Dynamische Prozessintegration von e-Procurement Web Services unter Verwendung semantischer Technologien

am Beispiel von Web Services
zum Austausch von openTRANS – Geschäftsdokumenten

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom Ingenieur - Medieninformatik (FH)

Fraunhofer IAO

11.05.2004

Betreuer: Prof. Walter Kriha
Alexander Hettrich

Diplomand: Christoph Diefenthal

eine Diplomarbeit im Studiengang Medieninformatik der Fachhochschule Stuttgart,
Hochschule der Medien (HdM)

Erklärung

Ich erkläre hiermit, die vorliegende Arbeit selbständig verfasst und keine anderen, als die angegebenen Hilfsmittel verwendet zu haben.

Die Diplomarbeit wurde noch keiner Prüfungsbehörde in gleicher oder anderer Form vorgelegt.

Stuttgart, den

Christoph Diefenthal

Danksagungen

Vielen Dank an Alexander Hettrich und Markus Lebender vom Fraunhofer IAO für den Themenvorschlag, die Betreuung und die Hinweise bei der Umsetzung dieser Diplomarbeit.

Meiner Familie danke ich für ihre Unterstützung und Finanzierung meines Studiums.

Bedanken möchte ich mich weiterhin bei Julia Rissler, Martin Anita Kraus und Sebastian Platz für die Korrekturen.

INHALT

1 EINLEITUNG

1.1 Überblick

Diese Arbeit beschäftigt sich übergreifend mit der automatisierten, dynamischen Prozessintegration von Web Services.

Sie behandelt im speziellen Fall den automatisierten Prozessabgleich von Web Services, die dem Austausch von Geschäftsdokumenten im openTRANS-Format [opT1] dienen. Semantische Technologien beschreiben die Bedeutung der Web Service-Schnittstellen und ermöglichen so einem Integrationssystem zu einer gemeinsamen, komponierte Prozessbeschreibung zu gelangen. Eine komponierten Prozessbeschreibung kann dann mit Hilfe von „Process Execution Languages“ als mittelnder Web Service zwischen den Einkaufsystemen gestartet werden.

Um eine Integration der Systeme von Geschäftspartnern überhaupt zu ermöglichen, ist es notwendig sich auf die Datenstrukturen zu einigen, die ausgetauscht werden. Dazu zählen Produktdaten, ganze Geschäftsdokumente und verwendeten Vokabularien, Konstanten, wie Länderkennzeichnungen und Datentypen. Geschäftsdokumentsstandards, die wie openTRANS auf XML beruhen, bieten die Möglichkeit des Austauschs von einheitlichen Dokumenten, die von Menschen wie Maschinen verstanden werden können.

Möglichkeiten, die Daten elektronisch zu übermitteln, gibt es viele: FTP, Email, HTTP, um nur einige zu nennen. In vielen großen Unternehmen werden aufwendige Eigenentwicklungen eingesetzt, um sichere und verlässliche Kommunikation mit Geschäftspartnern zu vollziehen [vgl. Schubert1 S.16, Haertsch1 , S. 9-10]. Diese Systeme müssen von Hand integriert werden, wodurch meist eine langfristige Bindung an einen Geschäftspartner entsteht. Eine Integration von Unternehmenssystemen „on the fly“, also eine Integration, die mit einem potenziellen Geschäftspartner erst zu dem Zeitpunkt durchgeführt wird, wenn sie benötigt wird, ist bis heute nicht möglich. Web Services bieten hier durch ihr losgelöstes Komponentendasein und ihre maschinenverständliche Selbstbeschreibung durch WSDL völlig neue Möglichkeiten der Integration.

WSDL beschreibt die Schnittstellen eines Web Service. Durch die Namensgebung ist einem menschlichen Betrachter meistens ersichtlich, welche Bedeutung die einzelnen Operationen und Parameter des Web Service besitzen. Ein Integrationssystem, dass eine Integration „on the fly“ von Web Services zweier Geschäftspartner umsetzen soll, besitzt ein solches Verständnis nicht. Hier können semantische Technologien die Bedeutung der in WSDL beschriebenen Schnittstellen für das Integrationssystem maschinenverständlich aufbereiten.

Zudem müssen die Prozessabläufe auf beiden Seiten aufeinander abgeglichen werden, damit der Austausch von Geschäftsdokumenten mit dem Geschäftspartner

reibungslos funktionieren kann. Was ist zum Beispiel, wenn der Einkäufer davon ausgeht, dass er eine Lieferavis erhält, der Verkäufer aber keine versendet? Auch diese Bedingungen, die Geschäftspartner aneinander stellen, können mit semantischen Technologien in einer potenziellen Prozessbeschreibung erläutert werden.

„Process Execution Languages" (PEL) wie BPEL4WS [BPEL4WS1] bieten heute die Möglichkeit Web Services in gemeinsamen Prozessabläufen miteinander zu integrieren. Eine ausführbare Prozessbeschreibung in einer PEL wird auf einer passenden Engine wiederrum als Web Service gestartet. So können beispielsweise ein Flugbuchungs-Web Service und ein Hotelbuchungs-Web Service miteinander integriert, und als ein einzelner Web Service angeboten werden. Ein Nutzer kann dann den einen Web Service aufrufen und mit ihm interagieren.

Genauso können auch Einkaufs- und Verkaufs-Web Service miteinander integriert werden. Ein Integrationssystem kann anhand der potenziellen Prozessbeschreibungen von zwei Geschäftspartnern überprüfen, ob ein gemeinsamer Prozessablauf gefunden werden kann. Ist eine Komposition möglich wird eine gemeinsame Prozessbeschreibung in einer PEL generiert, die dann als mittelnder Web Service zwischen den Systemen der Geschäftspartner fungiert.

1.2 Problemstellung

Durch die zunehmende Leistungsfähigkeit der Computer- und Netzwerktechnik verändert sich die traditionelle Wirtschaft zu einer immer stärker ereignisgesteuerten Wirtschaft (eventdriven economy). Ein realisierter Bedarf soll innerhalb von Sekunden gestillt werden können. Dazu sind grundlegende Anpassungen nötig, ohne die eine solche digital automatisierte „new economy" nicht vorstellbar ist [vgl. Linthicum1, S.3 ff].

Zudem wird von Unternehmen gewünscht, nicht langfristig an Lieferanten gebunden zu sein. Ein schneller Wechsel zwischen den Verkäufern soll möglich werden. [vgl. Trastour1] Des weiteren soll die Beschaffung indirekter Produkte, auch MRO-Produkte (Maintenance Repair Operating) genannt, die abhängig von Preis, Qualität u.ä von verschiedensten Lieferanten geordert werden, aus Kostengründen und Gründen der Prozessoptimierung elektronisch abzuwickeln sein [vgl. Dolmetsch1, S.9-14; Hentrich1, S. 33]. Hieraus ensteht der Bedarf nach einer dynamischen Integration „on the fly" der dazu von Ein- und Verkäufer gestellten elekronischen Systeme, bei denen kein menschiches Eingreifen mehr notwendig ist.

Um e-Procurement in dieser Form umzusetzen, sind zum einen innerhalb der Unternehmen Systeme nötig, die eine durchgängige elektronische Abwicklung der Beschaffung erlauben um teure Medienbrüche zu verhindern. [vgl. Dolmetsch1, S.9-14]. Zum anderen müssen nach außen hin diese Systeme eine einfache und dynamische Business to Business Integration (B2Bi) gewährleisten, um sich mit den verschiedenen Systemen der Anbieter verknüpfen zu können.

Im speziellen Fall des e-Procurement lassen sich hier die Rollen des Einkäufers und des Verkäufers identifizieren. Ein Einkäufer stellt den Bedarf an einem Produkt fest

und sucht sich einen passenden Verkäufer. Ist dieser gefunden, muss die B2Bi angestossen werden, um den Einkaufsprozess elektronisch abwickeln zu können. Dieser Vorgang ist in Abbildung 1.1 dargestellt.

In der B2Bi werden die Systeme der beiden Geschäftspartner miteinander verglichen und aufeinander abgestimmt. Geschäftsdokumente, Nachrichtenaustauschprotokolle und Sicherheitskonzepte müssen abgeglichen werden.

In diesen Punkten lassen sich durch die Verwendung von Standards schon Übereinstimmungen erzielen, bevor sich die Geschäftspartner überhaupt kennen, wie zum Beispiel die Verwendung von Web Services für den Datenaustausch und die Nutzung der openTRANS-Spezifikation als Standard für Geschäftsdokumente.

Trotzdem bleiben noch Fragen offen. Welche Schnittstellen sind für den Empfang

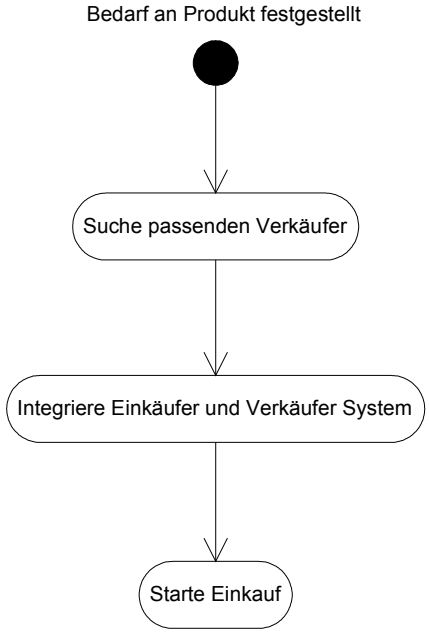

Abbildung 1.1 B2B Integration im Einkauf

welcher Dokumente zuständig? Hier müssen Beschreibungen gefunden werden, die die Bedeutung bzw. die Semantik dieser Schnittstellen maschinenverständlich erklären, damit Integration ohne menschliches Eingreifen vollzogen werden kann.

Es stellen sich zudem Fragen der Prozessintegration. Wie werden die Geschäftsdokumente in der gleichen Reihenfolge geschickt und empfängt jeder Geschäftspartner alle gewünschten Geschäftsdokumente? Hier muss ein einheitlicher Prozessablauf gefunden werden, um den Einkauf anstoßen zu können. Die Prozessabläufe der beiden Geschäftspartner können identisch, ähnlich – dass heißt sie können aufeinander abgeglichen werden – oder völlig inkompatibel sein. Um hier eine dynamische und automatisierte Integration möglich zu machen, werden exakte, maschinenverständliche Integrationsbeschreibungen der Prozesse in einer „Process Description Language" benötigt.

Auch wenn die Bezeichnung der einzelnen Schritte in einem Prozessablauf geklärt ist, kann ein Schritt in der Prozessbeschreibung trotzdem unterschiedliche Bedeutungen für die Teilnehmer besitzen. So kann eine Rechnung auch als Versandbenachrichtigung verstanden werden.

Solche semantischen Informationen müssen ebenfalls in einer Prozessbeschreibung enthalten sein.

Diese können maschinell aufeinander abgeglichen werden um so zu einer gemeinsamen Prozessbeschreibung zu gelangen, die festlegt, wann welche Geschäftdokumente zwischen den Geschäftspartnern ausgetauscht werden müssen bzw. können.

Diese Arbeit wird das Problem einer dynamischen Prozessintegration behandelt.

1.3 Ziel dieser Diplomarbeit

Diese Arbeit soll zum einen klären, welche Anforderungen an die Systeme gestellt werden, bei denen B2Bi betrieben werden soll. Welche Technologien werden benötigt und wo müssen Einigungen der Geschäftspartner erzielt werden. Sie beschränkt sich im weiteren auf einen Teilaspekt der B2Bi: die Prozessintegration der e-Procurement-Systeme von Einkäufer und Verkäufer.

Um diese Konzentration auf die Prozessintegration zu ermöglichen, werden für einige der Anforderungen Vorauswahlen getroffen. Für Geschäftsdokumente wird beispielsweise die openTRANS-Spezifikation herangezogen, für den Austausch derselben sollen Web Services verwendet werden.

Ein generisches Prozessintegrations-Konzept wird für diesen speziellen Fall entwickelt, dass die Integration von Einkäufer und Verkäufer Web Service zum Austausch der openTRANS-Geschäftsdokumenten in einem gemeinsamen Prozess ermöglicht.

Anpassungen dieser openTRANS-Web Services der Geschäftspartner an die Prozessintegration sollen wie bei jedweder B2Bi vermieden werden. Hierzu soll in dieser Arbeit das Konzept eines Mittlers zwischen Ein- und Verkäufer System verfolgt werden. Bei einer Prozessintegration muss anhand semantischer Informationen entschieden werden, wie und ob eine gemeinsame Prozessbeschreibung gefunden werden kann.

Die Prozessabläufe der Geschäftspartner müssen sich dabei selbst soweit mit einer potenziellen Prozessbeschreibung beschreiben, dass sie automatisch zu einer komponierten Prozessbeschreibung integriert werden können. Für die potenziellen Prozessbeschreibungen bedarf es einer „Process Description Language“ (PDL), die über die Bedeutung der Schnittstellen der openTRANS-Web Services Auskunft geben. Semantische Technologien bieten sich hier an.

Diese komponierte Prozessbeschreibung soll daraufhin als Web Service umgesetzt werden, der als Verbindungspunkt für Ein- und Verkäufer fungiert. Dieser mittelnde Prozess-Web Service – im weiteren Flow-Service genannt – regelt, welche Nachrichten an welche Schnittstelle eines Geschäftspartners gesendet werden, und

überwacht den Prozessablauf. So darf er immer nur die Nachrichten übermitteln, die zu einem bestimmten Zeitpunkt des Ablaufs erlaubt sind.

Weiterhin müssen die Mindestanforderungen geklärt sein, unter denen ein gemeinsamer Prozessablauf überhaupt möglich ist. Hieraus ergibt sich eine Metaprozessbeschreibung, an der sich die Prozessbeschreibungen der Geschäftspartner orientieren müssen.

Ein komplexes Zusammenwirken von Web Services kann erst mit einer auf ihren Technologien aufsetzenden Kontextdefinition erreicht werden. Um einen integrierten Prozess für die Web Services ansprechbar zu machen, muss er ebenfalls als Web Service gestartet werden. „Process Execution Languages" (PEL) dienen der Komposition von Web Services. Prozessbeschreibungen in einer PEL können als Web Services auf passenden Engines gestartet werden. Eine passende PEL muss für die Komposition der Geschäftspartner Web Services gefunden werden.

Letztendlich wird ein Prototyp entwickelt, der die grundlegenden Eigenschaften der Prozessintegration, die im Lösungskonzept gefunden wurden, umsetzen soll.

1.4 Aufbau der Diplomarbeit

e-Procurement

Im ersten Kapitel wird erläutert, welche Basismechanismen der Wirtschaft für den elektronischen Handel nötig sind, was e-Procurement bedeutet, und welche Anforderungen an eine Business -to-Business-Integration gestellt werden, von der ein Teilaspekt die Prozessintegration ist.

Basistechnologien

Im weiteren werden einige Basistechnologien vorgestellt, mit denen eine solche Prozessintegration vorgenommen werden kann. Dazu gehören zum einen die Technologien, die die zu integrierenden Systeme benutzen: openTRANS und Web Services, und zum anderen die Technologien, mit denen eine Prozessintegration umgesetzt werden soll: semantische Technologien, die für eine „Process Description Language" bereitstellen und „Process Execution Languages", die der Komposition von Web Services dienen.

Lösungskonzept und Prototyp

Im Lösungskonzept wird die Konzentration auf die Analyse einer Prozessintegration von Web Sevices für den Austausch von openTRANS Geschäftsdokumenten gelegt. Es wird geklärt, wie semantische Technologien verwendet werden könne, um die Bedeutung der Web Service Schnittstellen zu beschreiben. Weiterhin werden die Mindestanforderungen erörtert, die erfüllt sein müssen, um eine Prozessintegration möglich zu machen. Aus den gefundenen Bedingungen wird eine Metaprozessbeschreibung gewonnen, die angibt, welche Mindestbedingungen ein Prozess erfüllen muss. Die Prozessbeschreibungen der Geschäftspartner können entweder

exakt gleich sein, sich in wenigen kompensierbaren Punkten unterscheiden, oder völlig inkompatibel sein.

Hierzu wird ein Prototyp entwickelt, der die grundlegenden Ergebnisse des Lösungskonzept umsetzt, um seine Realisierbarkeit aufzuzeigen.

2 E-PROCUREMENT

e-Business, e-Procurement und weitere e-Begriffe haben in den letzten Jahren immer mehr an Bedeutung gewonnen. Grundsätzlich ist allen Unternehmen bekannt, dass es sich hierbei um die Einführung und den Einsatz neuer Informations- und Kommunikationstechnologien zur Verbesserung der bisherigen Infrastruktur handelt [Kalakota1 S. 6]. Viele Unternehmen sind bereits auf dem Weg ins e-Business, andere zögern noch mit dem Einsatz neuer Technologien.

In diesem Kapitel soll geklärt werden, welche Begrifflichkeiten wichtig für die Einordnung dieser Arbeit sind, und wie ein Beschaffungsprozess prinzipiell funktioniert.

