true"/>
<target variable="cdl:getVariable('barteringDone','','')"/>
</copy>
</assign>
</sequence>
<sequence>
<interaction channelVariable="Buyer2SellerC" name="RequestNewPrice" operation="quoteUpdate">
<description type="documentation">Buyer updates the Quote - in effect requesting a new price</descri
<participate fromRoleTypeRef="BuyerRoleType" relationshipType="BuyerSeller"
```

```
toRoleTypeRef="SellerRoleType"/>
<exchange action="request" informationType="QuoteUpdateType" name="updateQuote">
<send/>
<receive/>
</exchange>
<exchange action="respond" informationType="QuoteAcceptType" name="acceptUpdatedQuote">
<send/>
<receive/>
</exchange>
</interaction>
</sequence>
</choice>
</workunit>
<interaction channelVariable="Seller2CreditChkC" name="CreditCheck" operation="creditCheck">
<description type="documentation">Seller check credit with CreditChecker</description>
<participate fromRoleTypeRef="SellerRoleType" relationshipType="SellerCreditCheck"
toRoleTypeRef="CreditCheckerRoleType"/>
<exchange action="request" informationType="CreditCheckType" name="checkCredit">
<send/>
<receive/>
</exchange>
</interaction>
<choice>
<interaction channelVariable="Seller2CreditChkC" name="CheckFails" operation="creditCheck">
<description type="documentation">Credit Checker fails credit check</description>
<participate fromRoleTypeRef="SellerRoleType" relationshipType="SellerCreditCheck"
toRoleTypeRef="CreditCheckerRoleType"/>
<exchange action="respond" faultName="creditCheckFails" informationType="CreditRejectType"
name="creditCheckFails">
<send/>
<receive/>
</exchange>
</interaction>
<sequence>
<interaction channelVariable="Seller2CreditChkC" name="CreditOk" operation="creditCheck">
<description type="documentation">Credit Checker passes credit</description>
<participate fromRoleTypeRef="SellerRoleType" relationshipType="SellerCreditCheck"
toRoleTypeRef="CreditCheckerRoleType"/>
<exchange action="respond" informationType="CreditAcceptType" name="creditCheckPasses">
<send/>
<receive/>
</exchange>
</interaction>
<interaction channelVariable="Seller2ShipperC" name="ReqDelivery" operation="requestShipping">
<description type="documentation">Seller requests delivery details - passing channel for buyer and s
<participate fromRoleTypeRef="SellerRoleType" relationshipType="SellerShipper"
toRoleTypeRef="ShipperRoleType"/>
<exchange action="request" informationType="RequestDeliveryType" name="sellerRequestsDelivery">
<send/>
<receive/>
</exchange>
<exchange action="respond" informationType="DeliveryDetailsType" name="sellerReturnsDelivery">
<send/>
<receive/>
```

```
</exchange>
</interaction>
<interaction channelVariable="Seller2ShipperC" name="SendChannel" operation="sendChannel">
<description type="documentation">Shipper forward channel to shipper</description>
<participate fromRoleTypeRef="SellerRoleType" relationshipType="SellerShipper"
toRoleTypeRef="ShipperRoleType"/>
<exchange action="request" channelType="2BuyerChannelType" name="forwardChannel">
<send variable="cdl:getVariable('DeliveryDetailsC','','')"/>
<receive variable="cdl:getVariable('DeliveryDetailsC','','')"/>
</exchange>
</interaction>
<interaction channelVariable="DeliveryDetailsC" name="DeliveryDetails" operation="deliveryDetails">
<description type="documentation">Shipper sends delivery details to buyer</description>
<participate fromRoleTypeRef="ShipperRoleType" relationshipType="ShipperBuyer"
toRoleTypeRef="BuyerRoleType"/>
<exchange action="request" informationType="DeliveryDetailsType" name="sendDeliveryDetails">
<send/>
<receive/>
</exchange>
</interaction>
</sequence>
</choice>
</sequence>
</choreography>
</package>
```

6 BPEL vs. WS-CDL

In diesem Kapitel soll ein Überblick über Einsatzgebiete und unterschiedliche Herangehensweisen der beiden Sprachen gegeben werden. Auch die verschiedenen Möglichkeiten der Sprachen sollen beleuchtet werden.

6.1 Point of View

Um überhaupt einen Vergleich anstellen zu können, muss zuerst geklärt werden, welche Grundvoraussetzungen für den Einsatz der Sprachen notwendig sind und welche Einblicke in das Gesamtsystem ein Anwender haben muss, um sie benutzen zu können. Beide sind dafür vorgesehen, mehrere Web Services in Verbindung zu bringen. Diese Aussage trifft auf beide zu, ist aber für eine Betrachtung der Unterschiede und Gemeinsamkeiten zu allgemein.

BPEL4WS betrachtet den Geschäftsprozess aus Sicht eines an der Interaktion beteiligten Partners (siehe Abbildung 26). Das heißt, dass dabei die internen Abläufe eines Web Services und die Schnittstellen dieses Web Service zu anderen, externen Web Services betrachtet werden. Eine Folge dieser Herangehensweise ist das Fehlen einer globalen Sicht über die Schnittstellen zwischen den Web Services. Nur Schnittstellen zu Web Services, die mit dem eigenen Web Service Informationen austauschen, können mit BPEL4WS modelliert werden. Durch diese Beschränkung auf die Betrachtung eines einzelnen Web Service und dessen Schnittstellen schafft sich BPEL4WS jedoch einen Vorteil. Der Code ist ausführbar.

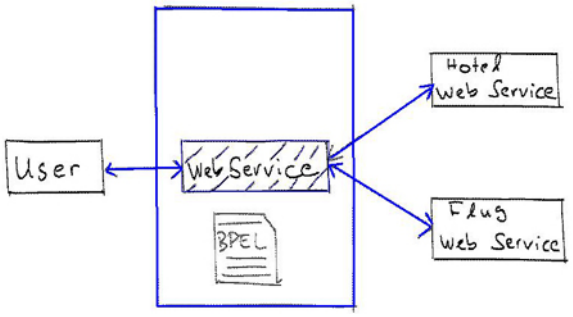

Abbildung 26: Point of View BPEL

WS-CDL dagegen arbeitet mit einer anderen Sichtweise (siehe Abbildung 27). Wichtig ist hier der globale Überblick über die zusammenarbeitenden Web Services. WS-CDL modelliert dabei jede einzelne Schnittstelle zwischen den einzelnen Web Services. Dabei ist es völlig egal, wie die einzelnen Web Services intern aufgebaut sind. Es wird gleich klar, dass damit das Entstehen eines ausführbaren Modells wie bei BPEL4WS nicht möglich ist, da interne Abläufe eines Partners überhaupt nicht berücksichtigt werden. Auch die Schnittstellen werden nicht bis ins kleinste Detail formalisiert und modelliert, sondern es werden nur grobe Schemata zur Kommunikation erstellt, beispielsweise Rollen, welche für eine Interaktion notwendig sind und welche Art Informationen ausgetauscht werden.

Diese Einordnung des jeweiligen Point of View ist für die Einordnung im Stack der vielen anderen WS-Standards wichtig.

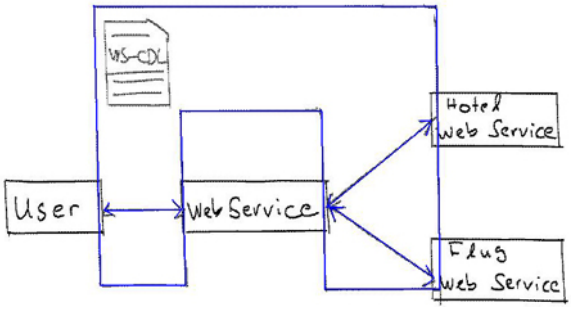

Abbildung 27: Point of View WS-CDL

6.2 Web Service Stack

Zur Nutzung von Web Services zur Modellierung von Geschäftsprozessen ist ein einzelner Standard nicht ausreichend. Vielmehr gibt es einen ganzen Stack mit aufeinander aufbauenden Standards, der eine sichere Kommunikation, Transaktionen, das Auffinden von Web Services und weitere grundlegende Operationen bietet, ohne die der Ablauf eines Prozesses nicht möglich wäre.

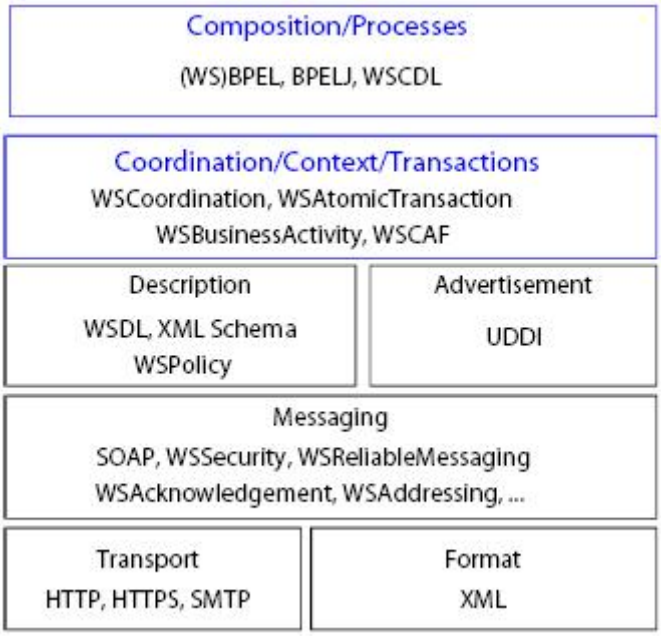

Abbildung 28: Web Service Stack - BPEL/WS-CDL auf einer Ebene [12]

Bisher wird das obere Ende dieses Stacks durch BPEL4WS repräsentiert, da hier die Komposition von mehreren Web Services zu einem Geschäftsprozess ermöglicht wird. Andere Standards werden benutzt, diese befinden sich auf derselben Ebene oder auf unterhalb liegenden Ebenen.

Will man nun für WS-CDL einen Platz in diesem Stack suchen, so muss man sich fragen, wo der richtige Platz für diese neue Definitionssprache ist. Dabei gibt es nur eine Möglichkeit: WS-CDL muss noch über BPEL4WS platziert werden, da mit WS-CDL keine einzelnen Web Services direkt umgesetzt werden können und somit auch die Komposition mehrerer Web Services nicht

möglich ist. Lediglich das Wiederverwenden von bereits vorhandenen Choreographien und das Zusammenstellen dieser zu einer neuen formalen Repräsentation ist möglich. Vielmehr ist WS-CDL mit seiner globalen Sichtweise und der grundlegenden Schnittstellenfestlegung für eine neue oberste Ebene prädestiniert.

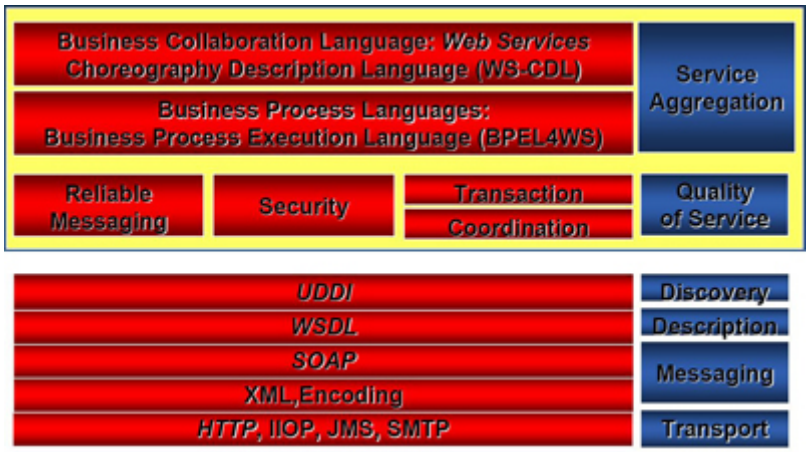

Abbildung 29: Web Service Stack (WS-CDL top level)

6.3 Einsatzgebiete und Anwendung

Wegen der unterschiedlichen Sichtweisen und der unterschiedlichen Ausrichtung sind die Einsatzgebiete und die Anwendung der beiden Sprachen deutlich verschieden.

BPEL4WS eignet sich zur Bottom-Up-Modellierung. Sind bereits Web-Services vorhanden und soll ein neuer Geschäftsprozess aus diesen bestehenden Services modelliert werden, dann ist BPEL4WS die erste Wahl. Jeder Partner kann für sich dann auch ein lauffähiges Modell erstellen und einsetzen. Auch die Ablauf-Modellierung eines einzelnen Web Service, der als WSDL -Beschreibung vorliegt und der keine Interaktionen mit anderen Partnern hat (siehe Szenario Kreditgenehmigung), lässt sich so problemlos umsetzen.

Ganz anders stellt sich das Einsatzgebiet von WS-CDL dar. Diese Sprache eignet sich vornehmlich zur Top-Down-Modellierung. Sind noch keine fertigen Web Services vorhanden, so können von vornherein die Schnittstellen formalisiert werden und Kommunikationsprobleme ausgeräumt werden. Auf Basis dieser formalen Beschreibungen in WS-CDL lassen sich dann in der Sprache der Wahl, z.B. BPEL4WS, die Web Services und die Partner-Interaktionen umsetzen. Schwieriger oder sogar gänzlich sinnlos wird die Modellierung eines Geschäftsprozesses in WS-CDL, wenn bereits vorhandene Web Services mit bestehenden Interaktionen modelliert werden sollen. Ganz unmöglich ist sogar die Modellierung eines einzelnen Web Service, da die Sprache nicht ablauffähig ist und auch keine Sprachkonstrukte für interne Modellierungen mitbringt.

6.4 Direkter Vergleich einiger Aspekte

Aspekt	BPEL4WS	WS-CDL
Modellierung von Kollaboration <1>	++	++
Modellierung von Ausführungkontrolle <2>	++	-
Repräsentation von Rollen <3>	+-	++
Transaktionen und Abgleich <4>	0	0
Exception Handling <5>	++	+
Sicherheit und reliable messaging <6>	-	-
Composability <7>	+	++