Letztendlich werden die Anforderungen erstellt, die sich an eine dynamische B2B-Integration ergeben, und welche Bedingungen für eine dynamische Prozessintegration erfüllt sein müssen, mit der sich diese Arbeit beschäftigt.

2.1 Basismechanismen der Wirtschaft

Zur Umsetzung einer Prozessintegration für Einkäufer und Verkäufer muss zuerst geklärt werden, wie ein Beschaffungsprozess abläuft, und an welcher Stelle die Prozessintegration betrieben werden muss.

In der Wirtschaft lassen sich drei Basismechanismen erkennen, die Transaktionen koordinieren: Netzwerke, Hierarchien und Märkte.[vgl. Williamson1, S. 347-370]

Bei Netzwerken werden Transaktionen vor allem durch kooperative und weniger durch kompetitive Mechanismen geregelt. Bei hierarchischen Strukturen wird durch Kontrollausübung von höheren Schichten eine Koordination von Transaktionen erreicht, während in Märkten Angebot und Nachfrage diese Kontrolle ausüben.

Diese Arbeit ist durch seine Zielsetzung auf dynamische „on the fly" Integrationen von Einkaufsprozessen in den Bereich des Marktmechanismus einzuordnen. Daher wird dieser Mechanismus im weiteren genauer betrachtet. Um die Koordinationsfähigkeit der Märkte zu optimieren und an der ereignisgesteuerte Wirtschaft auszurichten, wird es für die Unternehmen immer wichtiger an die B2B-Kommunikation angepasste Standards zu verwenden, um die typische Markttransaktionen zu unterstützen.

2.1.1 Ablauf einer Markttransaktionen

Eine Markttransaktion – ein Einkauf – besteht aus einer begrenzten Zahl von Interaktionen zwischen den Geschäftspartnern.

> *„Ziel ist die Vereinbarung und der Austausch von Verfügungsrechten an einer Marktleistung in den drei Schritten Anbahnung durch Information, Vereinbarung und Abwicklung“.* [Dolmetsch1, S.30]

Bei einer zeitlichen Einteilung einer Markttransaktion lässt sich eine Gliederung in mehr oder weniger Phasen vornehmen. Um dem openTRANS-Format gerecht zu werden, lässt sich die oben genannte erste Phase in Wissensphase und Absichtsphase unterteilen, womit hier letztendlich vier Phasen unterschieden werden:

- Wissensphase
- Absichtsphase
- Vereinbarungsphase
- Abwicklungsphase

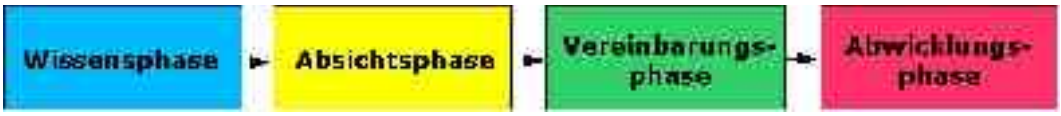

Abbildung 2.1 Markttransaktion

In der Wissenphase informieren sich Anbieter und Nachfrager über die angebotenen Produkte, deren Preise und über die potenziellen Geschäftspartner. Prinzipiell wird eine Markttransaktion immer auf Initiative des späteren Einkäufers begonnen. Er stellt einen Bedarf fest und startet die Wissensphase.

Darauf folgt die Absichtsphase, in der der potenzielle Kunde und auch der Anbieter ihre Bereitschaft signalisieren, miteinander ins Geschäft zu kommen. Hier kann ein reger Informationsaustausch stattfinden. Angebote und Anfragen wechseln zwischen den potenziellen Geschäftspartner hin und her.

In der Vereinbarungsphase werden die für beide rechtlich verbindlichen Bedingungen für ein Geschäft festgelegt. Mit der Zustimmung beider Parteien tritt der Vertrag in Kraft und es kommt zur Abwicklungsphase. In dieser Phase wird der eigentliche Austausch von Gütern im Rahmen des abgeschlossenen Vertrages vorgenommen.

Diese Arbeit beschäftigt sich mit der Absichts-, Vereinbarungs- und Abwicklungsphase eines Einkaufs. Wie in Kapitel 3.1.2 noch zu sehen ist, bietet openTRANS Geschäftsdokumente für Angebotsanforderung (RFQ) und Angebot (QUOTATION), welche in die Absichtsphase einzuordnen sind, sowie weitere für die folgenden Phasen. Standards oder Geschäftsdokumente, die in der Wissensphase zu nutzen wären, werden von openTRANS nicht angeboten, da der Such- und Informationsprozess (Wissensphase) nicht berücksichtigt wird.

Im Rahmen der B2B-Protokolle entspricht die Wissenphase der Ebene des Auffindens und Auswahl die für das Lösungskonzept (Kapitel 4) als abgeschlossen betrachtet wird. Es geht ausschließlich um die Absichtserklärung, das Versenden der Bestellung und deren Annahme (Vereinbarungsphase) und die darauffolgende Abwicklung des Einkaufs.

2.1.2 Ablauf des Einkaufs

Der Ablauf der Markttransaktion nach der Wissensphase wird hier als Ablauf des Einkaufs bezeichnet, da mit Absichts-, Vereinbarungs- und Abwicklungsphase der Austausch von Informationen zum Zwecke der Übertragung der Rechte an einem Produkt beginnt.

Der Ablauf des Einkaufs ist abhängig von der Komplexität der Nachrichten, die sich Ein- und Verkäufer in Form von Geschäftsdokumenten zukommen lassen, um die Markttransaktion fortzusetzen.

Da sich diese Arbeit mit dem organsisatorischen Charakter des Einkaufs befasst, sind Teile der Abwicklung wie die Bezahlung, die Statusabfrage oder eine Statusbenachrichtigung über eine Sendungsverzögerung nicht im Ablauf berücksichtigt.

Grundsätzlich muss der Einkäufer eine Bestellung verschicken und der Verkäufer dieselbe empfangen können. Das ist die Basis jeder Abwicklungsphase einer Markttransaktion. Darauf aufbauend kann die Abwicklungsphase unterschiedlich komplex werden. So kann es erlaubt sein, dass eine Bestellung geändert oder annulliert wird, nachdem sie abgeschickt wurde. Des weiteren kann ein Einkäufer eine Bestätigung der Bestellung, der Bestellungsänderung und der Annullierung verlangen.

Wird die Bestellung nicht abgebrochen, folgt als nächster Schritt das Abpacken und der Versand der Ware. Darüber möchte ein Einkäufer unter Umständen auch informiert werden. Im Gegenzug kann der Verkäufer eine Empfangsbestätigung verlangen. Davor, während dessen, oder danach wird der Verkäufer eine Rechnung stellen.

Die typischen Schritte und demnach benötigte Geschäftsdokumente sind demnach:

- Angebotsanforderung (durch den Einkäufer E)
- Angebot (durch den Verkäufer V)
- Bestellung (E)
- Bestellungsbestätigung (V)
- Änderung der Bestellung (E, V)
- Bestätigung der Änderung (E, V)
- Annullierung der Bestellung (E, V)
- Bestätigung der Annullierung (E, V)
- Versandbenachrichtigung (V)
- Eingangsbenachrichtigung (E)
- Rechnung (V)

Listing 2.1 typische Schritte einer Bestellung

Wie bereits bemerkt müssen nicht alle Schritte umgesetzt werden. Eine Nachricht kann durchaus auch zwei Schritte zusammenfassen. So kann eine Rechnung auch als Versandbenachrichtigung verstanden werden.

Einige Nachrichten, wie die Änderung und die Bestätigung der Änderung, können innerhalb eines Einkaufskontextes mehr als einmal auftreten. Das hängt aber wiederum von den Wünschen der Geschäftspartner ab. Unter Umständen ist überhaupt keine Änderung erwünscht. Ausserdem treten bei der Umsetzung dieser Nachrichten in einen ausführbaren Einkaufsprozess weitere Abhängigkeiten der Nachrichten voneinander auf. Beispielsweise hat eine Bestätigung ohne eine Bestellung keinen Sinn. Eine Bestätigung kann je nach Wunsch der Geschäftspartner verpflichtend, optional oder nicht erlaubt sein. Ist eine Annullierung erfolgt, macht eine Bestellungsbestätigung keinen Sinn mehr u.ä.

Die oben stehenden Aspekte werden im Kapitel 4 im Rahmen des Lösungskonzepts eingehend behandelt.

2.2 Definitionen

Um die Themenbereiche genauer behandeln zu können, sollen an dieser Stelle arbeitsfähige Definitionen gefunden werden die zudem der Einordnung dieser Arbeit dienen.

2.2.1 e-Business

Für einige bedeutet e-Business webbasierter Verkauf und für andere Middleware. Die meisten halten e-Business für die nächste Gerneration von internetbasierter Beschaffungskettenintegration, die dort einsetzt wo EDI[1] seine Grenzen fand.

Es gibt keine wohldefinierte Abgrenzung der Begriffe. David S. Linthicum beispielsweise versucht sich an einer umfassenden, und dadurch abstrakten Definition: *„... whatever else e-Business means, it means using innovative technology to build global relationships and commerce.“*[Linthicum1]

Damit wird der Begriff e-Business als sehr allgemeines Konzept der Verwendung von IT in der globalen Wirtschaft verstanden.

E-Business wird von [Merz1, S. 17]. eher als Oberbegriff, statt als konkrete Bezeichnung einer bestimmten Technologie bezeichnet.

In dieser Arbeit soll es ausreichen e-Business als eben einen solchen Oberbegriff zu verstehen.

E-Business verfolgt generell

> *„...das Ziel der erhöhten Wertschöpfung und der Effizienzsteigerung sowie der Beschleunigung von Geschäftsprozessen.“*

[1] Electronic Data Interchange – ein standardisiertes Datenformat für den Austausch von Geschäftsinformationen über Computer-Netzwerke

und ist

> *„...die Bezeichnung für die elektronische Abwicklung aller Geschäftsprozesse mithilfe von Informations- und Kommunikationstechnologien (z.B. über das Internet/Intranet/Extranet)"* [Möhrstädt1, S. 20/21]

So lassen sich folgende Merkmale des e-Business Begriffs festmachen:

- e-Business umfasst alle Geschäftsaktivitäten in der Wirtschaft: alle Arten von Geschäftsprozessen, Handel und Management von Kundenbeziehungen und weiteren teils unternehmensinternen und unternehmensübergreifenden Aktivitäten.
- Zur Unterstützung der Aktivitäten werden Informations- und Kommunikationstechnologien verwendet.

Folgende Definition erscheint somit sinnvoll:

E-Business ist der Oberbegriff für alle Geschäftsaktivitäten zwischen Unternehmen oder zwischen Unternehmen und Kunden, die unternehmensintern und -extern von Informations- und Kommunikationstechnologie unterstützt werden.

Ein e-Business System ist dementsprechend ein Unternehmenssystem, das mit Hilfe von Informations- und Kommunikationstechnologien die Verbindung und Abstimmung aller Arten von Geschäftsprozesse, Handel und Management von Kundenbeziehungen zwischen Unternehmen unterstützt.

2.2.2 e-Procurement

E-Business behandelt wie oben festgestellt alle elektronischen Geschäftsbeziehungen und -prozesse, die sich auf die neuen Kommunikationstechnologien stützen und zudem zwischen Unternehmen und Kunden ablaufen können. Der Begriff e-Commerce grenzt in den meisten Definitionen den elektronischen Handel von Unternehmen zu Kunden, auch Business to Customer (B2C) genannt, von dem der Unternehmen untereinander ab, was auch als Business to Business (B2B) bezeichnet wird [vgl Möhrstädt1, S. 21].

E-Procurement versteht sich als die elektronische Beschaffung von Produkten von Unternehmen für Unternehmen. [vgl Schubert1, S. 2; Möhrstädt1, S. 22] Damit ordnet sich e-Procurement in den Bereich des B2B ein.

Somit lässt sich folgende Definition festlegen:

E-Procurement behandelt den konkreten Beschaffungsprozess zwischen Unternehmen, der unternehmensintern und -extern von Informations- und Kommunikationstechnologien unterstützt wird.

Ein e-Procurement-System ist dementsprechend ein Unternehmenssystem, das mit Hilfe von Informations- und Kommunikationstechnologien die Beschaffung zwischen zwei Unternehmen unterstützt.

2.3 Business to Business Integration

E-Procurement bezeichnet die elektronische Beschaffung zwischen Unternehmen (B2B). Um e-Procurement betreiben zu können müssen die Unternehmen ihre e-Procurement-Systeme miteinander integrieren. Den Vorgang einer solchen System-Integration wird Business to Business Integration – B2Bi – genannt.

Experten sind sich einig, dass B2B auch in Zukunft eine größere Rolle spielen wird als der B2C-Bereich[2].

Im Folgenden wird geklärt welche Probleme bei einer B2Bi auftreten können, und welche Anforderungen zu erfüllen sind, um diese Schwierigkeiten zu überwinden.

2.3.1 Elektronischer Austausch von Geschäftsdaten

Mit der Zeit entfernt man sich in der Unternehmenskommunikation immer weiter von den üblichen Kommunikationswegen wie Briefverkehr, Telefon und Fax hin zu den neuen Errungenschaften des digitalen Informationszeitalters [vgl. Dolmetsch1, S.9-18; Hentrich1, S. 24].

Gründen dafür sind

- die Schnelligkeit, mit der Computer Daten verarbeiten können,
- das Vermeiden manuelle Eingaben, bei denen immer wieder Fehler vorkommen, sowie
- Prozessoptimierung und
- Kosteneinsparungen.

Gerade bei der Beschaffung direkter Produkte, also von Materialien, die Unternehmen für die Herstellung ihrer Produkte oder den Vertrieb benötigen, sind zum Teil enge Supply Chain Management-Konzepte entstanden, die über Unternehmensgrenzen hinweg für eine funktionierende elektronische Kommunikation zwischen Unternehmen sorgen. Dabei wurden Standardisierungen vorgenommen, wie zum Beispiel die Einführung einheitliche Materialnummern, welche wiederum eine elektronische Verabeitung erleichtern. Weiterhin wurden erste B2Bi und EDI-Anbindungen (Electronic Data Interchange) realisiert [vgl. Dolmetsch1, S.9].

Dabei wurde vor allem auf gemeinsame Dokumenten-Standards, wie zum Beispiel EDIFACT, Wert gelegt. Um diese Art der Datenübertragungssysteme betreiben zu können, mussten allerdings teure eigene Netzwerke (VAN's) zwischen den Unternehmen errichtet werden. Zudem waren dies meist Einzelentwicklungen, die sich kaum in andere Systeme integrieren ließen [vgl. Schubert1, S.16; Haertsch1, S. 9-10].

[2] Siehe beispielsweise http://www.zew.de/de/presse/presse.php3?action=article_show&LFDNR=167

EDI-Anbindungen sind, wie bereits in Kapitel 1.2 festgestellt wurde, nicht flexibel genug, um eine ereignisgesteuerte Wirtschaft zu ermöglichen. Diese ist erst mit Durchsetzung eines gemeinsamen Netzwerks und einer gemeinsamen Syntax zum Datenaustausch umsetzbar geworden: Internet und XML.

2.3.2 Integration in und zwischen Unternehmen

Medienbrüche innerhalb der Unternehmenskommunikation minimieren die Effizienz der internen Prozesse und führen zu hohen Kosten. Um diese zu verhinden, wuchs innerhalb der Unternehmen der Bedarf nach Integration der einzelnen Anwendungen. [vgl Dolmetsch1, S.9-14]. Diese Entwicklung zeigt einige Parallelen zur B2Bi.

Enterprise Application Integration (EAI) stellt ein Lösungskonzept zur innerbetrieblichen Integration von Anwendungen (A2A, Application to Application) dar. So gibt EAI als zentrale Komponente eine Integrations-Middleware (siehe Abbildung 2.2) vor, mit der sich die einzelnen Anwendungen als einziges verbinden müssen. Dadurch wird gewährleistet, dass bei einer Änderung innerhalb der Anwendung nur eine Systemkomponente – die Middleware – angepasst werden muss.

Die Middleware stellt zudem einige Funktionalitäten bereit, die immer wieder benötigt werden. Dazu gehört beispielsweise die Vermittlung zwischen unterschiedlichen Kommunikationsprotokollen, die Konvertierung von Datenformaten und die Abwicklung von Transaktionen.