Event Handling <8>	++	-
Context Unterstützung <9>	++	+
Unterstützung durch Softwarehersteller <10>	++	-

Tabelle 1: Direkter Vergleich einiger Aspekte

Legende:

- ++ = (sehr) gute Unterstützung
- + = Unterstützung
- +- = schwache Unterstützung
- **0** = indirekte Unterstützung
- **-** = keine Unterstützung

Bemerkungen:
zu <1>:

- BPEL: Schlüsselkonzept „Partner Link“ modelliert peer-to-peer Kollaboration
- WS-CDL: Schlüsselkonzept „choreography“ modelliert peer-to-peer Kollaboration

zu <2>:

- BPEL: blockstrukturiert (jeder Block definiert eigenen Zusammenhang, Verhalten), transitionsstrukturiert (direkter Graph, Reihenfolge der Aktivitäten)
- WS-CDL: nicht vorhanden (nur Sequenz der Basis-Aktivitäten einer Kollaboration)

zu <3>:

- BPEL: nur „myRole“ und „partnerRole“-Konstrukte
- WS-CDL: „role“ und „participants“-Konstrukte modellieren ausreichende Semantik

zu <4>:

- BPEL: Transaktion realisiert durch „fault handler“ und „compensation“
- WS-CDL: Compensation realisiert durch „finalizer“ block

zu <5>:

- BPEL: „fault handler“ um Fehler aufzufangen, „terminate“ activity
- WS-CDL: „exception“ block um Fehler aufzufangen

zu <6>:

- BPEL: Nur Empfehlung WS-Security zu benutzen.
- WS-CDL: Nur Empfehlung WS-Security und WS-Reliability zu benutzen

zu <7>:

- BPEL: Activities können beliebig genested werden. Die Funktionen, die ein Geschäftsprozess anbietet können als Web Service angeboten werden.
- WS-CDL: Choreography unterstützt rekursive Composition (durch Kombination von bereits vorhandenen Choreographien)

zu <8>:

- BPEL: Prozesse und Aktivitäten können mit „event handlers“ assoziiert werden. „Event handler“ verarbeitet eingehende Nachrichten.
- WS-CDL: Nicht unterstützt.

zu <9>:

- BPEL: „scope“ definiert Kontext der Aktivitäten
- WS-CDL: „choreography“ ist der scope des Kontexts.

zu <10>:

- BPEL: Viele Hersteller haben Tools am Markt und bieten Unterstützung.
- WS-CDL: Nicht vorhanden, da keine kommerziellen Tools. (Bedingte „Herstellerunterstützung“ durch Open-Source-Community bei den Open-Source-Tools)

6.5 Fazit

Ein direkter Vergleich ist kaum möglich. Zu unterschiedlich sind die Einsatzgebiete, die Möglichkeiten und die Sichten (Point of View) der Sprachen. Zum heutigen Zeitpunkt ist BPEL4WS auf jeden Fall eine ausgereiftere Sprache mit breiter Unterstützung nahmhafter Unternehmen und umfassender, z.T. sehr guter, Dokumentation. WS-CDL hat dagegen einen schweren Stand. Das Einsatzgebiet ist beschränkt auf die Schnittstellenformalisierung bzw. der Vorgabe von Constraints aufgrund derer Geschäftspartner ihre Web Services entwickeln können. Eine automatische Generierung von „Skelett"-Prozesse aus WS-CDL Choreographien oder sogar weitergehend die Generierung von Web Service liegt noch in ferner Zukunft, sollte es überhaupt in Angriff genommen werden. Wie bereits im Kapitel 4 - WS-CDL thematisiert könnte aber in naher Zukunft eine weitere bzw. stärkere Integration zwischen WS-CDL und BPEL erfolgen und somit die Einsatzmöglichkeiten und der Umfang des Einsatzes erweitert werden. Hier kann man nur abwarten, welche Akzeptanz die Zukunft bringt.

7 Anhang

7.1 Erstes Herantasten

7.1.1 Reisebuchung

Versuche der Formalisierung

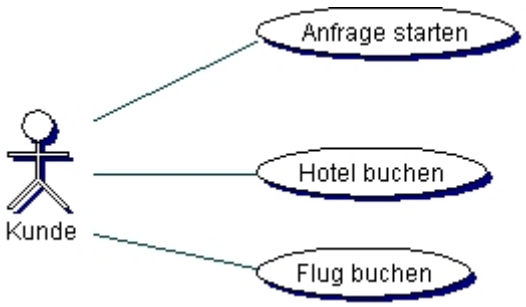

Abbildung 30: Use Case Kunde

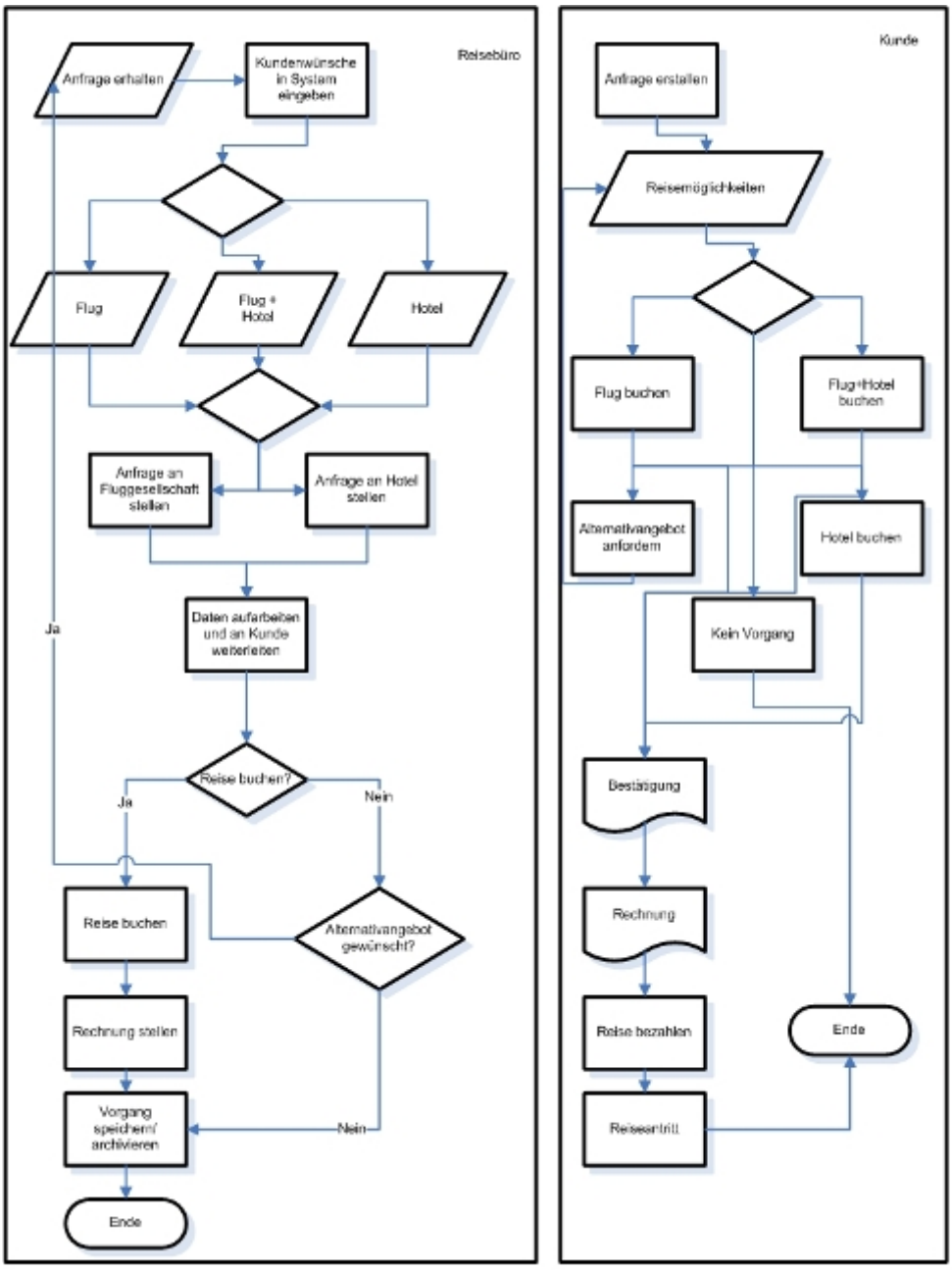

Abbildung 31: Real World Abläufe Reisebüro und Kunde

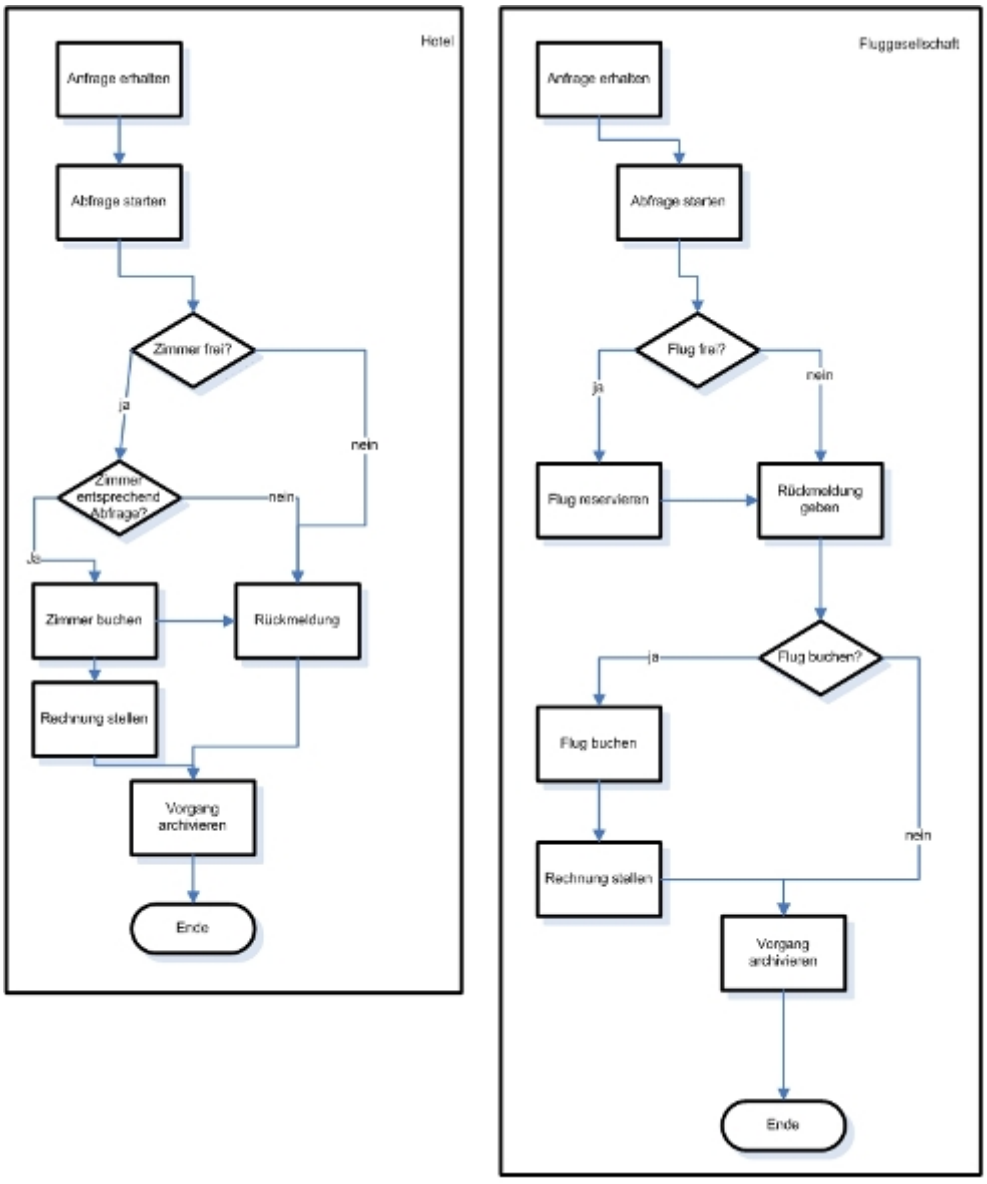

Abbildung 32: Real World Abläufe Hotel und Fluggesellschaft

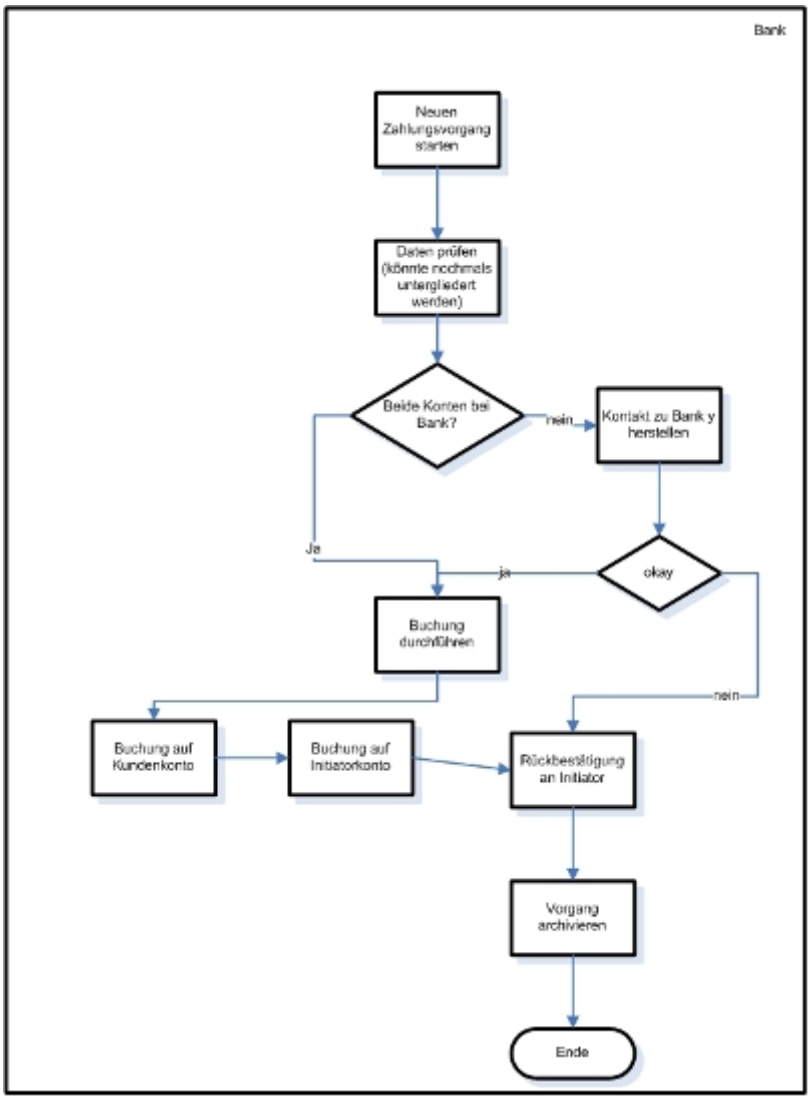

Abbildung 33: Real World Abläufe Bank

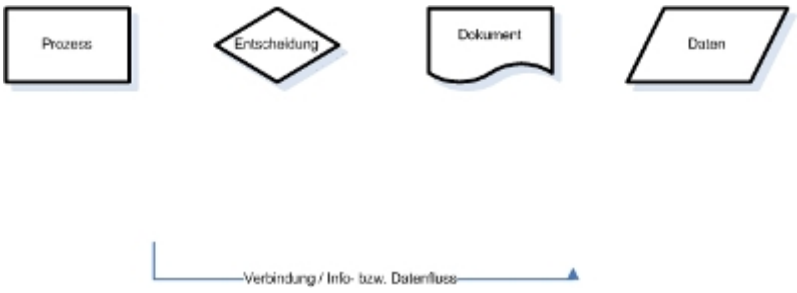

Abbildung 34: Legende zu den Real World Abläufe

WS-CDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Reisebuchung choreography, created 27/8/2005 by CS -->
<package
name="Travel-Service"
author="CS"
version="0.2"
targetNamespace="http://fachstudie.ingoal.info"
xmlns="http://www.w3.org/2004/12/ws-chor/cdl"
xmlns:tns="http://fachstudie.ingoal.info"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >

<!-- Information Types -->
<informationType name="travelData" type="xsd:string" exceptionType="false" />
<informationType name="bankBookingNumber" type="xsd:int" exceptionType="false" />
<informationType name="flightBookingNumber" type="xsd:string" exceptionType="false" />
<informationType name="hotelBookingNumber" type="xsd:int" exceptionType="false" />

<!-- Tokens -->
    <token name="flightRef" informationType="tns:uriType" />
    <token name="hotelRef" informationType="tns:uriType" />
    <token name="offerConfirmation" informationType="xsd:string" />
    <token name="sendBillt" informationType="xsd:string" />
    <token name="payBillt" informationType="xsd:string" />
<token name="amount" informationType="xsd:float" />

<!-- Token Locators -->
<tokenLocator tokenName="amount" informationType="String" part="price" query="/bill/price" />

<!-- Role Types -->
    <!-- Muss Interaktion mit Computern auch als Rolle implementiert werden? -->
    <!-- Kundensicht -->
    <roleType name="Client" >
        <behavior name="client" interface="tns:clientPT" />
    </roleType>

    <!-- Hotel -->
    <roleType name="ServiceEmpHotel">
        <behavior name="serviceEmpHotel" />
    </roleType>
    <roleType name="AccountingHotel">
        <behavior name="accounting" />
    </roleType>

    <!-- Bank -->
    <roleType name="ServiceEmpBank">
     <behavior name="serviceEmpBank" />
    </roleType>

    <roleType name="InitiatorClient">
     <behavior name="initiatorClient" />
    </roleType>
```

```
    <roleType name="RecipientClient">
     <behavior name="recipientClient" />
    </roleType>

    <roleType name="Bank_Computer">
     <behavior name="bank_Computer" />
    </roleType>

    <!-- Flightgesellschaft -->
    <roleType name="ServiceEmpFlight">
     <behavior name="serviceEmpFlight" />
    </roleType>

    <!-- Travelbuero -->
    <roleType name="ServiceEmpTravel">
     <behavior name="serviceEmpTravel" />
    </roleType>

<!-- Relationship Types -->
    <relationshipType name="ServiceEmpTravel-ServiceEmpHotel">
        <role type="tns:ServiceEmpTravel" />
        <role type="tns:ServiceEmpHotel" />
    </relationshipType>

    <relationshipType name="ServiceEmpTravel-ServiceEmpFlight">
        <role type="tns:ServiceEmpTravel" />
        <role type="tns:ServiceEmpFlight" />
    </relationshipType>

    <relationshipType name="Client-ServiceEmpTravel">
        <role type="tns:Client" />
        <role type="tns:ServiceEmpTravel" />
    </relationshipType>

    <relationshipType name="Client-ServiceEmpBank">
        <role type="tns:Client" />
        <role type="tns:ServiceEmpBank" />
    </relationshipType>

    <relationshipType name="Entry_Initiator">
        <role type="tns:InitiatorClient" />
        <role type="tns:Bank_Computer" />
    </relationshipType>

    <relationshipType name="Entry_Recipient">
        <role type="tns:RecipientClient" />
        <role type="tns:Bank_Computer" />
    </relationshipType>

<!-- Participant Types -->
<participantType name="Bank_Business">
<role type="InitiatorClient" />
<role type="RecipientClient" />
```

```
<role type="Bank_Computer" />
</participantType>


<!-- Channel Types -->

   <!-- Zusammenarbeit / Collaboration zwischen Geschaeftspartnern, wie wird
    Information ausgetauscht -->
    <channelType name="TravelChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:ServiceEmpTravel" />
        <reference>
            <token name="tns:amount" />
        </reference>
    </channelType>

    <channelType name="OfferConfirmationChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:Client" />
        <reference>
            <token name="tns:offerConfirmationt" />
        </reference>
    </channelType>

    <channelType name="SendBillChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:Client" />
        <reference>
            <token name="tns:sendBillt" />
        </reference>
    </channelType>

    <channelType name="PayBillChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:ServiceEmpTravel" />
        <reference>
            <token name="tns:payBillt" />
        </reference>
    </channelType>

<channelType name="FlightChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:ServiceEmpFlight" />
        <reference>
            <token name="tns:flightRef" />
        </reference>
    </channelType>

    <channelType name="BankChannelInternal"
```

```
                usage="unlimited"
                action="request-respond">
        <role type="tns:Bank_Computer" />
        <!-- gibt es Kanaele in 2 Richtungen wo man 2 Rollen angeben kann? -->
        <reference>
            <token name="tns:bankRef"/>
        </reference>
    </channelType>

    <channelType name="BankChannelExternal"
                usage="unlimited"
                action="request-respond">
        <role type="tns:Bank_Computer" />
        <reference>
            <token name="tns:bankRefExt" />
        </reference>
    </channelType>

    <channelType name="HotelChannel"
                usage="unlimited"
                action="request-respond">
        <role type="tns:ServiceEmpHotel" />
        <reference>
            <token name="tns:hotelRef" />
        </reference>
    </channelType>

<!-- Choreographies -->
<choreography name="TravelBooking" root="true" coordination="true" >
<relationship type="Client-ServiceEmpTravel" />

<variableDefinitions>
<!-- information exchange capturing variables = iec
        state capturing variables = sc
        channel capturing variables = cc -->

<!-- allgemeine Variablen -->
<variable name="amount" informationType="xsd:float"/> <!-- iec -->
<variable name="bookFlightHotel" free="false"/>

<!-- Reisebuero-Sachen-->
<variable name="amount" informationType="xsd:float" free="true"
            mutable="true" roleTypes="peopleHandlingCash"/> <!-- iec -->
<variable name="travelOffer" informationType="travelData"/>
<variable name="bookFlightHotel" informationType="xsd:string"
             free="false" roleTypes="ServiceEmpTravel"></variable>
<variable name="offerConfirmation" roleTypes="ServiceEmpTravel"/>
<variable name="roomReq"/>
<variable name="flightReq"/>

<!-- Bank-Sachen -->
<variable name="sameInstitute" roleTypes="tns:Bank_Computer"/> <!-- sc -->
<variable name="contactOtherInst"/> <!-- sc -->
<variable name="amount" informationType="xsd:float" free="false"
```

```
            roleTypes="peopleHandlingCash"> </variable> <!-- iec -->
<variable name="BookingAck" roleTypes="tns:Bank_Computer"/>
<variable name="ClientAccount" informationType="tns:uriType"
roleTypes="peopleHandlingCash"/>
<!--xi:include href="someVariableDefinitions" /-->

 <!-- Kunde -->
<variable name="bookFlightHotel" mutable="false" free="true"/>
<variable name="travelRequest" informationType="travelData"/>
<variable name="travelRequestAlt" informationType="travelData"/>
<variable name="offerConfirmation" roleTypes="Client"/>
<variable name="bill"/>


<!-- Hotel-Sachen -->
<variable name="reservationStatus"/>
<variable name="bookRoom"/>
<variable name="chargeToAccount"/>
<variable name="roomConfirmation"></variable>

<!-- Flug-Sachen -->
<variable name="flightStatus"/>
<variable name="flightConfirmation"/>
<variable name="bookFlight"/>
<variable name="chargeToAccount"/>

<variable name="travel-channel" channelType="TravelChannel" />
<variable name="flight-channel" channelType="FlightChannel" />
    <variable name="bank-channel-internal" channelType="BankChannelInternal" />
    <variable name="bank-channel-external" channelType="BankChannelExternal" />
    <variable name="hotel-channel" channelType="hotelChannel" />
    <variable name="offerConfirmation-channel" channelType="OfferConfirmationChannel" />
    <variable name="sendBill-channel" channelType="SendBillChannel" />
    <variable name="payBill-channel" channelType="PayBillChannel" />

</variableDefinitions>

<!-- Remove an substitute for Choreography or Activity Notation -->
     <!-- <noAction roleType="nothing" /> -->

<!-- travel agency choreography -->
<choreography name="TravelBookingTA" root="false">
    <sequence>
    <interaction name="getTravelOffer"
                channelVariable="travel-Channel"
                operation="handleOfferRequest"
                align="true"
                initiate="true">
        <participate relationshipType="Client-ServiceEmpTravel"
                    fromRole="Client" toRole="ServiceEmpTravel" />
        <!-- Kann ein Informationsaustausch nicht beidseitig funktionieren?
        Muss man Infos bzw. den Austausch dann fuer jede Richtung extra
        implementieren (Problem mit action=request|respond -->
        <exchange name="travelData_req"
```

```
            informationType="travelData"
            action="request">
        <send variable="cdl:getVariable("tns:travelRequest", ","", ","","")"/>
        <receive variable="cdl:getVariable("tns:travelRequest", ","", ","","")"/>
    </exchange>
    <!-- Frage: Sind Anfrage und Antwort die gleichen Daten? -->
    <exchange name="travelData_res"
            informationType="travelData"
            action="response">
        <send variable="cdl:getVariable("tns:travelOffer", ","", ","","")"/>
        <receive variable="cdl:getVariable("tns:travelOffer", ","", ","","")"/>
    </exchange>
    <exchange name="travelDataAlternative"
            informationType="travelData"
            action="request">
        <send variable="cdl:getVariable("tns:travelRequestAlt", ","", ","","")"/>
        <receive variable="cdl:getVariable("tns:travelRequestAlt", ","", ","","")"/>
    </exchange>
</interaction>

<interaction name="confirmTravelOffer"
            channelVariable="offerConfirmation-Channel"
            operation="handleOfferConfirmation"
            align="true"
            initiate="false">