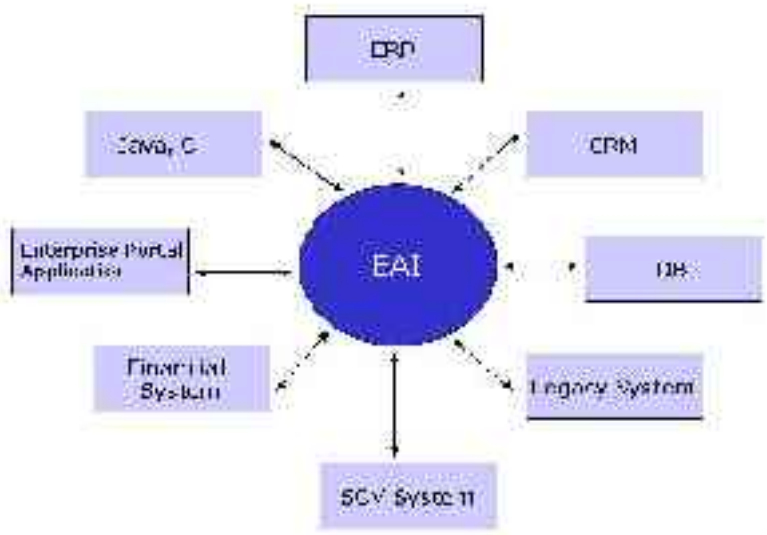

Abbildung 2.2 A2Ai Enterprise Application Integration

Auf diese Weise kann auch die technologische Umsetzung durch die Middelware-Komponente von der reinen Modellierung der Geschäftsprozesse getrennt werden. [vgl. BERL1, S. 17]

Obwohl EAI und B2Bi verschiedene Ziele verfolgen sind die Methoden und Technologien von EAI und B2Bi sehr ähnlich. EAI hat als Aufgabe verschiedene Anwendungen eines Unternehmens miteinander zu verknüpfen, B2Bi zielt darauf ab die Systeme von zwei oder mehr Unternehmen zu verbinden.

2.3.3 Anpassungen für die B2B Integration

Mit dem standardisierten Datenaustauschformat XML und dem Internet ist es nun auch kleinen Unternehmen möglich B2Bi kostengünstig zu betreiben. Das Internet, das vergleichsweise kostengünstig ist, verspricht im Gegensatz zu EDI und der Nutzung von VANs eine einfache Integration von allen daran angeschlossenen Unternehmen.

Das Internet bietet folgende Vorteile:

- Es erlaubt neben der reinen Datenübertragung bei EDI auch Interaktivität zwischen den Geschäftspartnern.
- Bei EDI müssen sich die Partner kennen bevor elektronische Transaktionen durchgeführt werden können, da erst ein Netzwerk aufgebaut werden muss. Ein solches Netz ist mit dem Internet bereits gegeben. An diesem sind bereits viele potenzielle Geschäftspartner angeschlossen, die sich dort präsentieren. Sie können über das Internet gefunden und direkt an das eigenen e-Business-System angeschlossen werden.

- Durch die steigende Verbreitung des Internet wächst die Anzahl potenzieller Kunden und Partner ständig an.

Mit Hilfe von XML als menschen- und maschinenverständlicher Standard wird die Möglichkeit gegeben Daten zwischen Unternehmen mit der gleichen Syntax auszutauschen [vgl.Schubert1, S.18; Dolmetsch1, S. 109].

Die Herausforderungen sind:

- Dem günstigen und für jeden erreichbare Kommunikationsmittel Internet fehlen wichtige Grundvorraussetzungen für einen fehlerfreien und verlässlichen Geschäftsverkehr, die in VANs implizit enthalten sind.
- XML bildet eine gemeinsame Syntax für den Datenaustausch, aber es müssen noch semantische Definitionen gefunden werden, die darauf aufbauend regeln wie die übermittelten XML-Strukturen zu verstehen sind [Bussler1].
- Auch die Übertragung von EAI auf B2Bi ist nicht ohne weiteres möglich. An die Stelle einer gemeinsamen konfigurierbaren Middleware, treten nun Standards, die den Informationsaustausch zwischen den Unternehmensanwendungen regeln.

Um diese Erweiterungen durch zuführen lassen sich die Anforderungen an B2Bi in folgenden vier Kriterien zusammenfassen[vgl. Bussler1, Linthicum1, S. 20; BERL1, S. 26]:

- **Unterstützung von B2B Standards**
- **Inter- und Intraprozessintegration**
- **Internetfähiger Informationsaustausch**
- **Weiterentwickelte Sicherheitskonzepte**

Abbildung 2.3 macht deutlich, auf welcher Ebene XML im Vergleich zur menschlichen Interaktion anzusiedeln ist. Die über XML liegenden Ebenen der Abbildung definieren die Bedeutungen von übermittelten XML-Strukturen. Das Vokabular legt auf Basis von XML Datentypen Konstanten zur Klassifikation oder auch Produktnamen fest. Dokument-Standards regeln, in welchen Formaten das Vokabular verwendet werden darf.

Abbildung 2.3 menschliche und B2B Interaktion (Quelle: http://www.rosettanet.org)

So wie die Middleware bei EAI als zentraler Verknüpfungsknoten fungiert, der verhindert, dass massenweise Punkt-zu-Punkt Verbindungen zwischen den Anwendungen entwickelt werden, so bewirken **B2B Standards**, dass nicht unzählige Übersetzungen in die Datenformate der jeweiligen Unternehmen durchzuführen sind. openTRANS steht hier beispielsweise als Standard für die auszutauschenden Geschäftsdokumente zur Verfügung (siehe Kapitel 3.1.2).

Als Grundvoraussetzung zur **Prozessintegration** der eigenen Systeme mit denen eines anderen Unternehmens müssen die internen Prozesse bereits integriert sein, um das System optimal nutzen zu können [vgl. Dolmetsch1, S.18].

Bei der Systemintegration zwischen Unternehmen spielen allerdings Anpassungsnotwendigkeiten eine Rolle, die bei EAI in diesem Maße nicht auftreten. Beispielsweise kann das Fremdsystem nicht an das eigene System angepasst werden, wenn ein Integrationsproblem auftaucht. Daher müssen die Systeme über unproblematische Schnittstellen miteinander verbunden werden, um die Interoperabilität zu erhalten.

Transaktions-Standards und Prozessdefinitionen, wie sie in Abbildung 2.3 oberhalb der Dokumenten-Standards dargestellt sind, müssen festgelegt werden. Sie definieren, in welcher Reihenfolge welche Dokumente wie oft übermittelt werden.

Die Nicht-Verlässlichkeit der Unternehmenssysteme muss eingeplant werden. Ein **internetfähiger Informationsaustausch** wird benötigt. In der EAI ist die Verwendung eines traditionellen synchronen request-response Modells völlig ausreichend (beispielsweise bei RPCs), da man meist auf die Verlässlichkeit des eigenen Netzwerks und der eigenen Anwendungen vertrauen kann, bzw. Fehler innerhalb

kürzester Zeit kommuniziert werden können. Bei der B2Bi kann nicht davon ausgegangen werden, dass der Kommunikationspartner immer erreichbar ist oder überhaupt antwortet. Daher bietet sich eine ereignisgesteuerte Verarbeitung von Informationen und Nachrichten eher an. Dazu müssen Nachrichtenprotokolle entwickelt bzw. ausgewählt werden, die Anforderungen, wie Transaktionsfähigkeit, das Herstellen von Beziehungen zwischen Nachrichten u.ä. erfüllen. SOAP (siehe Kapitel 3.2) ist ein solches Protokoll.

Die **Sicherheitskonzepte** für die B2Bi müssen erweitert werden. Hier bestehen hohe Anforderungen, da es um die Integration verteilter Anwendungen geht. Es existiert kein zentraler Server, auf dem die Sicherheit überwacht werden kann. Die Sicherheitsaspekte Vertraulichkeit, Autorisierung, Authentizität und Integrität müssen auch über das „unsichere" Internet gewährleistet sein.

2.3.4 Neue Anforderungen

Um B2Bi „on the fly" durchzuführen müssen die zu integrierenden Unternehmenssysteme weitere Charakteristiken aufweisen. Linthicum zählt hier vier Anforderungen, die Systeme erfüllen müssen, um „on the fly" mit anderen Unternehmenssystemen interagieren zu können. [vgl. Linthicum1, S.3 ff]

- Als erstes müssen die Unternehmenssysteme den **Bedarf** an beispielsweise einem Produkt **sofort registrieren** können, um nachforschen zu können, ob Potenziale vorhanden sind, diesen Bedarf zu stillen.
- Alle Systeme müssen in jede Richtung kommunizieren können. Das Absenden einer Bestellung muss im empfangenden System in Echtzeit ein Bestellungs-Ereignis und eine sofortige Reaktion – beispielsweise eine Verzögerungsmeldung wegen Lieferschwierigkeiten – auslösen. Dazu sind genaue Informationen notwendig. In einer ereignisgesteuerten Wirtschaft mit Reaktionen in Echtzeit kommt es darauf an, dass dem System **exakte Informationen** über Lagerbestände, Lieferzeiten und ähnliches zur Verfügung stehen. So kann es ohne die Eingabe eines Menschen abzuwarten, auf Anfragen reagieren.
- **Prozessdefinitionen** werden benötigt. Es reicht nicht nur, Informationen hin und her zu schicken, es muss auch definiert sein, in welcher Reihenfolge und mit welchen Regeln, auf welchen Pfaden sowie unter welchen Bedingungen eine B2B-Interaktion zeitlich und räumlich abzulaufen hat. Dazu müssen auch die Geschäftspartner beschreiben, wie sie aufgerufen werden wollen. Dies wird mit einer Process Description Language (PDL) durchgeführt. Für eine PDL eignen sich zum Beispiel semantische Technologien sehr gut, wie in Kapitel 3.3.3 noch erläutert wird.
- Letztendlich müssen sich alle teilnehmenden Systeme stets über den aktuellen **Stand eines ablaufenden Einkaufsprozess** informieren können, um exakt kalkulieren zu können.

2.3.5 Anforderungen an dynamische B2B Integration

Die Anforderungen, die in diesem Kapitel herausgestellt wurden lassen sich in elf Punkten unter fünf Oberbegriffen zusammenfassen:

Unternehmensinterne Anpassungen:

1. Ein Bedarf muss sofort registriert werden können.
2. Exakte Informationen müssen in dem e-Business System jederzeit verfügbar sein.

⇨ Um die Punkte eins und zwei durchzusetzen bedarf es einer durchgängigen Integration auch der internen Systeme des B2Bi beteiligten Unternehmens. Nur wenn Informationen automatisch bereitgestellt werden können, ohne dass ständig menschliche Interaktionen und Abstimmungen nötig sind, ist eine effiziente Integration möglich.

Unternehmensexterne Prozessabläufe:

3. Prozessdefinitionen werden benötigt, die steuern, wie und wann welche Operationen durchgeführt werden müssen.
4. Der Stand eines ablaufenden Einkaufsprozess muss den Geschäftspartnern jederzeit zur Verfügung stehen, um exakt kalkulieren zu können.

⇨ Zu Punkt 3 und 4 sind einige Process Execution Languages (PEL) in der Entwicklung, die ermöglichen Prozessabläufe zu definieren, und die teilnehmenden Partner über deren Stand zu informieren (siehe Kapitel 3.4).

Schnittstellendefinitionen:

5. Es werden einfache Schnittstellendefinitionen benötigt, über die die e-Business-Systeme der Partner anzubinden sind
   ⇨ Web Services bieten sich hier als effiziente Schnittstellenbeschreibung an (siehe Kapitel 3.2).
6. Es werden Prozessbeschreibungen benötigt, die aufzeigen, welchen Prozessablauf ein System erlaubt.
   ⇨ Diese „potenziellen Prozessbeschreibung“ wird mit Hilfe von Process Description Languages durgeführt.

Nachrichtenaustausch:

7. Es wird ein gemeinsames Netzwerk benötigt
   ⇨ Internet
8. Die Architektur des Informationsaustausches muss an die Unverlässlichkeit und Dynamik des Internet angepasst werden
   ⇨ Beispielsweise asynchrone Kommunikation
9. Weiterentwickelte Sicherheitskonzepte werden benötigt
   ⇨ Das Nachrichtenprotokoll SOAP, das von Web Services genutzt wird, liefert

mittlerweile einige Mittel um sichere Übertragungen zu gewährleisten (siehe Kapitel 3.2).

Datenformate:

10. Es wird eine gemeinsame Syntax benötigt
    ⇨ XML

11. B2B Standards müssen unterstützt werden, die auf der gemeinsamen Syntax aufbauen.
    ⇨ In Kapitel 3.1 werden einige offene B2B-Standards und im speziellen openTRANS vorgestellt.

3 BASISTECHNOLOGIEN

Die Aussichten Unternehmen elektronisch miteinander kommunizieren zu lassen sind, wie im letzten Kapitel erläutert, durch die Errungenschaft des Internet und XML stark gestiegen. Weiterhin hat sich herausgestellt, dass Standards für Geschäftsdokumente, die auf XML beruhen, benötigt werden und Technologien derer Austausch. Im Folgenden werden die openTRANS Spezifikation und Web Services vorgestellt, die für diese Standards einstehen.

In Zukunft soll es auch möglich sein, dass die Suche, der Verhandlungsablauf und der B2B Integrationsprozess vollautomatisch abläuft, wozu diese Arbeit eine mögliche Lösung für den Teillaspekt der Prozessintegration liefert.

Die Schnittstellen der Unternehmenssysteme müssen dabei miteinander verglichen werden. Dazu werden Process Description Languages benötigt, die die Bedeutung der Schnittstellen erklären, damit ein Integrationssystem erkennen kann, wie ein Unternehmenssystem in einen Prozess integriert werden kann.

Kapitel 3.3 wird aufzeigen, dass sich semantische Technologien sehr gut eignen, um die Bedeutung von solchen im Internet referenzierbaren Ressourcen zu beschreiben.

Letztendlich muss aus den Prozessbeschreibungen ein gemeinsam ablaufender Prozess entstehen. Die Web Services der Unternehmen müssen komponiert werden. Wie dies möglich ist, wird im Kapitel 3.4 erläutert.

3.1 openTRANS

„Das openTRANS-Format wurde mit dem Ziel entwickelt, einheitliche elektronische Dokumente für den zwischenbetrieblichen E-Commerce bereitzustellen." [opT2, S.7]

openTRANS ist ein B2B-Standard zum Austausch von Geschäftsdaten, wie er in Kapitel 2.3.5 gefordert wird.

Im Folgenden werden zwei solche B2B Standards kurz erläutert und herausgestellt, in wiefern sich openTRANS von diesen unterscheidet.

Darauf hin wird erklärt wo und wie openTRANS in technologischer Hinsicht im Geschäftsprozess eingesetzt werden kann.

3.1.1 offene B2B-Standards für Marktransaktionen

Mittlerweile existieren mehrere konkurrierende B2B Standards die elektronische Dokumente für die zwischenbetrieblichen e-Procurement bereitstellen. Dazu zählen Spezifikationen wie RosettaNet[3] und xCBL[4] und openTRANS[5].

3 http://www.rosettanet.org/
4 http://www.xcbl.org
5 http://www.opentrans.de

In werden die typischen Schritte einer Bestellung aufgezeigt. Diese sind in den verschiedenen Spezifikationen allesamt enthalten. Einige Standards haben die grundlegenden Schritte noch um Statusabfragen erweitert, oder sie unterscheiden sich durch Verallgemeinerung der Schritte bzw. einer differenzierteren Sichtweise.

Beispielhaft sollen hier kurz die Abwicklungsphase bei xCBL und RosettaNet als Transaktionsstandards neben openTRANS vorgestellt werden.

Der Aufbau der Geschäftsdokumente ist von Standard zu Standard unterschiedlich komplex. Dieser soll hier aber nicht eingehender verfolgt werden, da hauptsächlich der Ablauf der Abwicklungsphase für diese Arbeit interessant ist.

xCBL

xCBL beispielsweise umfasst einige Untergruppen, so genannte Namespaces, die sich mit verschiedenen Geschäftsdokument-Bereichen des e-Business befassen. So existieren Namespaces für den Produktdatenaustausch, den Rechnungsverkehr und für Bestellungen der Namespace „OrderManagement". [xCBL1]

```
ChangeOrder
Order
OrderRequest
OrderResponse
OrderConfirmation
OrderConfirmationResponse
OrderStatusRequest
OrderStatusResult
```

Listing 3.1 xCBL – Geschäftsdokumente des OrderManagement

Das Versenden der Rechnung ist nicht mehr im eigentlichen OrderManagement enthalten, die in Listing 3.1 dargestellt werden. Diese Dokumente, unter anderem der Dokumentyp „Invoice", sind in dem Namespace „financial" eingebettet.

RosettaNet

Bei RosettaNet sind die Geschäftsdokumente erst nach Clustern unterteilt, wobei hier der „Cluster 3: Order Management" für diese Arbeit von Bedeutung ist. Darin werden weitere Segmentierungen gemacht. Innerhalb des Segments „Quote and Order Entry" sind die typischen Geschäftsdokumenttypen der Abwicklungsphase enthalten (siehe Listing 3.2).