    <participate relationshipType="Clien-ServiceEmpTravel"
                fromRole="Client" toRole="ServiceEmpTravel" />
    <exchange name="offerConf"
            action="request">
        <send variable="cdl:getVariable("tns:offerConfirmation", ","", ","","")"/>
        <receive variable="cdl:getVariable("tns:offerConfirmation", ","", ","","")"/>
    </exchange>
</interaction>

<interaction name="sendBill"
            channelVariable="sendBill-Channel"
            operation="handleBilling"
            align="true"
            initiate="false">
    <participate relationshipType="Client-ServiceEmpTravel"
                fromRole="Client" toRole="ServiceEmpTravel" />
    <exchange name="writeSendBill"
            channelType="TravelChannel"
            action="respond">
            <!-- Ist der channelType notwendig/sinnvoll/moeglich? -->
        <send variable="cdl:getVariable("tns:bill", ","", ","","")"/>
        <receive variable="cdl:getVariable("tns:bill", ","", ","","")"/>
    </exchange>
</interaction>

<interaction name="payBill"
            channelVariable="payBill-Channel"
            operation="handlePaying"
            align="true"
```

```
                initiate="false">
        <participate relationshipType="Client-ServiceEmpTravel"
                    fromRole="Client" toRole="ServiceEmpTravel" />
        <exchange name="payBill"
                  action="request">
                <send variable="cdl:getVariable("tns:bill", ","", ","","")"/>
                <receive variable="cdl:getVariable("tns:bill", ","", ","","")"/>
            </exchange>
        </interaction>
        </sequence>
    </choreography>

<!-- hotel choreography -->
<choreography name="RequestTA_Hotel" root="false">
    <sequence>
    <interaction name="requestHotel"
                channelVariable="hotel-channel"
                operation="handleHotelRoom"
                align="true"
                initiate="false">
        <participate relationshipType="ServiceEmpTravel-ServiceEmpHotel"
                    fromRole="ServiceEmpTravel" toRole="ServiceEmpHotel" />
        <exchange name="room_req"
                  action="request">
            <send variable="cdl:getVariable("tns:roomReq", ","", ","","")"/>
            <receive variable="cdl:getVariable("tns:roomReq", ","", ","","")"/>
        </exchange>
    </interaction>

    <interaction name="confirmationHotel"
                channelVariable="hotel-channel"
                operation="confirmHotelRoom"
                align="true"
                initiate="false">
        <participate relationshipType="ServiceEmpTravel-ServiceEmpHotel"
                    fromRole="ServiceEmpTravel" toRole="ServiceEmpHotel" />
        <exchange name="room_res"
           informationType="hotelBookingNumber"
                  action="respond">
            <send variable="cdl:getVariable("tns:roomConfirmation", ","", ","","")"/>
            <receive variable="cdl:getVariable("tns:roomConfirmation", ","", ","","")"/>
        </exchange>
    </interaction>

    <interaction name="billHotel"
                channelVariable="hotel-channel"
                 operation="billHotelRoom"
                 align="true"
                 initiate="false">
         <participate relationshipType="ServiceEmpTravel-ServiceEmpHotel"
                     fromRole="ServiceEmpTravel" toRole="ServiceEmpHotel" />
         <exchange name="billRoom_res"
                    action="respond">
             <send variable="cdl:getVariable("tns:chargeToAccount", ","", ","","")"/>
```

```
            <receive variable="cdl:getVariable("tns:chargeToAccount", ","", ","","")"/>
         </exchange>
     </interaction>
    </sequence>
</choreography> <!-- das gleiche fuer Flug, mit Reservierung oder ohne? -->

<!-- flight choreography -->
<choreography name="RequestTA_Flight" root="false">
    <sequence>
    <interaction name="requestFlight"
                channelVariable="flight-channel"
                operation="handleFlight"
                align="true"
                initiate="false">
        <participate relationshipType="ServiceEmpTravel-ServiceEmpFlight"
                    fromRole="ServiceEmpTravel" toRole="ServiceEmpFlight" />
        <exchange name="flight_req"
                  action="request">
            <send variable="cdl:getVariable("tns:flightReq", ","", ","","")"/>
            <receive variable="cdl:getVariable("tns:flightReq", ","", ","","")"/>
        </exchange>
    </interaction>

    <interaction name="confirmationFlight"
                channelVariable="flight-channel"
                operation="confirmFlight"
                align="true"
                initiate="false">
        <participate relationshipType="ServiceEmpTravel-ServiceEmpFlight"
                    fromRole="ServiceEmpTravel" toRole="ServiceEmpFlight" />
        <exchange name="flight_res"
                  informationType="flightBookingNumber"
                  action="respond">
            <send variable="cdl:getVariable("tns:flightConfirmation", ","", ","","")"/>
            <receive variable="cdl:getVariable("tns:flightConfirmation", ","", ","","")"/>
        </exchange>
    </interaction>

    <interaction name="billFlight"
                channelVariable="flight-channel"
                operation="billFlightTicket"
                align="true"
                initiate="false">

        <participate relationshipType="ServiceEmpTravel-ServiceEmpFlight"
                    fromRole="ServiceEmpTravel" toRole="ServiceEmpFlight" />
        <exchange name="billFlight_res"
                  action="respond">
            <send variable="cdl:getVariable("tns:chargeToAccount", ","", ","","")"/>
            <receive variable="cdl:getVariable("tns:chargeToAccount", ","", ","","")"/>
        </exchange>
    </interaction>
    </sequence>
</choreography>
```

```
<!-- bank choreography -->
<choreography name="Bank_internal" root="false">
    <interaction name="requestBank"
                 channelVariable="bank-channel-external"
                 operation="handleBooking"
                 align="true"
                 initiate="false">

        <participate relationshipType="Client-ServiceEmpBank"
                     fromRole="Client" toRole="ServiceEmpBank" />
        <exchange name="payingBill"
                  action="request">
            <send variable="cdl:getVariable("tns:bill", "","", "","","")"/>
            <receive variable="cdl:getVariable("tns:bill", "","", "","","")"/>
        </exchange>
    </interaction>
</choreography>


<!-- Uncomment and augment to add Exception Block -->

<!-- <exceptionBlock  name="name"> -->
<!-- WorkUnit-Notation -->
<!-- </exceptionBlock> -->
<!-- Uncomment and augment to add a Finalizer Block -->
<!-- <finalizerBlock  name="name"> -->
<!-- WorkUnit-Notation -->
<!-- </finalizerBlock> -->
</choreography>

</package>
```