PIP 3A1: Request Quote
PIP 3A3: Request Shopping Cart Transfer
PIP 3A4: Request Purchase Order
PIP 3A5: Query Order Status
PIP 3A6: Distribute Order Status
PIP 3A7: Notify of Purchase Order Update
PIP 3A8: Request Purchase Order Change
PIP 3A9: Request Purchase Order Cancellation
PIP 3A10: Notify of Quote Acknowledgment
PIP 3A11: Notify of Authorization to Build
PIP 3A12: Notify of Authorization to Ship
PIP 3A13: Notify of Purchase Order Information
PIP 3A14: Distribute Planned Order

Listing 3.2 RossettaNet – Geschäftsdokumente des „Quote and Order Entry"

PIP bedeutet „Partner Process Interface". Ein PIP beschreibt jeweils Request und Response von Einkäufer und Verkäufer samt ausgetauschter Nachrichten für einen Einkaufsschritt.

Die RosettaNet Spezifikation geht bei der Definition der Geschäftsdokumenttypen noch weiter ins Detail als xCBL. So existiert nicht nur ein „ChangeOrder" wie bei xCBL sondern auch ein „Notify of Purchase Order Update", womit der Einkäufer über die Notwendigkeit der Änderung einer Bestellung informiert werden kann.

Hier ist das Geschäftsdokument für Rechnungen wiederum ausgelagert in ein anderes Segment namens „Returns and Finance". Dagegen sind Geschäftsdokumente, die zur Absichtsphase gehören, in diesem Segment untergebracht („Request Quote", „Notify of Quote Acknowledgment")

3.1.2 Das openTRANS Format

Die Version 1.0 des Standards wurde in einem gemeinsamen Arbeitskreis, dem "eBusiness Standardization Committee" (eBSC), in Kooperation mit dem BME[6] erstellt. Die Spezifikation [opT2] wurde von Mitarbeitern des Fraunhofer IAO und Universität Essen BLI verfasst.

openTRANS hat im Gegensatz zu xCBL und RosettaNet nicht den Anspruch alle möglichen e-Business Transaktionsarten abzubilden. Mit openTRANS ist zum Beispiel kein komplexer Rechnungsverkehr möglich. Es existiert nur ein Geschäftsdokument „Invoice". openTRANS ist primär auf die Abwicklungsphase einer unkomplizierten Markttransaktion ausgelegt. Es existieren zudem noch zwei Dokumente – RFQ und QUOTATION – die eigentlich in die Absichtsphase einzuordnen sind.

6 Bundesverband Materialwirtschaft, Einkauf und Logistik e.V., (http://www.bme.de/)

Die Dokumenttypen

openTRANS stellt die folgenden Geschäftsdokumente zur Verfügung:

- DISPATCHNOTIFICATION (Lieferavis)
- INVOICE (Rechnung)
- ORDER (Auftrag)
- ORDERCHANGE (Auftragsänderung)
- ORDERRESPONSE (Auftragsbestätigung)
- QUOTATION (Angebot)
- RECEIPTACKNOWLEDGEMENT (Wareneingangsbestätigung)
- RFQ (Request For Quotation, Angebotsanforderung)

Listing 3.3 openTRANS – Geschäftsdokumenttypen

openTRANS baut auf dem BMEcat[7] Standard auf. BMEcat ist ein Standard zur elektronischen Datenübertragung für Artikelkataloge. Anders als beispielsweise EDIFACT ist BMEcat auf XML aufgebaut und somit auch leichter von Menschen zu verstehen.

Durch Verwendung von Syntax und Semantik der BMEcat Spezifikation in den openTRANS Dokumenttypen sind die beiden Formate hunderprozentig kompatibel. So können BMEcat Produktdatenkataloge, von openTRANS-Geschäftsdokumenten referenziert und deren Inhalte ohne grosse Transformation verwendet werden.

Die in Listing 3.3 vorhandenen Geschäftsdokumenttypen stimmen mit denen nach Listing 2.1 benötigten Dokumenttypen überein, wobei eine Bestellungsannullierung als Spezialfall durch den openTRANS-Typ „ORDERCHANGE“ abgedeckt wird.

Unterschied zu anderen Standards

Hier wird einer der Hauptunterschiede von openTRANS zu RosettaNet und xCBL deutlich, auf deren Analysen openTRANS entwickelt wurde. Es gibt nicht eine Unmenge von Dokumenten für jede mögliche Mitteilung an einen Geschäftspartner, wie zum Beispiel die Aufforderung an einen Einkäufer seine Bestellung zu ändern, sondern nur einige wenige Dokumente.

Das macht die Integration des Standards in ein Unternehmen einfacher, da nicht so viele Dokumenttypen unterstützt werden müssen, um den Verkehr mit Geschäftspartnern zu gestatten. Dadurch wird die Verwendung von openTRANS auch in mittelständigen Unternehmen attraktiv, bei denen in den wenigsten Fällen all die Spezialfälle auftreten, die in umfassenderen B2B Standards definiert sind.

7 http://www.bmecat.org

3.1.3 Prozessintegration mit openTRANS

In der openTRANS-Spezifikation ist des weiteren keinerlei Choreographie für die Dokumente vorgesehen.

> *„..., wobei hier keine Aussage über die Prozessgestaltung gemacht wird, d.h. es bleibt den Geschäftspartnern überlassen, welche der zur Verfügung stehenden Dokumente sie verwenden."* [opT2, S.7]

Da nicht jeder Geschäftspartner alle openTRANS-Dokumenten benötigt, können Unterschiede in der Zahl der unterstützten Geschäftsdokumente auftreten.

Abgesehen von diesen Umsetzungsalternativen der Spezifikation können sich die Geschäftsregeln der Partner unterscheiden. Ein Einkäufer muss damit rechnen, dass er eine „ORDERCHANGE" nur bis zu 20 Stunden vor der Lieferung durchführen darf, o.ä.

Die unterschiedlichen Prozessabläufe, die bei Geschäftspartnern auftreten können, müssen wie in Kapitel 2.3.5 festgehalten detailiert beschrieben werden, um eine Prozessintegration zu ermöglichen. In Kapitel 4 werden die genauen Anforderungen an die Prozessbeschreibungen erarbeitet.

3.2 Web Services

3.2.1 Was sind Web-Services?

Die genaue Definition eines Web Service ist sehr umstritten. Jede Firma die mit Web Services arbeitet sieht die von ihnen entwickelte Technologie als Definitionskriterium an. Das W3 Konsortium nimmt zum Beispiel die WSDL[8] [WS3] mit in die Definition auf. Bei Microsoft, die in die Entwicklung der SOAP and UDDI Protokolle involviert war, sind diese Technologien bei der Definition von Web Services hervorgehoben [MS1]. Die Frage, in wie weit die Technologien in die Definition hinein bezogen werden kann, ist noch offen, allerdings gibt es bereits klare Präferenzen. Fünf Prinzipien lassen sich für Web Services grundsätzlich ausmachen:

1. Web Services sind Softwarekomponenten, die lose miteinander über ein gemeinsames Netzwerk verbunden werden können.
   Dieses Netzwerk ist im allgemeinen das Internet. Sicherlich können Web Services auch in anderen Netzwerken entwickelt oder aufgesetzt werden, aber zum Internet hat prinzipiell jeder Zugang.
2. Web Services beschreiben sich selbst in einer gemeinsamen Sprache. Sie legen fest, wie sie mit welchen Informationen aufgerufen werden können.
   WSDL [WSDL1] ist momentan die sich durchsetzende Wahl zur Selbstbeschreibung der Schnittstellen von Web Services.

[8] WSDL – Web Service Description Language

3. Es muss eine Möglichkeit geben, Web Services über das gemeinsame Netzwerk zu finden.
   UDDI[9] [UDDI1] dient heute hauptsächlich als Mechanismus zum Auffinden von geeigneten Web Services.

4. Jeder Service muss die übermittelten Informationen des anderen verstehen können. Eine gemeinsame Syntax ist nötig, in der die kommunizierenden Systeme Informationen austauschen können.
   XML[10][XML1] wird nicht nur hauptsächlich für den Datenaustausch genutzt, sie liegt auch den anderen Technologien zu Grunde. XML ist aus den Web Services heute nicht mehr wegzudenken. Auch die Weiterentwicklungen der Selbstbeschreibung von Web Services, wie OWL-S[11] oder auch die Web Service Prozess-Ausführungssprachen, wie BPEL4WS[12][BPEL4WS1], sind in XML gehalten.

5. Die Art der Verbindung von Web Services ist immer die gleiche, unabhängig von ihrem zugrundeliegenden System oder der Programmiersprache in der sie implementiert wurden. Hierfür wird ein gemeinsam genutztes Protokoll benötigt, mit dem die Informationen ausgetauscht werden.
   Als Nachrichtenprotokoll wird im allgemeinen SOAP[13] [SOAP1] genutzt.

Im Folgenden werden die Technologien genauer erläutert.

3.2.2 SOAP

Das Simple Object Access Protocol (SOAP) wird in der Spezifikation als ein „einfacher […] Mechanismus für den Austausch strukturierter und typisierter Informationen zwischen Rechnern in einer dezentralen verteilten Umgebung auf Basis von XML" beschrieben [SOAP1].

1998 wurde die Entwicklung von SOAP von Microsoft und anderen Firmen begonnen. Eine erste Version von SOAP hat Microsoft dann im November 1999 über die Internet Engineering Task Force (IETF) veröffentlicht [MS2]. Firmen wie IBM, SAP u.a. schlossen sich daraufhin der Entwicklungsgemeinschaft um Microsoft an. Die SOAP Spezifikation Version 1.1 wurde im Mai 2000 beim World Wide Web Consortium (W3C) eingereicht. Seit Version 1.2 hat SOAP den Status einer „W3C Recommendation" [SOAP1] erreicht, womit die Spezifikation als Standard anzusehen ist.

Struktur

In der SOAP Spezifikation wird eine XML Grammatik definiert, mit der typisierte und strukturierte Informationen zu einer Nachricht zusammengefasst werden können. Bei einer SOAP Nachricht handelt es sich um ein XML-Dokument, dass auch Envelope (Umschlag) genannt wird. Dieses besteht aus einem oder mehreren optionalen Header-Teilen mit anwendungs-spezifischen Steuerungsinformationen. Darin sind

9 UDDI - Universal Description Discovery and Integration
10 XML – eXtended Markup Language
11 OWL-S Web Ontology Language – Services, eine Ontology zur Beschreibung von Web Services
12 BPEL4WS Business Process Execution Language 4 Web Services
13 Simple Object Access Protocl

beispielsweise Angaben zur Transaktionssteuerung oder zum Sitzungsmanagement enthalten. Auf die Header folgt der eigentliche Body-Teil, in dem die zu verarbeitenden Informationen eingeschlossen sind. Siehe dazu auch Abbildung 3.1.

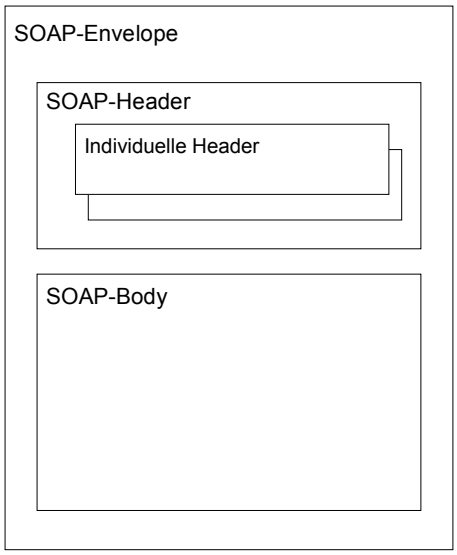

Abbildung 3.1 Aufbau einer SOAP-Nachricht

Übermittlung

Um eine langfristige und flexible Verwendung des SOAP-Protokolls zu garantieren, wurde kein spezielles Transportprotokoll für SOAP vorgeschrieben. Es kann auf verschiedenen Protokollen aufsetzten. Typischerweise wird das Hypertext Transfer Protocol (HTTP) genutzt, aber auch andere Protokolle wie das Simple Mail Transfer Protocol (SMTP) oder Java Message Service (JMS) könnten für SOAP Verwendung finden.

Kommunikationsmodelle

SOAP findet in zweierlei Kommunikationsmodellen Verwendung. Bei der ersten Alternative handelt es sich um nachrichtenorientierte Kommunikation, mit der die Übertragung von beliebigen XML-Dokumenten ermöglicht wird. Darauf aufbauend realisiert die zweite Variante einen entfernten Methodenaufruf (remote procedure call, RPC). Dabei wird für den Aufruf einer Methode durch SOAP ein spezielles Format festgelegt. (siehe Abbildung 3.2) Die Antwort muss ein dementsprechendes Format besitzen, und enthält entweder das Ergebnis oder eine Fehlermeldung.

Es existieren keine Vorschriften wie die Verwendung der beiden Kommunikationsvarianten genau auszusehen hat. Die RPC-Variante ist für einen Entwickler einfacher umzusetzten, da die Nachrichtenformate bestimmten Konventionen folgen müssen. Bei Nutzung der nachrichtenorientierten Variante entsteht ein größerer Aufwand in der Nachrichtenverarbeitung. Allerdings erlaubt sie größere Flexibilität, da der Nachrichteninhalt frei bestimmt werden kann.

Asynchrone Kommunikation

Abbildung 3.2 SOAP - RPC Aufruf

SOAP erlaubt durch nachrichtenorientierte oder RPC-Kommunikation entweder eine Übermittlung von völlig voneinander unabhängigen Nachrichten oder ein synchrones Anfrage-Antwort-Modell.

Bei der B2Bi ist ein RPC-Modell, wie in Kapitel 2.3.3 festgestellt, nicht angemessen. Hier wird asynchrone Kommunikation benötigt.

Mitterweile sind einige Anwendungsszenarien in Arbeit [SOAP2], wo unter anderem beschrieben wird, wie eine Nutzung des Headers mit ID-Elementen und dazu korrespondierende Referenz-Elementen asynchrone Kommunikation erlauben können. WS-Addressing ist eine dazu anwendbare Spezifikation.

> *Web Services Addressing (WS-Addressing) provides transport-neutral mechanisms to address Web services and messages. [...] This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.* [WS4]

Vormals unter WS-Routing bekannt, lassen sich mit dieser Spezifikation mehrere Web Services in einer bestimmten Ablaufreihenfolge in einen Business Prozess einbinden. Dabei kann mit WS-Addressing nicht nur die exakte Route, sondern auch die für einen Knoten bestimmten Bereiche einer SOAP-Nachricht beschrieben werden.

3.2.3 WSDL

Um einen Web Service nutzen zu können, muss ein Client wissen, welche Nachricht er an welche Schnittstelle in welchem Format schicken muss.

Microsoft und IBM entwickelten für diesen Zweck gemeinsam die Web Service Description Language [WSDL1], deren Spezifikation inzwischen vom W3C vorangetrieben wird.

Bei WSDL handelt es sich um eine XML-Grammatik, mit der die Beschreibung der abstrakten Schnittstelle und der technischen Eigenschaften eines Web-Services ermöglicht wird.

Die abstrakte Schnittstelle eines Web Service wird mit den folgenden Elementen definiert:

- Typdefinitionen (`types`) in XML können optional definiert werden, um sie als Nachrichteninhalte zu verwenden
- Nachrichtenformate (`messages`) müssen definiert werden, die der Client zum Service senden oder von diesem empfangen kann. Die Inhalte einer Message werden über `types` festgelegt.
- Operationen (`operation`) werden definiert, die sich aus einer Message und optional einer Antwort-Message zusammensetzen.
- Operationen werden in einem Schnittstellentyp (`portType`) zusammengefasst.

Um den Service tatsächlich erreichbar zu machen werden ihm weitere technische Eigenschaften zugeordnet:

- Über ein Binding Element (`binding`) wird definiert, mit welchem Protokoll die Nachrichtenübermittlung stattfindet. Typischerweise wird die Übertragung durch SOAP-Nachrichten über HTTP genutzt, wie dies auch in dieser Arbeit vorgesehen ist. Es ist aber auch möglich `messages` direkt über HTTP oder SMTP[14] als Anhang (MIME) zu übertragen. Zudem kann hier den einzelnen Operationen der Kommunikationsmodus (nachrichten- bzw. RPC-orientiert) zugewiesen werden
- Ein Kommunikationsendpunkt (`port`), spezifiziert durch eine Adresse (beispielsweise eine HTTP – Adresse) und das verwendete Binding.
- Mehrere Ports werden in einem Service Element (`service`) zusammengefasst.

WSDL Dokumente können zudem aufeinander aufbauen. Durch `import` Elemente wie in Listing 3.4 ist es möglich andere WSDL-Dokumente zu importieren. Die Definitionen von `messages` oder `types` lassen sich so auslagern und von verschiedenen WSDl-Dokumenten nutzen.

```
<import
 namespace="http://openTRANSFlow.openTRANS.fhg.de/messages"
 location="openTRANSMessages.wsdl"/>
```

Listing 3.4 import von WSDL Dokumenten

Als `location` wird eine URI angegeben, die entweder relativ oder absolut den Ort des zu importierenden Dokuments angibt.

[14] Simple Mail Transfer Protocol - Das meistverwendete Protokoll für die Versendung von E-Mails

3.2.4 UDDI

Das Auffinden von Web Services ist für diese Arbeit nicht von Bedeutung, da eine Konzentration auf die Prozessintegration vorgenommen wird. Daher wird nur kurz erwähnt welche Technologie normalerweise dazu genutzt wird.

Um einen Web Services nutzen zu können, muss zuvor seine Zugriffsadresse und seine Schnittstellendefinition bekannt sein, die gewöhnlich im WSDL Format vorliegt. Diese Informationen über Web-Services sollen in Verzeichnissen bereitgestellt werden. Microsoft, IBM, Ariba und andere entwickelten die Spezifikation Universal Description, Discovery and Integration. Die UDDI community, bei der es sich um einen Zusammenschluss vieler Firmen handelt, entwickelt diese Spezifikation weiter [UDDI].

UDDI ist so konzipiert, dass Anwendungen selbstständig Register, die nach der UDDI Spezifikation verfasst wurden, durchforsten können. Hinzukommend werden wegen der Lesbarkeit auch browserbasierte Benutzeroberflächen angeboten.

3.3 Semantische Technologien

Mit dem Konzept des Semantic Web steht das WWW vor den größten Veränderungen seit der Erfindung der Hyperlinks 1989. Die Aufbereitung seiner Inhalte mit semantischen Informationen durch semantische Technologien wird es ermöglichen, dass das WWW in Zukunft nicht ausschließlich von Menschen sondern auch von Maschinen interpretiert werden kann. Computer werden die Daten lesen, einordnen und für den Menschen vorbereiten können.

Die Vision von Web Services umfasst ein voll integriertes Rechnernetzwerk inklusive PCs, Server, Programme, Anwendungen und Netzwerk-Komponenten, die alle zusammenarbeiten. Dieses Netzwerk führt automatisch verteilte Operationen mit den am besten passenden Komponenten durch und liefert die Informationen in einer vom Benutzer gewünschten Form [WS1]. Dazu bedarf es Möglichkeiten von sofortiger Konnektivität und Interoperabilität der einzelnen Komponenten. Diese Art der automatisierten Komposition war in den Anfängen von Web Services noch ungelöst [WS2]. Heute nähert man sich diesem Problem mit der Verknüpfung von Semantic Web und Web Services.

In WSDL besteht keine Möglichkeit zu erkennen, welche Bedeutung die einzelnen Operationen und ihre Parameter besitzen und in welcher Reihenfolge eine Anordnung Sinn macht. Durch die Namensgebung können Menschen meist erschliessen, welche Auswirkungen eine Operation besitzt, welche Parameter sie benötigt, und welche Werte sie zurückgibt. Über diese Schlussfolgerungen hinaus kann ein Mensch auch auf eine sinnvolle Anordnung von Web Services schliessen. Eine Maschine, die eine Integration von verschiedenen Web Services anstrebt, ist dazu nicht ohne weiteres in der Lage. Hier bieten sich semantischen Technologien als Beschreibungssprache für Web Services an, um solche Bedeutungen maschinenverständlich darzustellen.

Ein Ansatz dazu ist OWL-S [OWLS1], das gerade zur semantischen Aufbereitung von Web Services gedacht ist. OWL-S baut auf semantischen Technologien wie RDF und OWL auf, die in Kapitel 3.3.1 und 3.3.2 für das Verständnis von OWL-S kurz vorgestellt werden.

3.3.1 Realisation des Semantic Web

Das Konzept des Semantic Web Framework besteht aus mehreren Ebenen.

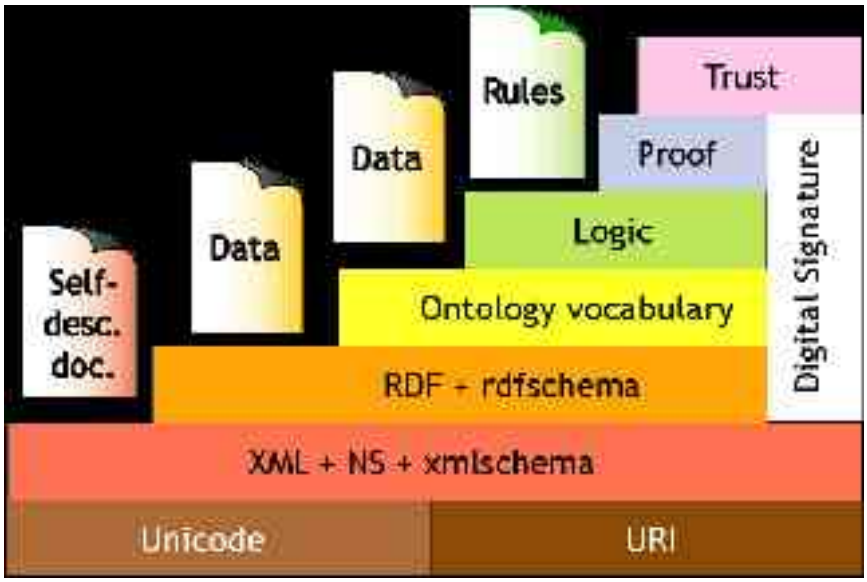

Abbildung 3.3: Architektur des Semantic Web [TBLee1]

Aufgebaut wird auf den schon bekannten und standardisierten Technologien wie HTTP für den Transport von Daten, URIs für naming und location, sowie **XML** [XML1] als allgemein verwendetes Syntaxformat.

Darauf auf setzt das „**RDF + rdfschema**". Das Resource Description Framework [RDF1] stellt eine einfache Möglichkeit zur Beschreibung von Informationen bereit. Es ermöglicht dabei die Nutzung von Ontologien zur Beschreibung von Ressourcen.

So lassen sich eindeutige Aussagen (Subjekt – Prädikat – Objekt) über bestimmte Ressourcen treffen.

Ein Beispiel ist die folgende Aussage, die RDF konform in Abbildung 3.4 dargestellt wird:

„Diese Website (http://www.w3c.org/People/Berners-Lee = Subjekt) wurde erstellt (http://purl.org/dc/elements/1.1/creator = Prädikat) von 'Tim Berners-Lee' (ein Text als Objekt)"

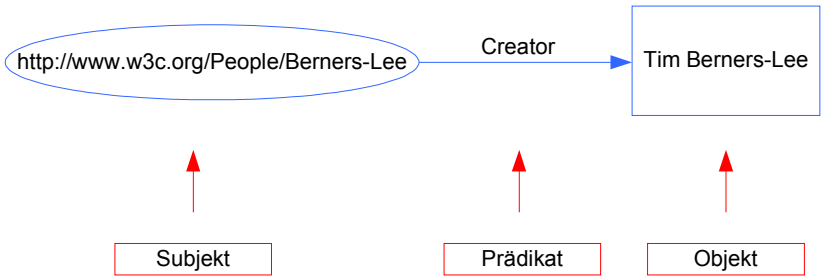

Abbildung 3.4 Beispiel eines RDF Statement

Das Prädikat „Creator“ ist in diesem Fall eine Klassendefinition in einem RDF-Dokument, die über die URI http://purl.org/dc/elements/1.1/creator referenziert wird. Dadurch ist auch die Bedeutung, die das Objekts für das Subjekt besitzt, eindeutig festgelegt. Auf einer solchen Basis ist auch der Austausch von Informationen von Rechner zu Rechner möglich, wenn beide den Zugriff auf die beschreibenden Ressourcen besitzen.

Um RDF-Graphen zwischen verschiedenen Applikationen austauschen zu können, wird eine standardisierte Syntax benötigt. RDF stellt für diesen Zweck die XML-Syntax „RDF/XML“ zur Verfügung. RDF/XML wird in der „RDF/XML Syntax Specification“ [RDF4] definiert.

Die Umsetzung des obigen Statements in RDF/XML ist in Listing 3.5 dargestellt. Subjekt, Prädikat und Objekt der Aussage sind dabei hervorgehoben.

```
1. <?xml version="1.0"?>
2. <rdf:RDF
3.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.   xmlns:dc="http://purl.org/dc/elements/1.1/" >
5.   <rdf:Description
6.     about=" http://www.w3c.org/People/Berners-Lee ">
7.     <dc:Creator>Tim Berners-Lee</dc:Creator>
8.   </rdf:Description>
9. </rdf:RDF>
```

Listing 3.5 Beispiel eines RDF-Statements in RDF/XML

Oberhalb von RDF liegt das „**Ontology Vocabulary**". Als Ontology kann auch rdfschema (RDFS) eingesetzt werden, welches sich selbst als „lightweight ontology"[RDFS1]versteht. RDFS ermöglicht die Erstellung von weiteren Ontologien.

RDFS ist eine semantische Erweiterung von RDF und erlaubt es einfache Hierarchien von Konzepten – durch die in RDFS definierte RDF-Resource Class – und Eigenschaften – durch die in RDFS definierte RDF-Resource Property – zu bilden, ähnlich einem Klassendiagramm in einer objektorientierten Programmiersprache wie Java. Im Gegesatz zum Klassendiagramm in einer Programmiersprache erlaubt RDFS einen Referenzierung der in ihr definierten Konzepte über das Internet.

Ontologien wie RDFS sind der Kern der Semantik. Sie spezifizieren Konzepte von Bereichen der Umwelt, um deren Begriffe maschinenlesbar und austauschbar zu machen.

Andere Ontology Languages, z.B. OWL [OWL1], die in Kapitel 3.3.2 kurz erläutert werden, gehen noch einen Schritt weiter als RDFS und beinhalten auch Möglichkeiten zur logischen Verknüpfung der in Ontologien beschriebenen Begriffe. Diese Möglichkeiten beschreibt die nächste Ebene der „**Logic**". Die Logikebene erlaubt z.B. Begriffe als Synonyme oder Antonyme zu beschreiben.

„Proof", „Trust" und „Digital Signature" sollen in Zukunft darüber entscheiden, wie verlässlich eine Quelle semantisch angereicherter Informationen ist. Anstatt nur eine Antwort zu liefern, könnte eine Semantic Web Applikation einen Beweis - einen nachprüfbaren „**Proof**" - mitliefern, wie sie zu diesem Fakt gekommen ist. Anhand von Bewertungen könnten in Zukunft Informationsquellen nach ihrer Verlässlichkeit bewertet werden um eine Vertrauensbasis, eine „**Trust**"-Basis, für den Austausch von Informationen zu schaffen. Eine „**Digitale Signature**" erlaubt es festzustellen, dass die gelieferten Informationen wirklich von einem vertrauenswürdigen Semantic Web Teilnehmer stammen.

Diese drei Ebenen „Logic", „Proof" und „Trust" des Semantic Web Frameworks sind noch Forschungsgebiet. Ansätze von „Logic" sind in einigen Ontologie Sprachen schon enthalten, wie zum Beispiel OWL [OWL1].

3.3.2 Web Ontology Language (OWL)

Bis vor kurzem wurde die Entwicklung von DAML-OIL[15] von einem Komitee aus Mitgliedern der Design Teams der amerikanischen DAML[16] und der europäischen OIL[17] durchgeführt. Im Dezember 2001 wurde DAML-OIL beim W3C eingereicht[18], und bildet den Grundstock für die heute in der Version 1.0 vom 10. Februar 2004 vorliegende Web Ontology Language (OWL).

Der Begriff Ontology kommt ursprünglich aus der Philosophie und wurde mit der Zeit in verschiedenen Bereichen der IT leicht verzerrt eingesetzt. Eine Definition von Dieter Fensel und Chris Bussler [WSMF1] versteht Ontologien für das Semantic Web als formale und in Konsens geschlossene Spezifikationen von Konzepten, die eine gemeinsam genutztes allgemeines Verständnis über einen bestimmten Definitionsbereich liefern, das zwischen Personen und Anwendungen ausgetauscht werden kann.

Die Web Ontology Language [OWL1] nutzt RDFS um Ressourcen zu definieren, mit denen sich logische Relationen zwischen Klassen (Classes) und Eigenschaften (Properties) ausdrücken lassen. So können Eigenschaften von Klassen als symmetrische, transitive und funktionale Eigenschaften deklariert werden.

Als Beispiel für eine symmetrische OWL-Eigenschaft sei `istFreundVon` definiert. Diese sei der OWL-Klasse `Mensch` zugewiesen. Da `istFreundVon` eine symmetrische Eigenschaft ist, muss der Wert den die Eigenschaft für einen

15 http://www.daml.org/

16 DAML steht für "DARPA Agent Markup Language", deren Ziel es ist die eine Sprache und Werkzeuge bereitzustellen um die Konzepte des Semantic Web zu fördern. DARPA ist die "Defense Advanced Research Projects Agency" des US-amerikanischen Department of Defense.

17 Ontology Inference Layer - wurde entwickelt vom IST OntoKnowledge Projekt – http://www.ontoknowledge.org/oil/

18 http://www.w3.org/Submission/2001/12/

Menschen (eine Instanz der Klasse `Mensch`) annimmt, ebenfalls vom Typ `Mensch` sein.

Mit Hilfe dieser und anderer logischer Definitionen in OWL können durch Anwendungen, die semantische Technologien nutzen, Schlussfolgerungen aus Ontologien gezogen werden.

3.3.3 OWL-S

Die OWL-S[19] Ontologie baut auf OWL auf. Sie ermöglicht die Beschreibung eines Web Service hinsichtlich seiner Nutzungsmöglichkeiten und Integrationsmöglichkeiten mit anderen Web Services. Es beruht auf einem Klassensystem, das es erlaubt einen Prozess sowohl in Hinsicht eines Kontroll- und Datenflusses, als auch in Bezug zu den Voraussetzungen einer Ausführung und deren Auswirkungen in der realen Welt zu modellieren.

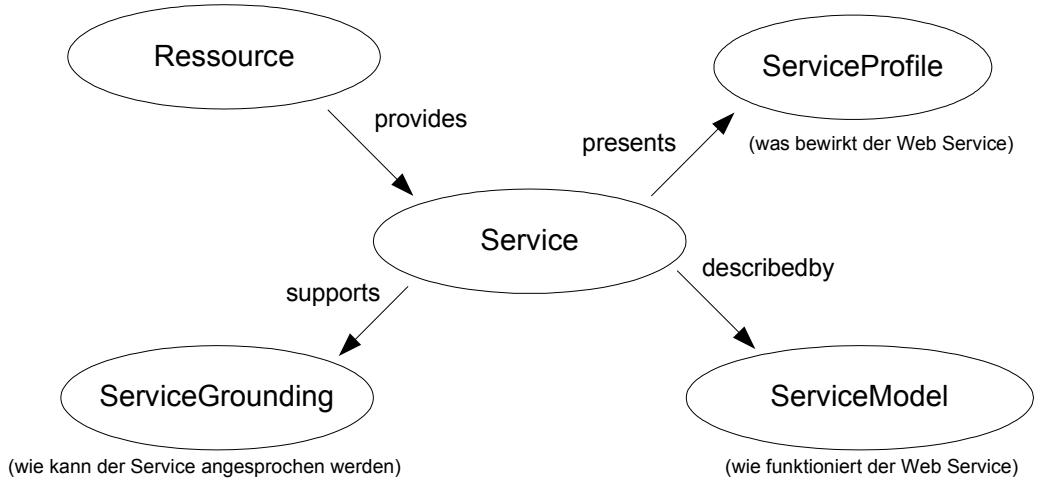

Abbildung 3.5 oberste Ebene der OWL-S Ontologie

Das OWL-S Klassensystem gliedert sich in

- Service Profil
  Diese Komponente beschreibt, wer den Dienst bereitstellt, welche Funktion er erfüllt, und eine Anzahl von Charakteristika. [OWLS1]. Sie dient der Werbung und der Suche nach einem passenden Web Service .
- Service Grounding
  definiert detailliert, wie eine Software den Web Service nutzen kann. Das Service Grounding ordnet die abstrakte Spezifikation eines Prozesses den realen Operationen und Ein- und Ausgabeschnittstellen des Web Service zu. Dabei nutzt das Service Grounding die Selbstbeschreibungen des Web Service durch seine WSDL-Spezifikation.
- Service Model
  Das Process Model ist eine spezielle Unterklasse des Service Model, die die Pro-

[19] http://www.daml.org/services/owl-s/1.0/

zessbeschreibung stellt. Diese wiederum besitzt eine Process Ontology und eine Process Control Ontology.
Die Process Ontology wird im Folgenden genauer beleuchtet.
Die Process Control Ontology soll in Zukunft die Überwachung eines laufenden Prozesses möglich machen. Sie ist allerdings in der vorliegenden Version 1.0 noch nicht exakt ausgearbeitet.

Die Kardinalitäten von Service zu seinen drei Komponenten sind dabei unterschiedlich. So kann ein Service keine oder n Service Profiles besitzen, die unterschiedliche Anwendungsmöglichkeiten des Service beschreiben. Hier legt OWL-S den Nutzern keine Beschränkungen auf. Zwei Einschränkungen gibt es: 0 oder 1 Service Model legen fest, wie ein Service arbeitet. Besitzt ein Service ein Model, müssen dessen Schnittstellen auch durch ein Grounding beschrieben werden.