7.1.2 Kreditgenehmigung

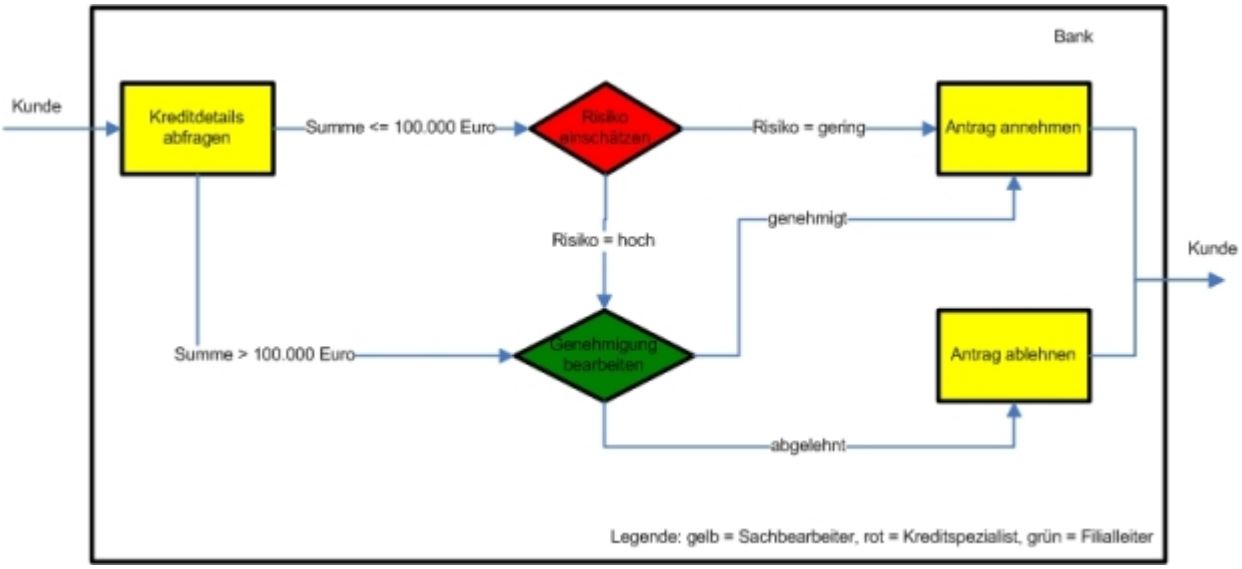

Abbildung 35: Grober Ablauf Szenario Kreditgenehmigung

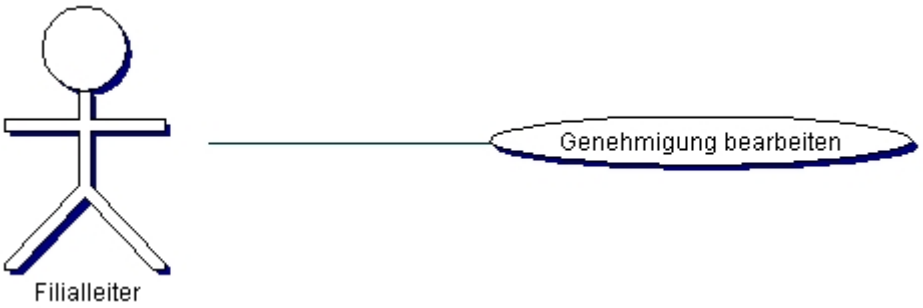

Abbildung 36: Use Case Filialleiter

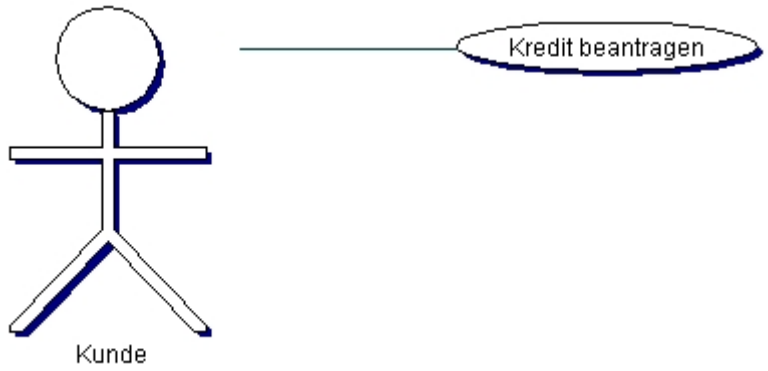

Abbildung 37: Use Case Kunde

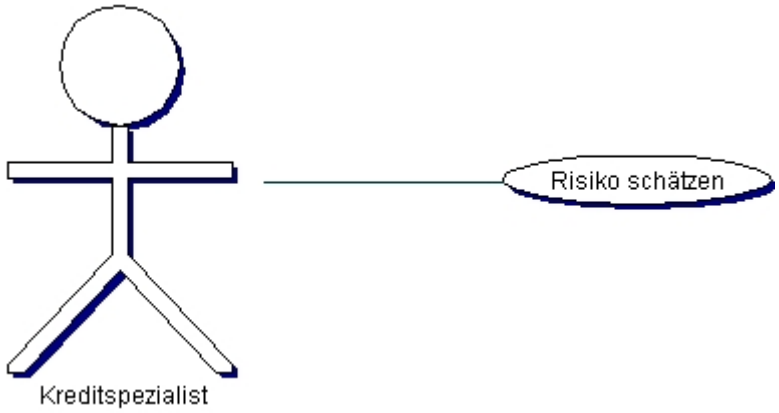

Abbildung 38: Use Case Kreditspezialist

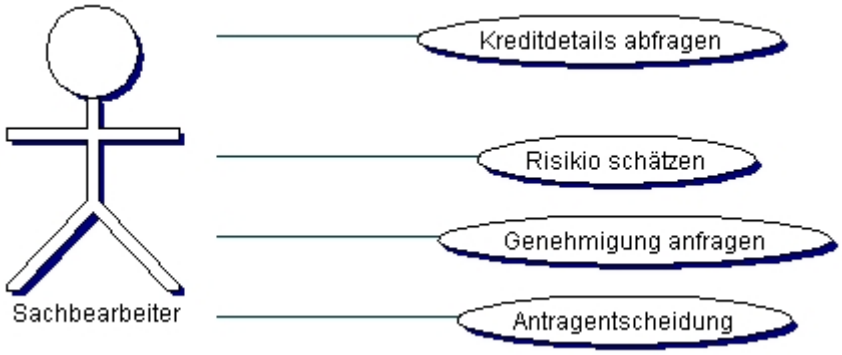

Abbildung 39: Use Case Sachbearbeiter

7.2 BPEL Prozesse

7.2.1 Kreditgenehmigung

loanProcess.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<process expressionLanguage="Java" name="LoanProcess" suppressJoinFailure="yes"
targetNamespace="http://www.example.com/ process42500892/"
wpc:autoDelete="no" wpc:autonomy="peer" wpc:businessRelevant="yes"
wpc:compensationSphere="notSupported" wpc:displayName="LoanProcess"
wpc:executionMode="longRunning" wpc:validFrom="2003-01-01 T00:00:00"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:wpc="http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/"
xmlns:wsdl="http://loans.org/wsdl/loan-approval"
xmlns:wsdl0="http://www.example.com/process42500892/interface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/business-process/
http://schemas.xmlsoap.org/ws/2003/03/business-process/">
<partnerLinks>
<partnerLink myRole="loanService" name="customer" partnerLinkType="wsdl:loanPartnerLinkType"/>
<partnerLink myRole="approverCallback" name="approver"
partnerLinkType="wsdl:loanApprovalLinkType" partnerRole="approver"/>
<partnerLink name="assessor" partnerLinkType="wsdl:riskAssessmentLinkType" partnerRole="assessor"/>
</partnerLinks>
<variables>
<variable messageType="wsdl:approvalMessage" name="approval"/>
<variable messageType="wsdl:riskAssessmentMessage" name="risk"/>
<variable messageType="wsdl:errorMessage" name="error"/>
<variable messageType="wsdl:creditInformationMessage" name=" request"/>
</variables>
<correlationSets>
<correlationSet name="id" properties="wsdl0:id"/>
</correlationSets>
<faultHandlers>
<catch faultName="wsdl:loanProcessFault" faultVariable="error">
<reply faultName="wsdl:unableToHandleRequest" name="ReplyFault"
operation="request" partnerLink="customer" portType="wsdl:loanServicePT"
variable="error" wpc:displayName="ReplyFault" wpc:id="12"/>
</catch>
</faultHandlers>
<sequence wpc:id="1073741826">
<scope name="Scope" wpc:businessRelevant="no" wpc:id="18">
<flow name="Flow" wpc:displayName="Flow" wpc:id="4">
<links>
<link name="Link1"/>
<link name="Link3"/>
<link name="Link6"/>
<link name="Link2"/>
<link name="Link4"/>
<link name="Link5"/>
<link name="Link7"/>
```

```
</links>
<receive createInstance="yes" name="Receive" operation="request"
partnerLink="customer" portType="wsdl:loanServicePT" variable="request"
wpc:displayName="Receive" wpc:id="6">
<source linkName="Link2"/>
<source linkName="Link1" transitionCondition="DefinedByJavaCode">
<wpc:transitionCondition>
<wpc:javaCode>
<![CDATA[// @generated
//
// request.amount < 10000
return getRequest().getAmount().intValue() < 10000; ]]>
</wpc:javaCode>
</wpc:transitionCondition>
</source>
</receive>
<reply name="Reply" operation="request" partnerLink="customer" portType="
wsdl:loanServicePT" variable="approval" wpc:displayName="Reply" wpc:id="7">
<target linkName="Link7"/>
<target linkName="Link6"/>
</reply>
<scope name="Scope1" wpc:businessRelevant="no" wpc:id="19">
<target linkName="Link1"/>
<source linkName="Link4"/>
<source linkName="Link3" transitionCondition="DefinedByJavaCode">
<wpc:transitionCondition>
<wpc:javaCode>
<![CDATA[// @generated
//
// risk.level =="low"
return getRisk().getLevel() =="low"; ]]>
</wpc:javaCode>
</wpc:transitionCondition>
</source>
<invoke inputVariable="request" name="Assessor1" operation="check"
outputVariable="risk" partnerLink="assessor" portType="wsdl:riskAssessmentPT"
wpc:displayName="Assessor" wpc:id="8"/>
</scope>
<assign name="Assign" wpc:displayName="Assign" wpc:id="10">
<target linkName="Link3"/>
<source linkName="Link6" transitionCondition="DefinedByJavaCode">
<wpc:transitionCondition>
<wpc:true/>
</wpc:transitionCondition>
</source>
<copy>
<from>
<wpc:literal type="xsd:string"><![CDATA[yes ]]></wpc:literal>
</from>
<to part="accept" variable="approval"/>
</copy>
</assign>
<invoke inputVariable="request" name="Approver1" operation="approve"
partnerLink="approver" portType="wsdl:loanApprovalPT" wpc:displayName="Approver" wpc:id="13">
```

```
<target linkName="Link4"/>
<target linkName="Link2"/>
<source linkName="Link5"/>
<correlations>
<correlation initiate="yes" set="id"/>
</correlations>
</invoke>
<pick name="Pick" wpc:displayName="Pick" wpc:id="14">
<target linkName="Link5"/>
<source linkName="Link7"/>
<onMessage operation="approveCallback" partnerLink="approver"
portType="wsdl:loanApprovalCallbackPT" variable="approval">
<correlations>
<correlation initiate="no" set="id"/>
</correlations>
<empty name="Empty" wpc:displayName="Empty" wpc:id="15"/>
</onMessage>
<onAlarm for="DefinedByJavaCode">
<wpc:for>
<wpc:javaCode><![CDATA[ return 1800; //Zeitdauer in Sekunden]]></wpc:javaCode>
</wpc:for>
<assign name="Assign1" wpc:displayName="Assign1" wpc:id="17">
<copy>
<from>
<wpc:literal type="xsd:string"><![CDATA[timeout]]></wpc:literal>
</from>
<to part="accept" variable="approval"/>
</copy>
</assign>
</onAlarm>
</pick>
</flow>
</scope>
</sequence>
</process>
```

7.2.2 Reise buchen

7.2.3 bookingIntegration.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<process expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
name="BookingIntegration" suppressJoinFailure="yes"
targetNamespace="http://www.example.com/process82121773/"
wpc:autoDelete="no" wpc:businessRelevant="yes" wpc:compensationSphere="required"
wpc:displayName="BookingIntegration" wpc:executionMode="microflow"
wpc:validFrom="2003-01-01 T00:00:00"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:wpc="http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/"
xmlns:wsdl="http://www.example.com/process82121773/" xmlns:wsdl0="http://hotel.booking"
mlns:wsdl1="http://flight.booking"
xmlns:wsdl2="http://www.example.com/process82121773/interface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/ws/2003/03/business-process/
http://schemas.xmlsoap.org/ws/2003/03/business-process/">
<wpc:javaGlobals>
<wpc:import packages="java.util.Date"/>
<wpc:import packages="booking.flight.BookFlightElement"/>
<wpc:import packages="booking.hotel.BookRoomElement"/>
</wpc:javaGlobals>
<partnerLinks>
<partnerLink name="HotelBooking" partnerLinkType="wsdl:HotelBookingLT"
partnerRole="HotelBookingRole"/>
<partnerLink name="FlightBooking" partnerLinkType="wsdl:FlightBookingLT"
partnerRole="FlightBookingRole"/>
<partnerLink myRole="ProcessRole" name="Client" partnerLinkType="wsdl2:PartnerLinkType"/>
</partnerLinks>
<variables>
<variable messageType="wsdl2:InputMessage" name="input"/>
<variable messageType="wsdl2:OutputMessage" name="output"/>
<variable messageType="wsdl0:bookRoomRequest" name="hotelIn"/>
<variable messageType="wsdl0:bookRoomResponse" name="hotelOut"/>
<variable messageType="wsdl1:bookFlightRequest" name="flightIn"/>
<variable messageType="wsdl1:bookFlightResponse" name="flightOut"/>
<variable messageType="wsdl2:counterMessage" name="counter"/>
<variable messageType="wsdl2:priceMessage" name="totalPrice"/>
<variable messageType="wsdl0:cancelOrderRequest" name="cancelOrder"/>
<variable messageType="wsdl1:cancelFlightRequest" name="cancelFlight"/>
<variable messageType="wsdl2:FaultMessage" name="processError"/>
<variable messageType="wsdl1:flightFaultMessage" name="flightError"/>
<variable messageType="wsdl1:cancelFlightResponse" name="cancelFlightOut"/>
<variable messageType="wsdl0:cancelOrderResponse" name="cancelOrderOut"/>
</variables>
<faultHandlers>
<catch faultName="wsdl1:noFlightFault" faultVariable="flightError">
<sequence wpc:id="1073741868">
<assign name="Assign2" wpc:displayName="Assign" wpc:id="43">
<copy>
<from>
<wpc:literal type="xsd:string"><![CDATA[" Es sind keine Fluege verfuegbar"]]></wpc:literal>
</from>
<to part="errorCode" variable="processError"/>
</copy>
</assign>
<reply faultName="wsdl2:fault" name="Reply2" operation="ProcessOperation"
partnerLink="Client" portType="wsdl2:ProcessPortType" variable="processError"
wpc:displayName="Reply" wpc:id="45"/>
</sequence>
</catch>
<catchAll>
<sequence wpc:id="1073741839">
<assign name="Assign" wpc:displayName="Assign" wpc:id="14">
<copy>
<from>
<wpc:literal type="xsd:string"><![CDATA[Es ist ein Fehler aufgetreten !]]></wpc:literal>
</from>
```

```
<to part="errorCode" variable="processError"/>
</copy>
</assign>
<reply faultName="wsdl2:fault" name="Reply1" operation="ProcessOperation"
partnerLink="Client" portType="wsdl2:ProcessPortType" variable="processError"
wpc:displayName="Reply" wpc:id="16"/>
</sequence>
</catchAll>
</faultHandlers>
<sequence name="Sequence" wpc:displayName="Sequence" wpc:id="1073741825">
<receive createInstance="yes" name="Receive" operation="ProcessOperation"
partnerLink="Client" portTyp="wsdl2:ProcessPortType" variable="input"
wpc:displayName="Receive" wpc:id="42"/>
<invoke name="AssignInput" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="AssignInput" wpc:id="20">
<wpc:script>
<wpc:javaCode><![CDATA[// get input
String id = getInput().getID();
Date arrival = getInput().getArrival().getTime();
Date departure = getInput().getDeparture().getTime();
// set hotelIn
BookRoomElement bookRoom = new BookRoomElement();
bookRoom.setId(id);
bookRoom.setArrival(arrival);
bookRoom.setDeparture(departure);
getHotelIn().setParameters(bookRoom);
// set flightIn
BookFlightElement bookFlight = new BookFlightElement();
bookFlight.setId(id);
bookFlight.setArrival(arrival);
bookFlight.setDeparture(departure);
getFlightIn().setParameters(bookFlight);
// set counter
getCounter().setCounter(1);
// set cancelOrderOut
getCancelOrderOut().getParameters().setCancelOrderReturn("nix");]]></wpc:javaCode>
</wpc:script>
</invoke>
<scope name="Scope" wpc:businessRelevant="no" wpc:id="32">
<invoke inputVariable="hotelIn" name="Invoke" operation="bookRoom" outputVariable="hotelOut"
partnerLink="HotelBooking" portType="wsdl0:HotelBooking" wpc:continueOnError="no"
wpc:displayName="Invoke" wpc:id="4">
<wpc:undo inputVariable="cancelOrder" operation="cancelOrder" partnerLink="HotelBooking"
portType="wsdl0:HotelBooking"/>
</invoke>
</scope>
<scope name="Scope1" wpc:businessRelevant="no" wpc:id="48">
<invoke inputVariable="flightIn" name="Invoke1" operation="bookFlight"
outputVariable="flightOut" partnerLink="FlightBooking" portType="wsdl1:FlightBooking"
wpc:continueOnError="no" wpc:displayName="Invoke" wpc:id="5">
<wpc:undo inputVariable="cancelFlight" operation="cancelFlight"
partnerLink="FlightBooking" portType="wsdl1:FlightBooking"/>
</invoke>
</scope>
```

```
<while condition="DefinedByJavaCode" name="While" wpc:displayName="While" wpc:id="21">
<wpc:condition>
<wpc:javaCode><![CDATA[// @generated
//
// counter.counter <= 2
return getCounter().getCounter() <= 2;]]></wpc:javaCode>
</wpc:condition>
<switch name="Switch" wpc:displayName="Switch" wpc:id="22">
<case condition="DefinedByJavaCode" wpc:id="24">
<wpc:condition>
<wpc:javaCode><![CDATA[// @generated
//
// counter.counter == 1
return getCounter().getCounter() == 1;]]></wpc:javaCode>
</wpc:condition>
<sequence wpc:id="1073741847">
<invoke name="JavaSnippet" operation="null" partnerLink="null" portType="wpc:null"
wpc:displayName="JavaSnippet" wpc:id="26">
<wpc:script>
<wpc:javaCode><![CDATA[getTotalPrice().setTotalPrice(getHotelOut().
getParameters().getBookRoomReturn());
int i = getCounter().getCounter();
i++;
getCounter().setCounter(i);]]></wpc:javaCode>
</wpc:script>
</invoke>
</sequence>
</case>
<otherwise>
<invoke name="JavaSnippet1" operation="null" partnerLink="null"
portType="wpc:null" wpc:displayName="JavaSnippet" wpc:id="27">
<wpc:script>
<wpc:javaCode><![CDATA[getTotalPrice().setTotalPrice(getTotalPrice().
getTotalPrice() + getFlightOut().getParameters().getBookFlightReturn());
int i = getCounter().getCounter();
i++;
getCounter().setCounter(i);]]></wpc:javaCode>
</wpc:script>
</invoke>
</otherwise>
</switch>
</while>
<invoke name="AssignOutput" operation="null" partnerLink="null"
portType="wpc:null" wpc:displayName="AssignOutput" wpc:id="18">
<wpc:script>
<wpc:javaCode><![CDATA[float price = getTotalPrice().getTotalPrice();
String confirmation ="";
if (price>= 0) confirmation ="Der Gesamtpreis betrÃďgt: " + price + "EUR";
else confirmation ="Die Buchung ist nicht mÃűglich.";
getOutput().setConfirmation(confirmation);]]></wpc:javaCode>
</wpc:script>
</invoke>
<reply name="Reply" operation="ProcessOperation" partnerLink="Client"
portType="wsdl2:ProcessPortType" variable="output" wpc:displayName="Reply" wpc:id="3"/>
```

```
</sequence>
</process>
```