Process Ontologie

Die Process Ontology – kurz Process.owl – genießt hier besondere Beachtung, da sie für die Prozessintegration am wichtigsten ist.

In der Process.owl sind Begriffe für die Strukturierung eines Prozesses und ihre Beziehungen zueinander enthalten, die für eine Prozessbeschreibung instanziiert werden können.

Ein oben erwähnte Process Modell enthält eine `Process` –Instanz.

`Process` ist Subklasse von `IntervalEvent` aus der `http://www.isi.edu/~pan/damltime/time-entry.owl` Ontologie.

`Process` ist die abstrakte Superklasse für `SimpleProcess`, `CompositeProcess` und `AtomicProcess`, wie in Abbildung 3.6 dargestellt.

In der RDF/XML Variante von OWLS wird das folgendermaßen ausgedrückt:

```
1. <owl:Class rdf:ID="Process">
2.   <rdfs:comment>
3.    The most general class of processes
4. </rdfs:comment>
5.   <rdfs:subClassOf rdf:resource="&time;#IntervalEvent"/>
6.   <owl:unionOf rdf:parseType="Collection">
7.     <owl:Class rdf:about="#AtomicProcess"/>
8.     <owl:Class rdf:about="#SimpleProcess"/>
9.     <owl:Class rdf:about="#CompositeProcess"/>
10.  </owl:unionOf>
11.</owl:Class>
```

Listing 3.6 Die OWL-S Process-Klasse

Ein `SimpleProcess` ist eine vereinfachte Sichtweise einer `Process` Instanz. Er kann entweder `realizedBy` einem `AtomicProcess` sein, oder `expandsTo CompositeProcess`.

Ein `AtomicProcess` entspricht dem direkten Aufruf einer Web Service Operation, die ihm in der Service Grounding zugeordnet wird.

Bei `CompositeProcesses` handelt es sich um zusammengesetzte `AtomicProcesses` und/oder `CompositeProcesses`.. Diese werden innerhalb eines `CompositeProcesses` mit Hilfe von sog. `ControlConstructs` strukturiert, wie zum Beispiel:

- `Sequence`: dient der Festlegung der Reihenfolge von `Processes`
- `Repeat-While`: `Processes` lassen sich mehrmals ausführen, solange eine bestimmte Bedingung erfüllt ist.
- `Unordered`: `Processes` in einem `Unordered ControlConstruct` können in beliebiger Reihenfolge ausgeführt werden.
- `Choice`: Von in `Choice` enthaltenen `Processes` kann einer für die weitere Ausführung ausgesucht werden.

Einem Process können `Inputs` und `Outputs` sowie `Preconditions` und `Effects` zugeordnet werden.

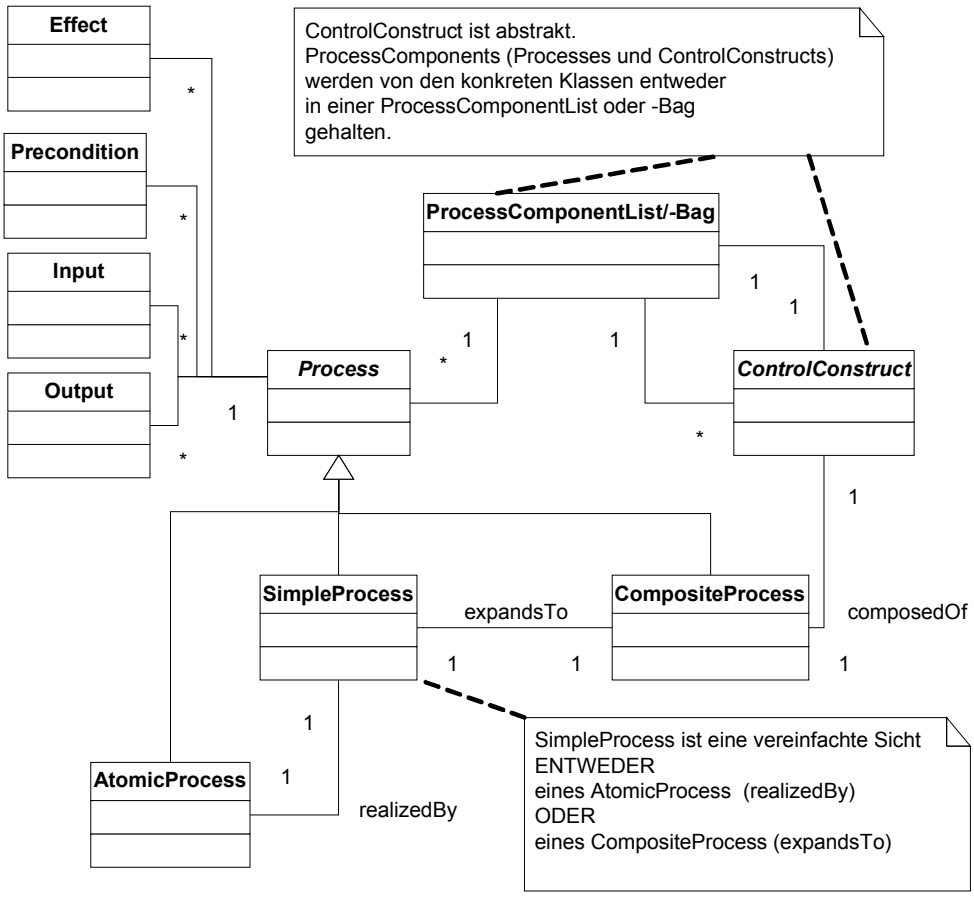

Abbildung 3.6 Process.owl - Klassendiagramm

`Inputs` und `Outputs` werden ebenfalls über das Service Grounding auf bestimmte Nachrichtentypen in einem WSDL-Dokument gemappt.

Eine `Precondition` ist zum Beispiel das ein Buch auf Lager sein muss, damit es bestellt werden kann. Ein `Effect` wäre, dass ein Buch gekauft worden ist.

Die Ontologie ist noch nicht vollständig entwickelt. Die `Condition` Klasse, die genutzt wird um Bedingungen für Schleifen oder andere `ControlConstructs`, sowie um eine `Precondition` zu bestimmen, besteht bisher nur als Platzhalter, ebenso wie die Klasse `Effect`.

Die Deklaration von Timeouts ist allerdings mit Hilfe der in der Process.owl deklarierten Property `timeout` implizit möglich, da die Process.owl sich mit ihren Hauptklassen (`Process` und `ControlConstruct`) auf eine Ontologie der Zeit bezieht.

Die Zusammenarbeit von verschiedenen Parteien lässt sich in OWL-S nicht explizit ausdrücken. Allerdings besteht die Möglichkeit in einem ServiceGrounding für verschiedene Operationen verschiedene WSDLs zu referenzieren.

Die Entwicklung von OWL-S ist zwar noch nicht abgeschlossen, und vorgesehene Teilaspekte sind noch nicht umgesetzt, aber für die hier umzusetzende Anwendung ist OWL-S genügend weit entwickelt.

3.4 Komposition von Web Services

Komposition meint das gleiche wie Orchestrierung, Choreographie und Collaboration. In der Literatur werden alle diese Begriffe genutzt, um die Zusammenarbeit von Web Services in einer gemeinsamen Prozessbeschreibung zu realisieren.

Sie bieten grundlegende Aktivitäten, wie das Empfangen und Versenden von Nachrichten von und zu Web Services, sowie das Kopieren von Nachrichteninhalten von einer empfangen Nachricht in eine andere.

Zudem werden Konstrukte angeboten, mit denen diese Aktivitäten strukturiert werden können. So können sie nacheinander oder parallel ausgeführt werden. Sie können in Schleifen ausgeführt werden, oder erst ausgeführt werden, wenn eine bestimmte Bedingung wahr wird.

Diese Prozessbeschreibung können auf einer Engine installiert und als einzelne Instanzen gestartet werden. Daraufhin läuft eine solche Instanz vollautomatisch ab.

In dieser Arbeit werden solche Kompositions-Technologien benötigt, um die komponierte Prozessbeschreibung zweier Geschäftspartner als ausführbaren Prozess ablaufen zu lassen.

Im Folgenden werden zwei Technologien vorgestellt, die die Komposition von Web Services erlauben: ebBPSS und BPEL4WS.

3.4.1 ebBPSS

Das ebXML Business Process Specification Schema (ebBPSS) ist Teil der ebXML[20] Spezifikation [ebXML1]. ebXML ist ein globaler Standard für elektronische Geschäftsverkehr der unter der Federführung von UN/CEFACT[21] und OASIS[22] entwickelt wird. ebXML kann als Framework für globalen elektronischen Geschäftsverkehr verstanden werden, welches es Geschäftspartnern möglich macht, sich gegenseitig zu finden und ihren Geschäftsverkehr mit Hilfe wohldefinierter XML-Nachrichten aufeinander abzustimmen.

Das Grundkonzept von ebXML beruht auf einem öffentlichen Repository, in dem Industriestandards und Szenarios für Geschäftsprozesse enthalten sind, die für die Beschreibung der meisten Geschäftsprozesse geeignet sind. Unternehmen können diese Prozesse und Szenarios für ihre eigenen Zwecke erweitern [ebXML2].

Die wesentliche Teile des ebXML-Standards sind:

- BPSS – das Business Process Specification Schema: ein XML-Dokument, das den Business Process eines Geschäftspartners beschreibt. BPSS wird auch ebBPSS genannt, um die Zugehörigkeit zu ebXML auszudrücken.
- CC – die Core Components: ein Verzeichnis von Datenstrukturen, die in die eigene Datenstruktur-Definition aufgenommen werden können (z.B. Kunde, Adresse).
- CPP – das Collaboration Protocol Profile: es erlaubt einem Geschäftspartner seine Dienste, d. h. sein Angebot und seine Schnittstellen, zu beschreiben. Dieses Profile wird dann in einem ebXML Repository gespeichert.
- CPA – das Collaboration Protokoll Agreement: Nachdem sich Geschäftspartner über das ebXML Repository gefunden und ihre CPPs verglichen haben, definieren sie ein konkretes CPA, welches ihre Geschäftsbeziehung festlegt.

Eine Business Process Specification (BPS) auf Grund des ebBPSS baut normalerweise auf einer CPA auf, und kann auch in einer solchen referenziert werden, was beides nicht verpflichtend ist. ebBPSS wurde nicht direkt für die Verwendung in Web Services festgelegt, wie zum Beispiel BPEL4WS.

ebBPSS geht einen anderen Weg und legt nur die Rollen fest, zwischen denen ein Austausch von Nachrichten stattfinden soll. Um sie zu einem Excecutable Business Process (siehe oben) weiterzuentwickeln, muss eine solche Spezifikation weiter an eine bestimmte „API" wie Web Services angepasst werden. Es legt sich somit nicht auf ein zugrundeliegende Komponentenmodell fest, von denen Web Services die neuste Generation darstellt.

ebBPSS besteht aus

20 Electronic Business using eXtensible Markup Language
21 United Nations Center For Trade Facilitation And Electronic Business
22 Organization for the Advancement of Structural Information Standards

- einem UML-Klassendiagramm
  Einem Satz von Statements und Relationships, die für das Erzeugen einer ebXML konforme BPS benötigt werden.
- einer XML Version des BPSS
  abgeleitet von der UML Version. Sie beschreibt die BPS-Umsetzung im Detail.
- Production Rules für das Mapping von UML nach XML.
- Business Signal Definitions
  „Application Level Documents" die den momentanen Stand einer „Business Transaction"(siehe unten) 'signalisieren'.

Eine BPS, die einen Prozessablauf beschreibt, wird auch `BinaryCollaboration` genannt. Eine `Collaboration` besteht aus einem Satz von `BusinessTransactions` zwischen Geschäftspartnern, die in Binary- oder Multiparty Collaborations gesammelt sind. `BinaryCollaborations` bestehen zwischen exakt zwei Rollen, `MultipartyCollaborations` zwischen drei oder mehr Rollen. `MultipartyCollaborations` werden aus `BinaryCollaboration` aufgebaut. Rekursiver Aufbau von Collaborations wird durch verschachtelte Wiederverwendung von Binary-Collaborations realisiert.

Eine `BusinessTransaction` ist eine atomare Arbeitseinheit in einem Handelsprozess zwischen zwei Geschäftspartnern. Die Geschäftspartner nehmen dabei gegensätzliche Rollen ein: Requester und Responser.

Eine `BusinessTransaction` ist ein sehr spezielles und sehr einschränkendes Protokol um präzise und durchsetzbare Transaktionssemantiken zu erhalten, die von der bearbeitenden Software erfüllt werden müssen. `BusinessTransactions` werden immer entweder erfolgreich beendet oder schlagen fehl. Bei einem Fehlschlag, müssen alle im bisherigen Ablauf der `BusinessTransaction` getroffenen Änderungen zurückgezogen werden (Rollback).

Realisiert wird eine `BusinessTransaction` durch die Referenzierung eines „Document Flows". Dieser beschreibt die übergebenen Dokumente für Request und Response einer `BusinessTransaction`. Ein Document Flow kann one-way oder two-way sein, jenachdem, ob auf einen Request kein oder ein Response erfolgt.

Listing 3.7 zeigt das Beispiel eines „Document Flow" in ebBPSS durch die Verwendung von `BusinessDocument`-Elementen in einer `BusinessTransaction`.. Die `BusinessTransaction` „Create Order" enthält eine `RequestingBusinessActivity` und eine `RespondingBusinessActivity` die zwei Dokumenttypen zurückgibt.

Diese `BusinessTransaction` wird als `BusinessTransactionActivity` in der `BinaryCollaboration` verwendet. Die referenzierenden und referenzierten IDs sind fettgedruckt.

```
1. <BusinessDocument name=" PO Acknowledgement "
2. specificationLocation="someplace"/>
3. ...
4. <BusinessTransaction name="Create Order">
```

```
5.  <RequestingBusinessActivity name=""
6.    <DocumentEnvelope isPositiveResponse="true"
7.      BusinessDocument="ebXML1.0/PO Acknowledgement">
8.        <Attachment
9.        name=”DeliveryNotes”
10.        mimeType=”XML”
11.        BusinessDocument=
12.        "ebXML1.0/Delivery Instructions"
13.        specification=””
14.        isConfidential=”true”
15.        isTamperProof=”true”
16.        isAuthenticated=”true”>
17.        </Attachment>
18.    </DocumentEnvelope>
19.  </RequestingBusinessActivity>
20.  <RespondingBusinessActivity name=""
21.    <DocumentEnvelope
22.      BusinessDocument="ebXML1.0/PO
23.      Acknowledgement"/>
24.    </DocumentEnvelope>
25.    <DocumentEnvelope isPositiveResponse="false"
26.      sinessDocument=" ebXML1.0/PO Rejection"/>
27.    </DocumentEnvelope>
28.  </RespondingBusinessActivity>
29.</BusinessTransaction>
30....
31.<BinaryCollaboration name="Product Fulfillment"
32.    timeToPerform="P5D">
33.  <Documentation>
34.    timeToPerform =
35.    Period: 5 days from start of transaction
36.  </Documentation>
37.  <InitiatingRole name="buyer"/>
38.  <RespondingRole name="seller"/>
39.  <BusinessTransactionActivity
40.    name="Create Order"
41.    businessTransaction="Create Order"
42.    fromAuthorizedRole="buyer"
43.    toAuthorizedRole="seller"
44.    isLegallyBinding=”true” />
45.  <BusinessTransactionActivity
46.    name="Notify shipment"
47.    businessTransaction="Notify of advance
48.    shipment"
49.    fromAuthorizedRole="buyer"
50.    toAuthorizedRole="seller"/>
51.  <Start toBusinessState="Create Order"/>
52.  <Transition
53.    fromBusinessState="Create Order"
54.    toBusinessState="Notify shipment"/>
55.  <Success fromBusinessState="Notify shipment"
56.    conditionGuard="Success"/>
57.  <Failure fromBusinessState="Notify shipment"
58.    conditionGuard="BusinessFailure"/>
59.</BinaryCollaboration>
```

Listing 3.7 Beispiel einer ebBPSS Prozessbeschreibung[ebBPSS1]

`BusinessStates` geben an, in welchem Zustand sich eine `BinaryCollaboration` befindet. Subtypen von `BusinessStates` sind zum Beispiel `Start`, `Success` oder `Failure`, wie sie in Listing 3.7 in Zeile 51,55 und 57 verwendet werden.

`Transistions` (Zeile 52) beschreiben die Ablauffolge und die Übergänge von einem `BusinessState` zum nächsten oder von `BinaryCollaborations` innerhalb einer `MultipartyCollaboration`.

Jede BPS besitzt ein Äquivalent als UML Activity Diagramm. Es gibt keine spezielle Möglichkeit um „Data Flows" zwischen Transaktionen zu beschreiben. Nur der Kontrollfluß eines Prozesses kann spezifiziert werden. Eine Ausführung von Aktivitäten in Schleifen werden dabei nicht direkt unterstützt.

Andererseits besitzt ebBPSS einige andere Charakteristika, die es zu einem einfachen, aber effizienten Schema für langlebige Prozessbeschreibungen macht. Der Transaktionsteil von ebBPSS beruht auf dem bewährten Modell von RosettaNet [Riordan1]. Unterstützt werden vor allem die Definition von Quality-of-Service Bestimmungen für Transaktionen, wie Authentisierung, Eingangsbestätigungen, Non-Repudiation und Timeouts.

Dazu zählt zum Beispiel die Möglichkeit Nachrichten zweimal bestätigen zu lassen. Zuerst wird eine Eingangsbestätigung verschickt, und darauf folgend eine Akzeptanzbestätigung. Eine solche zeigt dem Geschäftpartner, dass der Inhalt der gesendeten Nachricht für den Geschäftsprozess akzeptabel ist und weiterverarbeitet werden kann.

In ebBPSS lassen sich auch bestimmte Sicherheitsrichtlinien festlegen. Es lässt sich festlegen, dass Nachrichten vor Versenden gesichert werden müssen. Zusätzlich kann bestimmt werden, dass digital signierte Eingangsbestätigungen zurückgeschickt werden müssen, die vom Sender gespeichert werden.

3.4.2 BPEL4WS

BPEL4WS (Business Process Execution Language For Web Services) [BPEL4WS1] ist speziell für die Komposition von Web Services entwickelt worden. Sie ist ein Gemeinschaftsprodukt von IBM, Microsoft und anderen Unternehmen und wurde im August 2002 in einer ersten Version veröffentlicht. Im Mai 2003 wurde Version 1.1 bekannt gegeben. In ihre Spezifikation flossen die Erfahrungen der Prozesssprachen XLANG [XLANG1] und WSFL [WSFL1] ein, die zukünftig durch BPEL4WS ersetzt werden sollen.

Der Prozess präsentiert sich nach außen als Web Service mit einer dazugehörigen WSDL-Schnittstellendefinition. Die Wolke in Abbildung 3.7 steht für den BPEL4WS Business Prozess. Die von dem Prozess Web Service angebotenen `portTypes`, werden in dem Prozess implementiert.

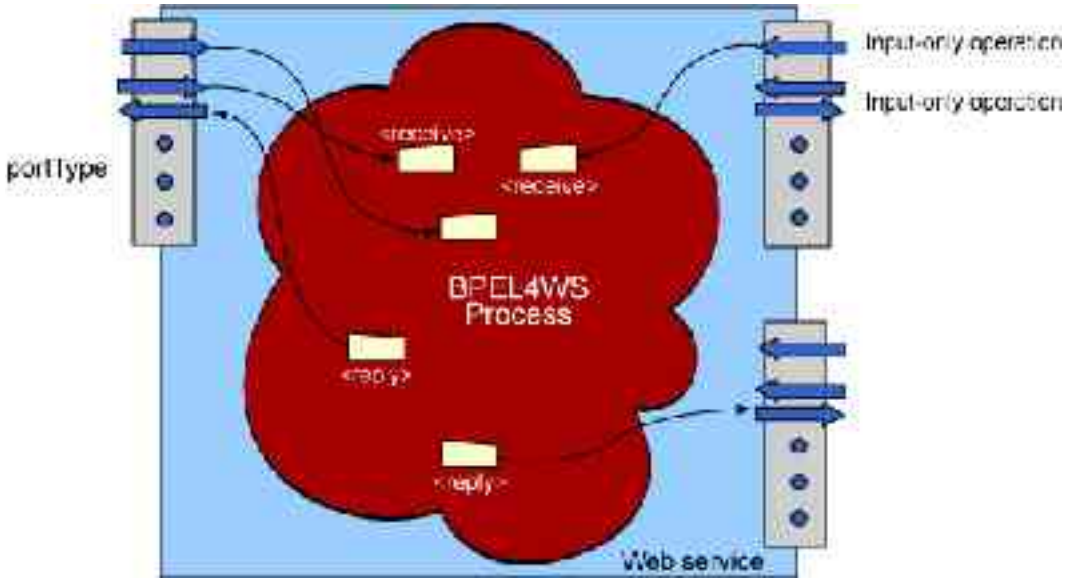

Abbildung 3.7 BPEL4WS Prozess als Web service [BPEL4WS1]

Activities

Der BPEL4WS Prozess an sich ist ein flow-chart-artiger Ausdruck eines Algorithmus. Der einzelne Schritt innerhalb eines Prozesses wird Activity genannt. Man unterscheidet Primitve- und Structure-Activities.

Primitve Activities erlauben:

- den Aufruf von Operationen fremde Web Services (`invoke`),
- das Warten auf Nachrichten von externen Partner (`receive`),
- ein synchrones Response auf ein Request zu erzeugen (`reply`),
- einen spezifizierten Zeitraum abzuwarten (`wait`),
- Daten auf Variablen zu kopieren (`assign`),
- Fehler zu werfen (`throw`),
- den gesamten Prozess und damit die Instanz des Service explizit zu beenden (`terminate`)
- oder einfach gar nichts zu tun (`empty`).

Structure Activities erlauben die Strukturierung von Primitive Activities zu Algorithmen. So ist es unter anderem möglich Activities in geordneter Reihenfolge festzulegen (`sequence`), Schleifen zu definieren (`while`), alternative Wege in einem Prozess zu beschreiten (`pick`) und zu definieren, welche Schritte parallel ausgeführt werden sollten (`flow`). Zwischen Activities, die in `flows` strukturiert sind, können über `links` Abhängigkeiten definiert werden.

Das `pick` Konstrukt bedarf noch einer genaueren Erläuterung. In einem pick sind `onMessage` und ein `onAlarm` Konstrukte erlaubt. Bei einem `onMessage` wird ähnlich einem `receive` auf den Eingang einer Nachricht gewartet. Es ist sogar

gleichbedeutend mit `receive`, wenn nur ein `onMessage` in einem `pick` definiert ist. Ab zwei `onMessage`, kann die eine oder die andere Nachricht empfangen werden, um im Prozess fortzuschreiten. `onAlarm` bietet die Möglichkeit nach einem bestimmten Zeitraum Activities auszuführen, wenn auf ein `onMessage` zulange gewartet wurde.

Partner

Web Services, die mit einem Prozess kommunizieren wollen – ein Partner Web Service müssen einen `partnerLinkType` in ihrem WSDL Dokument definieren:

```
1. <partnerLinkType name="BuyerType">
2.     <role name="BuyerProvider">
3.       <portType name="tns:BuyerPortType"/>
4.     </role>
5.     <role name="BuyerRequester">
6.       <portType
7.        name="tns:BuyerCallbackPortType"/>
8.     </role>
9. </partnerLinkType>
```

Listing 3.8 Beispiel eines `partnerLinkType`

In Listing 3.8 ist die Deklaration eines `partnerLinkTypes` dargestellt. Der `partnerLinkType` mit dem Namen „BuyerType“ deklariert die Rollen „BuyerProvider“ und „BuyerRequester“. In einer BPEL4WS Prozessbeschreibung kann nun der `partnerLinkType` referenziert werden.

```
1.  <partnerLink name="Buyer"
2.         partnerLinkType="buyerflow:Buyer"
3.         partnerRole="BuyerProvider"
4.         myRole="BuyerRequester"/>
```

Listing 3.9 Beispiel eines `partnerLink`

In Listing 3.9 wird ein `partnerLink` mit dem Namen „Buyer“ deklariert, der den `partnerLinkType` „BuyerType“ aus Listing 3.8 referenziert. Dem Prozess wird nun die Rolle „BuyerRequester“ zugewiesen und dem Partner Web Service die Rolle „BuyerProvider“.

In einem `partnerLinkType` wird für jede Rolle ein WSDL `portType` zugewiesen, wie in Listing 3.8 zu sehen. So wird angegeben, welchen Schnittstellentyp der Partner Web Service anbietet und welchen er von seinem Gegenüber erwartet.

Im obigen Beispiel bedeutet dies, dass der Web Service mit der `partnerLinkType`-Deklaration den portType „BuyerCallbackPortType“ erwartet und er selber den portType „BuyerPortType“ anbietet.

Über die definierten `partnerLinks` können nun Primitive Activities ausgeführt werden.

```
1. <receive name="setRFQ"
2.      partnerLink="Buyer"
3.      portType="buyerflow:BuyerCallbackPortType"
4.      operation="setRFQ"
5.      variable="rfqReq"/>
```

Listing 3.10 Beispiel eines `receive`

In Listing 3.10 wird auf den Aufruf der Operation setRFQ die im `portType` „Buyer-CallbackPortType" durch den Partner Web Service mit der Rolle „Buyer" gewartet.

Variablen

`variables` werden verwendet, um die XML-Daten, die durch Primitive Activities wie `receive` oder `invoke` empfangen oder gesendet werden zu speichern. Eine `variable` besitzt immer einen `name` und einen `messageType`. Der `messageType` ist der Type einer WSDL-message, die normalerweise von einem der WSDL der Partners oder vom Prozess selber stammt.

Mit `assign` können die Werte von Variablen kopiert oder deklariert werden.

```
1.  <assign>
2.       <copy>
3.         <from variable="input" part="payload"
4.            query="/urlforbuyer"/>
5.         <to variable="initiateProcessReq" part="payload"/>
6.       </copy>
7.     </assign>
```

Listing 3.11 Beispiel eines `assign`

Über das `query` Attribute wie in Listing 3.11, Zeile 4 zu sehen, können einzelne XML-Elemente abgefragt werden.

Zudem bietet BPEL4WS die Möglichkeit über XPATH[23] Erweiterungen die Werte von Variablen abzufragen. Dazu wird die Funktion

```
bpws:getVariableProperty ('variableName', 'propertyName')
```

angeboten. So können beispielsweise die Bedingungen von `while` Schleifen abgefragt werden:

```
<while condition=
"bpws:getVariableData('variableOne', 'payload') > 0 ">
```

Zwei weitere Bereiche sind die Fehlerbehandlung mit `faulthandlers`, und die Kompensation von nicht erwarteten Ergebnissen (`compensationhandlers`).

`CorrelationSets` erlauben die Zuordnung von Nachrichten zur laufenden Instanz einer Prozessbeschreibung. Dazu werden die einzelne Felder der XML-Dokumente eingehender Nachrichten als Bestandteile eines `CorrelationSets` definiert.

[23]XPATH unterstützt die Navigation und Abrage von XML Dokumenten
http://www.w3.org/TR/1999/REC-xpath-19991116

Stimmen in einem laufenden Prozess die Werte dieser Felder von verschiedenen Nachrichten überein, werden sie als Bestandteile einer Prozessinstanz erkannt.

Im Haupt-Prozess-Bereich wird festgelegt, welche Teile des Prozesses parallel (`flow`) oder hintereinander (`sequence`) in bedingter Abhängigkeit auszuführen sind. Dieser wird mit einem `sequence` Element eingeleitet. Darauf folgen wahlweise weitere `sequences`, oder `flows`.

Das folgende Beispiel [aus Yendluri1] zeigt eine typische BPEL4WS Prozessbeschreibung:

```
<process name="purchaseOrderProcess"
         targetNamespace="http://acme.com/ws-bp/purchase"
          xmlns=
          "http://schemas.xmlsoap.org/ws
           2002/07/business-process/"
         xmlns:lns="http://manufacturing.org/wsdl/purchase">
   <partnerLinks>
     <partnerLink name="customer"
              partnerLinkType="lns:purchaseLT"
              myRole="purchaseService"/>
     <partnerLink name="invoiceProvider"
              partnerLinkType="lns:invoiceLT"
              myRole="invoiceRequester"/>
     ...
   </partnerLinks>

   <variables>
      ...
   </variables>

   <sequence>
     <receive partnerLink="customer"
              portType="lns:purchaseOrderPT"
              operation="sendPurchaseOrder"
              variable="PO"/>
     <flow>
      <sequence>
       <assign>
         <copy>
           <from container="PO" part="customerInfo"/>
           <to      variable="shippingRequest"
                    part="customerInfo"/>
         </copy>
       </assign>
       <invoke  partnerLink="shippingProvider"
                portType="lns:shippingPT"
                operation="requestShipping"
                inputVariable ="shippingRequest"
                outputVariable="shippingInfo"/>
       <receive partnerLink="shippingProvider"
                portType="lns:shippingCallbackPT"
                operation="sendSchedule"
                variable="shippingSchedule"/>
     </sequence>
```

```
45.     <sequence>
46.       <invoke  partnerLink="invoiceProvider"
47.                portType="lns:computePricePT"
48.                operation="initiatePriceCalculation"
49.                inputVariable="PO"/>
50.       <invoke  partnerLink="invoiceProvider"
51.                portType="lns:computePricePT"
52.                operation="sendShippingPrice"
53.                inputVariable="shippingInfo"/>
54.     </sequence>
55.    </flow>
56.
57.    <reply partnerLink="customer"
58.        portType="lns:purchaseOrderPT"
59.        operation="sendPurchaseOrder"
60.        variable="Invoice"/>
61.   </sequence>
62. </process>
```

Listing 3.12 Beispiel einer BPEL4WS-Prozessbeschreibung

Diese ausführbare Prozessdefinition beginnt mit einer Hauptaktivität `sequence` (Zeile 21) Aktivität. Sie umfasst eine `receive` (Zeile 22), eine `flow` (Zeile 26) strukturierte und eine `reply` (Zeile 57) Aktivität, die alle nacheinander ausgeführt werden müssen. Die zwei `sequence` Aktivitäten (Zeilen 27,45) innerhalb des `flow` Elements können parallel ausgeführt werden. Alle `invoke`, `receive`, und `reply` Aktivitäten werden auf einer festgelegten Operation eines WSDL-PortTypes ausgeführt.

3.5 Zusammenfassung

B2B-Standards erlauben den Austausch von Geschäftsdokument in bekannten Formaten. openTRANS ist eine Spezifikation, die das Format von Geschäftsdokumenten im XML-Format festlegt.

Per UDDI können Web Services im Internet gefunden werden. WSDL definiert die Schnittstellen eines Web Service. Der Austausch ihrer Daten erfolgt per SOAP. SOAP, WSDL und UDDI basieren auf XML.

Web Services eigenen sich in hohem Maße für die Übertragung von XML-Dokumenten, wie openTRANS-Geschäftsdokumenten.

Um eine Integration von Web Services zu ermöglichen muss die Bedeutung der die einzelnen Operationen in maschinenverständlichem Format vorliegen, wozu sich semantische Technologien sehr gut eignen. OWL-S erlaubt als Process Description Languages zudem die Prozessbeschreibung von Web Services.

Zur Ausführung komponierter Prozesse werden Process Execution Languages benötigt, die auf dazugehörige Engines installiert werden können.

ebBPSS ist keine ausführbare Prozessbeschreibung von Web Services, sondern regelt, allerdings sehr exakt, wie zwei oder mehr Services zusammenzuarbeiten

haben. Die technische Realisation des spezifizierten Geschäftsprozesses liegt beim Entwickler.

BPEL4WS ist eine sofortausführbare Prozessbeschreibung für Web Services. Sie besitzt alle Informationen, die benötigt werden, um die Operationen der komponierten Web Services direkt anzusprechen.

4 LÖSUNGSKONZEPT

Kapitel 2 hat aufgezeigt welche Anforderungen an eine dynamische automatische B2B-Integration im Allgemeinen gestellt werden.

In Kapitel 3 wurden Technologien vorgestellt, mit denen das folgende Lösungskonzept realisiert werden kann.

Das Lösungskonzept stellt nicht den Anspruch eine Prozessintegration in allen Einzelheiten abbilden zu können, sondern zeigt die grundsätzliche Machbarkeit und welche Mindestvoraussetzungen dafür erfüllt sein müssen.

Dazu werden als erstes die Grundvoraussetzungen (Kapitel 4.1) geklärt, die für die Prozessintegration gegeben sein müssen. Dazu zählen unter anderem die Anforderungen an die Systeme der Geschäftspartner und an das Prozessintegrations-System.

In Kapitel 4.2 wird der allgemeine Ablauf beschrieben, wie in diesem Lösungskonzept die Prozessintegration realisiert wird. Daraus ergibt sich ein grundsätzlicher Ansatz, der eine erste Einteilung des Systems in seine einzelnen Komponenten vornimmt.

Die Anforderungen an die einzelnen Komponenten werden in Kapitel 4.4 erarbeitet. Dazu zählen zu einem großen Teil die Bedingungen, die an die Prozessbeschreibungen gestellt werden. Die gewonnenen Anforderungen erlauben die Technologien auszuwählen, die im folgenden Systementwurf verwendet werden.

Im Systementwurf (Kapitel 4.6) werden die einzelnen Schritte der Prozessintegration betrachtet, und Lösungen erarbeitet, mit denen die Realisierung des Konzepts möglich ist. Dazu gehören

- die Systemarchitektur,
- die Erweiterung der semantischen Technologien für die potenzielle Prozessbeschreibung,
- die Transformation von potenzieller in ausführbare Prozessbeschreibung und
- die Metaprozessbeschreibung, die die Mindestbedingungen umfasst, die die potenziellen Prozessbeschreibungen der Geschäftspartner erfüllen müssen.