7.3 WSDL Dokumente

7.3.1 Kreditgenehmigung

7.3.2 loan-approval.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:tns="http://loans.org/wsdl/loan-approval"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:lns="http://loans.org/wsdl/loan-approval" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://loans.org/wsdl/loan-approval">
<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>
<message name="approvalMessage">
<part name="accept" type="xsd:string"/>
<part name="id" type="xsd:string"/>
</message>
<message name="creditInformationMessage">
<part name="firstName" type="xsd:string"/>
<part name="name" type="xsd:string"/>
<part name="amount" type="xsd:integer"/>
<part name="id" type="xsd:string"/>
</message>
<message name="riskAssessmentMessage">
<part name="level" type="xsd:string"/>
</message>
<portType name="loanServicePT">
<operation name="request">
<input message="lns:creditInformationMessage"/>
<output message="lns:approvalMessage"/>
<fault name="unableToHandleRequest" message="lns:errorMessage"/>
</operation>
</portType>
<portType name="loanApprovalCallbackPT">
<operation name="approveCallback">
<input message="lns:approvalMessage"/>
</operation>
</portType>
<portType name="riskAssessmentPT">
<operation name="check">
<input message="lns:creditInformationMessage"/>
<output message="lns:riskAssessmentMessage"/>
<fault name="loanProcessFault" message="lns:errorMessage"/>
</operation>
</portType>
```

```
<portType name="loanApprovalPT">
<operation name="approve">
<input message="lns:creditInformationMessage"/>
</operation>
</portType>
<plnk:partnerLinkType name="loanPartnerLinkType">
<plnk:role name="loanService">
<plnk:portType name="lns:loanServicePT"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="loanApprovalLinkType">
<plnk:role name="approverCallback">
<plnk:portType name="lns:loanApprovalCallbackPT"/>
</plnk:role>
<plnk:role name="approver">
<plnk:portType name="lns:loanApprovalPT"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="riskAssessmentLinkType">
<plnk:role name="assessor">
<plnk:portType name="lns:riskAssessmentPT"/>
</plnk:role>
</plnk:partnerLinkType>
<bpws:property name="id" type="xsd:string"/>
<bpws:propertyAlias messageType="lns:approvalMessage" part="id" propertyName="lns:id"/>
<bpws:propertyAlias messageType="lns:creditInformationMessage" part="id" propertyName="lns:id"/>
</definitions>
```

7.3.3 Reise buchen

Flight.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:tns="http://flight.org/wsdl/flight-book"
xmlns:xst="http://travel.org/schema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
argetNamespace="http://flight.org/wsdl/flight-book">
<types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import namespace="http://travel.org/schema" schemaLocation="travel.xsd"/>
</xsd:schema>
</types>
<message name="bookRequestMessage">
<part name="pID" type="xsd:string"/>
<part name="arrival" type="xsd:date"/>
<part name="departure" type="xsd:date"/>
</message>
<message name="bookReplyMessage">
<part name="price" type="xsd:float"/>
</message>
<message name="cancelRequestMessage">
```

```
<part name="pID" type="xsd:string"/>
</message>
<message name="cancelReplyMessage">
<part name="confirmation" type="xsd:boolean"/>
</message>
<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>
<portType name="FlightServicePT">
<operation name="bookFlight">
<input name="bookRequest" message="tns:bookRequestMessage"/>
<output name="bookReply" message="tns:bookReplyMessage"/>
<fault name="bookFlightFault" message="tns:errorMessage"/>
<fault name="bookFlightFaultNoSeats" message="tns:errorMessage"/>
<fault name="bookFlightFaultNoFlight" message="tns:errorMessage"/>
<fault name="bookFlightFaultNoAirport" message="tns:errorMessage"/>
<fault name="bookFlightFaultNoConnection" message="tns:errorMessage"/>
</operation>
<operation name="cancelFlight">
<input name="cancelRequest" message="tns:cancelRequestMessage"/>
<output name="cancelReply" message="tns:cancelReplyMessage"/>
<fault name="cancelFlightFault" message="tns:errorMessage"/>
</operation>
</portType>
<plnk:partnerLinkType name="FlightServicePLT">
<plnk:role name="FlightService">
<plnk:portType name="tns:FlightServicePT"/>
</plnk:role>
</plnk:partnerLinkType>
</definitions>
```

Hotel.wsdl

```
<?xml version="1.0" encoding="utf-8"?>
<definitions targetNamespace="http://hotel.org/wsdl/hotel-book"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:tns="http://hotel.org/wsdl/hotel-book" xmlns:xst="http://travel.org/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import namespace="http://travel.org/schema" schemaLocation="travel .xsd"/>
</xsd:schema>
</types>
<message name="bookRequestMessage">
<part name="pID" type="xsd:string"/>
<part name="arrival" type="xsd:date"/>
<part name="departure" type="xsd:date"/>
</message>
<message name="bookReplyMessage">
<part name="price" type="xsd:float"/>
</message>
<message name="cancelRequestMessage">
```

```
<part name="pID" type="xsd:string"/>
</message>
<message name="cancelReplyMessage">
<part name="confirmation" type="xsd:boolean"/>
</message>
<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>
<portType name="HotelServicePT">
<operation name="bookRoom">
<input message="tns:bookRequestMessage" name="bookRequest"/>
<output message="tns:bookReplyMessage" name="bookReply"/>
<fault message="tns:errorMessage" name="bookHotelFault"/>
</operation>
<operation name="cancelOrder">
<input message="tns:cancelRequestMessage" name="cancelRequest"/>
<output message="tns:cancelReplyMessage" name="cancelReply"></output>
<fault message="tns:errorMessage" name="cancelHotelFault"/>
</operation>
</portType>
<plnk:partnerLinkType name="hotelServicePLT">
<plnk:role name="hotelService">
<plnk:portType name="tns:HotelServicePT"/>
</plnk:role>
</plnk:partnerLinkType>
</definitions>
```

Travel.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.example.com/process99585609/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:tns="http://www.example.com/process99585609/"
xmlns:wsdl0="http://hotel.org/wsdl/hotel-book"
xmlns:wsdl1="http://www.example.com/process99585609/interface/"
xmlns:wsdl2="http://flight.org/wsdl/flight-book">
<import location="TravelInterface.wsdl"
namespace="http://www.example.com/process99585609/interface/"/>
<import location="flight.wsdl" namespace="http://flight.org/wsdl/flight-book"/>
<import location="hotel.wsdl" namespace="http://hotel.org/wsdl/hotel-book"/>
</definitions>
```

Travelinterface.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.example.com/process99585609/interface/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://www.example.com/process99585609/interface/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
```

```
<message name="input">
<part name="pID" type="xsd:string"/>
<part name="arrival" type="xsd:date"/>
<part name="departure" type="xsd:date"/>
</message>
<message name="output">
<part name="message" type="xsd:string"/>
</message>
<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>
<portType name="travelServicePT">
<operation name="process">
<input name="input" message="tns:input"/>
<output name="output" message="tns:output"/>
<fault message="tns:errorMessage" name="timeoutFault"/>
</operation>
</portType>
<plnk:partnerLinkType name="travelServicePLT">
<plnk:role name="travelService">
<plnk:portType name="tns:travelServicePT"/>
</plnk:role>
</plnk:partnerLinkType>
</definitions>
```

Literatur

[1] [**Leymann et al, 2003**] - BPEL4WS,
Version 1.1, May 2003, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (Abruf: 08. Juli 2005)

[2] [**Leymann, 2005**] - Vorlesung Workflowmanagement SS2005

[3] [**Eclipse.org**] - Homepage von Eclipse, aktuelle Stable-Version 3.1 bzw. Milestone-Version 3.2M1, http://www.eclipse.org (Abruf: 12. September 2005)

[4] [**Nickolaos Kavantzas** (Designer and Lead-Editor of WS-CDL)] - WS-CDL-April 2004.pdf http://lists.w3.org/Archives/Public/www-archive/2004Jun/att-0008/WS-CDL-April2004.pdf (Abruf: 15. August 2005)

[5] [**pi4soa**] - Version 1.2
http://www.pi4soa.org (Abruf: 07. September 2005]

[6] [**WSCDL-Eclipse**] - Version 1.03
https://sourceforge.net/projects/wscdl-eclipse/ (Abruf: 07. September 2005)

[7] [**Barros et al**] - A critical overview of the Web Services Choreography Description Language (WS-CDL), http://www.bptrends.com -> Publications
http://www.bptrends.com/deliver_file.cfm?fileType=publication&fileName=03%2D05%20WP%20WS%2DCDL%20Barros%20et%20al%2Epdf (Abruf: 02. September 2005)

[8] [**WS-CDL Charter**] - Charta der WS-CDL Working Group
http://www.w3.org/2005/01/wscwg-charter.html (Abruf: 02. September 2005)

[9] [**Web Services Choreography Requirements**] - W3C Working Draft 11 March 2004
http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311/ (Abruf: 17. August 2005)

[10] [**Web Services Glossary**]
http://www.w3.org/TR/ws-gloss/ (Abruf: 20. August 2005)

[11] [**pi4soa Release Notes**] - Version 1.2.0
http://prdownloads.sourceforge.net/pi4soa/pi4soa1.2.0-ReleaseNote.pdf?download

[12] [**WS-CDL**] Version 1.0
http://www.w3.org/TR/ws-cdl-10/ (Abruf: 08. Juli 2005)

[13] [**IBM**] WebSphere Business Integration Modeler V5.1
http://www-128.ibm.com/developerworks/downloads/ws/wbimod/?S_TACT=105AGX28&S_CMP=TR
(Abruf: 10. September 2005)

[14] [**piCalculus**]
http://www.ebpml.org/pi-calculus.htm (Abruf: 03. September 2005)

[15] [**Einführung in BPEL**]
http://www.oio.de/public/xml/einfuehrung-in-bpel4ws.htm (Abruf: 25. Juli 2005)

[16] [**Wikipedia Eintrag zu BPEL**]
http://de.wikipedia.org/wiki/BPEL (Abruf: 25. Juli 2005)

[17] [**BPEL4WS-Coverpage**]
http://xml.coverpages.org/bpel4ws.html (Abruf: 25. Juli 2005)

[18] [**WSFL-Coverpage**]
http://xml.coverpages.org/wsfl.html (Abruf: 25. Juli 2005)

[19] [**XLANG-Coverpage**]
http://xml.coverpages.org/xlang.html (Abruf: 25. Juli 2005)

[20] [**ebpml.org - WSFL**]
http://www.ebpml.org/wsfl.htm (Abruf: 25. Juli 2005)

[21] [**ebpml.org - XLANG**]
http://www.ebpml.org/xlang.htm (Abruf: 25. Juli 2005)

[22] [**Wikipedia Eintrag zu WSFL**]
http://de.wikipedia.org/wiki/WSFL (Abruf: 25. Juli 2005)

[23] [**Leymann, 2001**] - WSFL-Spezifikation
http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf (Abruf: 25. Juli 2005)

[24] [**G. Bischoff, R. Kersten, T. Vetter**] - Vergleich von BPEL-Workflow Modellierungstools
http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=FACH-0038&inst=FAK&mod=&engl= (Abruf: 11. September 2005)

[25] [**Studienplan Infomatik/Softwaretechnik**] - Stand Oktober 2005
http://www.informatik.uni-stuttgart.de/fakultaet/lehre/softwaretechnik/studium/studienplan.html (Abruf: 27. September 2005)

[26] [**Buyer/Seller Guide**] - Understanding WS-CDL (Guide2WS-CDLV3.ppt)
http://www.pi4tech.com -> Download -> Presentations (Nur für registrierte Benutzer) (Abruf: 23. September 2005)

[27] [**Buyer/Seller Example**] - pi4soa 1.2.0 Examples (pi4soa1.2.0-examples.zip)
http://www.pi4tech.com -> Download -> pi4soa Examples (Nur für registrierte Benutzer) (Abruf: 23. September 2005)

[28] [**pi4soa Patch**] - Patch1 vom 23. September 2005
http://switch.dl.sourceforge.net/sourceforge/pi4soa/pi4soa-designer1.2.0-patch1.zip

8 Nachbetrachtung der Fachstudie

Laut Studienplan ist eine Fachstudie folgendermaßen definiert:

„Die Fachstudie (4 P) wird dezentral durchgeführt, die Themen werden von den Abteilungen bekanntgegeben. In der Regel arbeitet eine Gruppe von drei Studierenden an einem konkreten, realen Thema, das vielfach von einem externen Partner gegeben ist, beispielsweise: Sollten wir zukünftig dieses Werkzeug, jenes Verfahren einsetzen? Welcher Aufwand wäre notwendig, welcher Nutzen erzielbar, wenn wir unseren Entwicklungsprozess in bestimmter Richtung änderten? Die vorurteilsfreie Klärung der Frage und die Beantwortung durch einen Bericht und eine Präsentation sind die Voraussetzungen für den unbenoteten Schein.“ [26]

Um den Stand bezüglicher der Aufgabenstellung wiederzugeben wollen wir diese Fachstudie mit einem kurzen Soll-Ist-Vergleich abschließen.

8.1 Vergleich Ist <-> Soll

Aus 1.1 ergeben sich mit etwas Interpretation die zu erledigenden Aufgaben:

1. Einarbeitung in WS-CDL + BPEL4WS
2. Herausarbeiten von Differenzen, Gemeinsamkeiten (Überlappungen) WS-CDL <-> BPEL4WS
3. Erarbeitung verschiedener Szenarien (Orientierungshilfe: nur ein Prozess, mehrere Prozesse, RosettaNet)
4. Recherche und Auffinden von WS-CDL-Tools (soweit vorhanden)
5. Evaluierung von u.U. gefundenen WS-CDL-Tools
6. Bewertung und Ausblick WS-CDL

zu 1.) Wurde mit relativ großem Zeitaufwand realisiert. Es ist zu bemerken, dass ohne vorherige Beschäftigung mit dem allgemeinen Thema der „Web Services“ und der damit zusammenhängenden Technologien (Sprachen) eine Einarbeitung (nur) in BPEL4WS und (nur) in WS-CDL praktisch unmöglich ist, da so kein gröberes (Gesamt)Bild zu erkennen und zu verstehen ist. Deshalb beschränkte sich die Einarbeitung nicht nur auf die „Spezifikationen“ von BPEL4WS und WS-CDL, sondern auch auf viele weitere allgemeine Dokumente und weitere Spezifikationen (z.B. WSFL). In den Literaturangaben sind nur all diejenigen Dokumente zu finden, die entweder in dieser Fachstudie zitiert/benutzt wurden oder deren Inhalt entscheidend zum Verständnis beigetragen haben - es wurden aber weitaus mehr Dokumente gesichtet, gelesen, bewertet und dann z.T. auch wieder verworfen. – Stand: erledigt

zu 2.) Wir denken, dass dies im Kapitel WS-CDL vs. BPEL4WS in ausreichender Form geschehen ist. – **Stand: erledigt**

zu 3.)

- Szenarien - einfach strukturierte Szenarien bzw. beispielhafte Repräsentationen von Szenarien vorhanden. –**Stand: teilweise erledigt**

- Szenarien in BPEL - da BPEL nicht im Zentrum der Betrachtung stand, wurde auf bereits erarbeitete Resultate zurückgegriffen. –**Stand: sozusagen erledigt**
- Szenarien in WS-CDL - einfach strukturierte Szenarien bzw. beispielhafte Repräsentationen von Szenarien vorhanden. Verwendung von wenigen Konstrukten. –**Stand: teilweise erledigt**
- RosettaNet - wurde aufgrund der zeitintensiven Einarbeitung in WS-CDL und BPEL4WS nicht betrachtet. –**Stand: nicht erledigt**

zu 4. + 5.) Trotz langer Suche konnten im Wesentlichen eigentlich nur **zwei** Tools gefunden werden, die den Anspruch haben WS-CDL (Modellierungs-)Tools zu sein. Natürlich könnte man, wenn man von einer gewissen Unterstützung absehen würde, nahezu jeden beliebigen XML-Editor als WS-CDL-Tool klassifizieren - was jedoch keinen Sinn macht. Die gefundenen Tools sind mehr oder weniger geeignet den User/Experten bei der Erzeugung von Choreographies zu unterstützen, aber ein echter Zugewinn durch die Tools ist aufgrund des inherenten Problems der fehlenden graphischen Notation nicht zu verzeichnen, wobei pi4soa hier zu einem gewissen Teil auszunehmen ist, da eine erste graphische Notation implementiert wurde, die zumindest teilweise eine graphische Modellierung zulässt. Wir denken dies wurde im Kapitel 4 über WS-CDL (unter Out of Scope) bzw. im Kapitel 4.6.2 pi4soa ausführlich thematisiert. – **Stand: erledigt**

zu 6.) Wir denken dies wurde im Kapitel 4 WS-CDL erledigt. – **Stand: erledigt**

Erklärung

Wir erklären hiermit, dass wir die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet haben.

Stuttgart, den 03. Oktober 2005

_____

(Ingo Hildebrandt)

_____

(Marc Wagner)

_____

(Carsten Stützner)