4.1 Grundvoraussetzungen

Anforderungen an die B2Bi

Eine Lösung zur dynamischen Prozessintegration zu liefern, ist das Hauptziel. Daher werden für die in Kapitel 2.3.5 herausgestellten Anforderungen an B2Bi einige Annahmen und Vorauswahlen von Technologien getroffen:

- Unternehmensinterne Anpassungen werden als gegeben vorausgesetzt. (Punkt 1 und 2 der Anforderungen)
- Zur Nachrichtenübermittlung werden Web Services genutzt. Deren Interface wird in WSDL dargestellt, die die Schnittstellendefinitionen unter Punkt 5 der Anforderungen erfüllt.
- Die Kommunikation der Unternehmenssysteme erfolgt über das Internet. (Punkt 7)
- Web Services können an verschiedenste Übertragungsprotokolle gebunden werden (Bindung). In diesem Prototypen werden Nachrichten mit Hilfe des Nachrichtenaustauschformats SOAP über HTTP übertragen. (Punkt 9)
- Als Datenformat wird openTRANS genutzt, dessen Geschäftsdokumente in der XML-Syntax dargestellt werden. (Punkt 10 und 11)

Ungeklärt bleiben bis zu diesem Zeitpunkt die Punkte 3 4, 6 und 7 der Anforderungen.

Die Technologien, die sich für unternehmensexterne Prozessabläufe eignen, und die Technologien, die für die Prozessbeschreibungen der Services eingesetzt werden, sollen hier bewertet und festgelegt werden. Zudem muss geklärt werden, wie asynchrone Kommunikation ermöglicht werden kann.

Rollen

- Der Einkäufer:
  Der Einkäufer hat nach Abschluss der Wissensphase einen potenziellen Geschäftspartner gefunden und möchte in die Abwicklungsphase übergehen. Dazu startet er das Integrationssystem.
- Der Verkäufer:
  Der Verkäufer bietet einen openTRANS-Web Service und eine dazugehörige potentielle Prozessbeschreibung an, die eine Integration seines Web Service in einen komponierten Prozess erlauben.
- Mittelsmänner
  Mittelsmänner stellen die Komponenten des Integrationssystems zur Verfügung, sofern der Einkäufer bzw. Verkäufer diese Komponenten nicht selbst anbietet. Die Mittelsmänner dienen somit als e-Procurement Service Provider (e-PSP).

Einordung in die Markttransaktion

Nicht berücksichtigt wird für dieses System das automatische Auffinden und die Auswahl eines passenden Geschäftspartner-Web Service bzw. die Wissensphase, wie sie in einer typischen Markttransaktion als erstes stattfindet. Die Wissensphase wird für diese Arbeit als abgeschlossen betrachtet.

Ein- und Verkäufer können verschiedene Prozessbeschreibungen anbieten. So kann ein Verkäufer zum Beispiel verschiedene Prozessbeschreibungen pro Produkt anbieten, oder pro Zielgruppe. Es obliegt dem Einkäufer, die richtige Prozessbeschreibung auszuwählen bzw. dem Verkäufer, dem Einkäufer die richtige anzubieten. Auch diese Auswahl der Prozessbeschreibung ist Teil der Wissensphase und wird hier nicht vertieft.

Der Einkäufer hat nun nach Abschluss der Wissensphase einen potenziellen Geschäftspartner gefunden, der über einen entsprechenden openTRANS-Web Service für den Verkauf verfügt, und möchte in die Absichtsphase übergehen, wozu eine Prozessintegration nötig ist.

Den Anstoß für die Prozessintegration gibt demzufolge der Einkäufer. Der Verkäufer stellt nur sein System und dessen Beschreibung bereit.

Anforderungen an die Prozessintegration

Alle Informationen des Geschäftspartners, die der Einkäufer für eine Prozessintegration benötigt, sind dem Einkäufer bekannt. Dazu gehören

- die Schnittstellendefinitionen des Web Service (WSDL) und das Nachrichtenaustauschformat (SOAP),
- die Datenformate die der Geschäftspartner benötigt (openTRANS), sowie
- die potenzielle Prozessbeschreibung des Geschäftspartners, die bestimmt, wie die Web Service genutzt werden kann.

Die Integration und die Ausführung des Prozesses können bei einem der Geschäftspartner liegen. Liegt sie beim Einkäufer, handelt es sich um eine sogenannte Buy-Side-Lösung. Liegt sie beim Verkäufer, handelt es sich um eine Sell-Side-Lösung [Schubert1, S. 4].

Sie kann aber sich aber auch auf einen oder mehrere Mittelsmänner oder auch auf Mittelsmann (Komposition) und Geschäftspartner (Ausführung) verteilen.

Diese Variante wäre auch unter Berücksichtigung sicherheitsrelevanten Aspekten zu bevorzugen, da über das ausführende System – die Process Engine – alle Daten der Markttransaktion fließen, was unter Umständen von den Geschäftspartnern nicht gewünscht wird, vor allem bei unverschlüsselten Daten.

In dieser Arbeit soll ein komponentenbasiertes Konzept verfolgt werden, das die Nutzung des Integrationssystems in allen Varianten zulässt.

Zudem muss eine Prozessintegration unabhängig von den Systemen des Ein- und Verkäufers umgesetzt werden können, um die Prozessintegration so dynamisch wie

möglich zu halten. Anpassungen an die Systemen der Geschäftspartner sind, wie in Kapitel 1.2 festgestellt, bei einer B2Bi nicht so einfach durchzusetzen. Zudem verhindert die Notwendigkeit von Anpassungen die Integration und Nutzung „on the fly", da gewartet werden muss, bis der Geschäftspartner die Anpassungen vorgenommen hat.

Die beiden Anforderungen an das Integrationssystem im Überblick:

1. Ein komponentenbasiertes Integrationssystem muss angestrebt werden.
2. Das Integrationssystem muss unabhängig von den e-Procurement-Systemen der Geschäftspartner sein.

4.2 Allgemeiner Ablauf der Prozessintegration

Ein den obigen Ansprüchen funktionierendes System müsste folgendermaßen ablaufen:

Das System des Einkäufers besteht, wie das des Verkäufers, aus einem Web Service, der für den Empfang und das Versenden von openTRANS-Geschäftsdokumenten zuständig ist (openTRANS-Web Service), und einer genauen maschinenlesbaren potentiellen Prozessbeschreibung.

Eine solche Prozessbeschreibung definiert, welche Schnittstellen (Operationen des Web Service) er anbietet, um Geschäftsdokumente zu empfangen und welche Schnittstellen er von seinem Gegenüber erwartet, um diesem Geschäftsdokumente zustellen zu können. Für die Beschreibung der Schnittstellen werden semantische Technologien benötigt, die die Bedeutung der einzelnen Web Service Operationen für das Integrationssystem klären.

In der Prozessbeschreibung sind zudem die Kompositionsalternativen enthalten, mit der die Schnittstellen des openTRANS-Web Service genutzt werden dürfen.

Unter anderem müssen in der potenziellen Prozessbeschreibung Reihenfolge, Kardinalität, Abhängigkeiten, Kompensationsmöglichkeiten, Zeitlimits und mögliche Fehlermeldungen beschrieben werden.

Der Einkäufer startet den Integrationsprozess., wie in Abbildung 4.1 zu sehen.

Er sendet seine potenzielle Prozessbeschreibung und die des Verkäufers an das zu entwickelnde Integrationssystem.

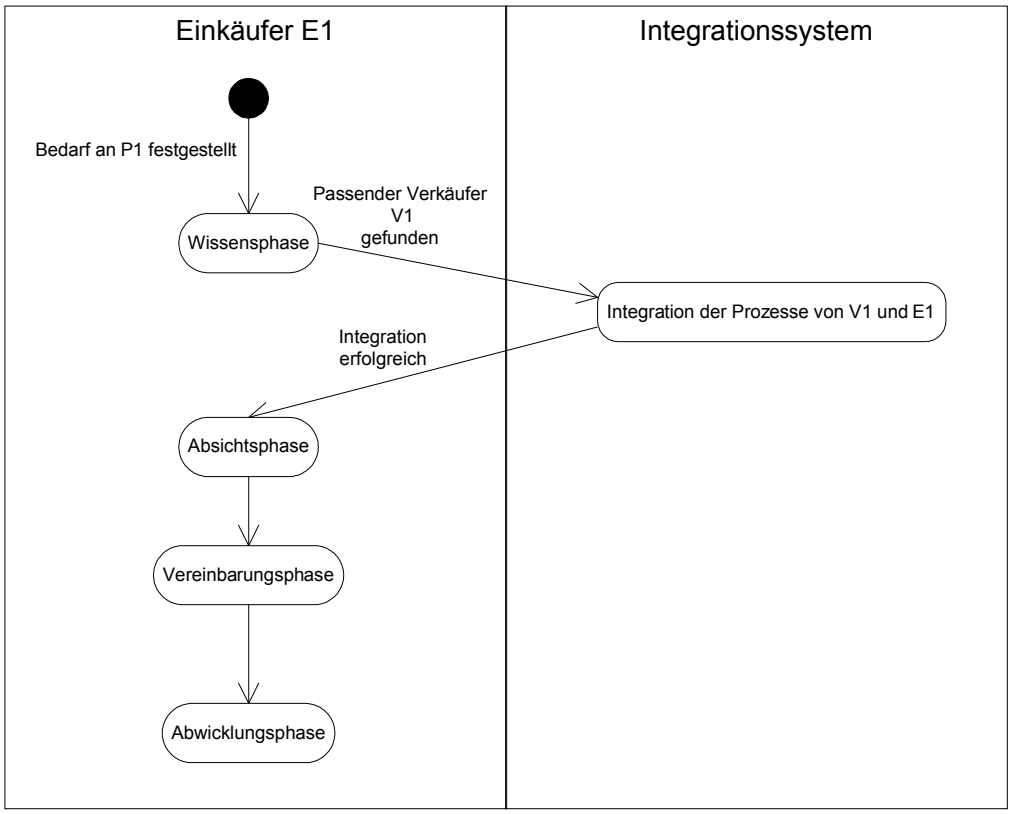

Abbildung 4.1 Einordnung der Prozessintegration in die Markttransaktion

Nun werden die beiden Prozessbeschreibungen aufeinander abgestimmt. Unter anderem wird geprüft, ob alle Schnittstellen vorhanden sind, die die Geschäftspartner gegenseitig verlangen und in welcher Reihenfolge sie gerufen werden dürfen bzw. müssen.

Existieren keine grundsätzlich Widersprüche, die der Geschäftspartner erwartet, wählt das System die geeignetste Kombinationsmöglichkeit der beiden Prozessbeschreibungen und generiert eine komponierte Prozessbeschreibung. Diese wird nun als eigener Web Service, gestartet, der im folgenden Flow-Service genannt wird. In Abbildung 4.2 kann dieser Prozess nachvollzogen werden.

Der Flow-Service dient als Vermittler zwischen Einkäufer und Verkäufer. Aus der Perspektive der Geschäftspartner erscheint der Flow-Service als ideales Abbild der von ihnen gestellten Anforderungen an ihr Gegenüber (siehe Abbildung 4.3).

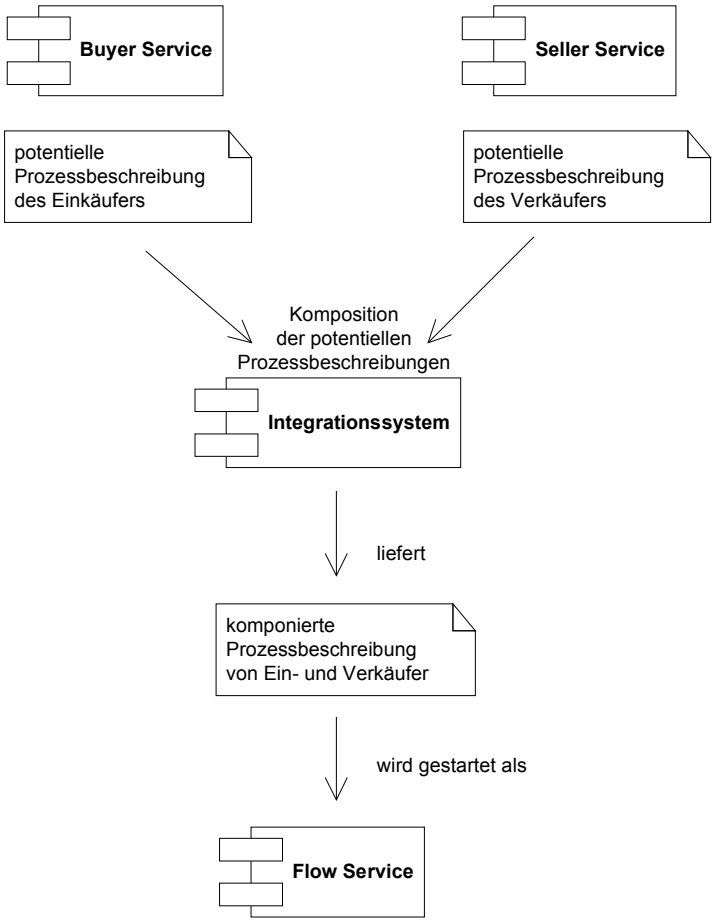

Abbildung 4.2 Von den Prozessbeschreibungen zum Flow-Service

So wird gewährleistet, dass die openTRANS-Web Services nie direkt die Schnittstellen des Geschäftspartners ansprechen müssen, und sich daher auch nicht an deren Schnittstellen anpassen müssen. Die e-Procurement - Systeme werden im Sinne einer B2Bi aneinander gekoppelt, ohne dass eine Anpassung in den Systemen der Partner nötig wird.

Nun wird der Einkäufer darüber informiert, dass der Flow-Service bereit steht. Die Absichtsphase beginnt.

Ab jetzt läuft der Prozess vollautomatisch im Rahmen der durch die potenziellen Prozessbeschreibungen festgelegten Parameter ab: Der Verkäufer antwortet auf die Geschäftsdokumente des Einkäufers wiederum an den Flow-Service, der die Dokumente weiterreicht usw.

Dabei erwartet der Flow-Service immer den Aufruf der Schnittstellen in der Reihenfolge, die durch die Prozessbeschreibung festgelegt wurde. Eine fehlerhafte Reihenfolge wird damit ausgeschlossen.

4.3 Ansatz

Im Rahmen der Zielsetzung dieser Arbeit in Kapitel 1.3 und des allgemeinen Ablaufs einer Prozessintegration in Kapitel 4.2 werden nun die Komponenten bestimmt, die für das Lösungskonzept notwendig sind.

Einem Einkäufer soll die Möglichkeit gegeben werden seinen Einkaufs-Web Service auf den eines Verkäufers abzustimmen und zu starten, ohne dass eine manuelle Anpassung stattfinden muss. Überflüssig wird zum Beispiel welche Web Service-Operationen des Einkäufers die des Verkäufers aufrufen müssen und umgekehrt.

Dazu wird hier ein System – das openTRANS-Integrationssystem – vorgestellt, mit dem es möglich sein soll anhand der beschreibenden Komponenten – den potentiellen Prozessbeschreibungen – zu erkennen, wie die Web Services der Geschäftspartner miteinander verbunden werden können. Für die potenzielle Prozessbeschreibung muss eine Process Description Language (PDL) ausgewählt werden, wie sie in Kapitel 3.3 beschreiben wurde.

Das System generiert eine komponierte Prozessbeschreibung aus beiden Systemen. Diese muss daraufhin in eine ausführbare Prozessbeschreibung transformiert werden, die in einer Process Execution Language (PEL) erstellt wird, wie sie in Kapitel 3.4 beschreiben werden.

Durch den Zwischenschritt der Umwandlung, zuerst in eine komponierten Prozessbeschreibung und danach erst in eine ausführbare Prozessbeschreibung, wird die Nutzungsmöglichkeit verschiedener PELs gewährleistet.

Die erstellte ausführbare Prozessbeschreibung wird daraufhin automatisch als Web Service gestartet.

Es lassen sich hier vier Teilaspekte isolieren, die nach der Anforderung an ein komponentenbasiertes System als einzelne Komponenten umgesetzt werden sollen.

1. Process Integrator – die verwaltende Komponente des Systems, die mit Hilfe der folgenden Komponenten den Flow-Service erstellt und startet.
2. Process Composer – komponiert die potenziellen Prozessbeschreibungen
3. Process Transformer – wandelt die komponierte in eine ausführbare Prozessbeschreibung um.
4. Process Engine – auf dieser startet der Process Integrator die ausführbare Prozessbeschreibung als Web Service.

4.4 Anforderungen

4.4.1 Allgemeine Anforderungen an die PPD

Die Process Description Language (PDL), in der die potenzielle Prozessbeschreibung (PPD) verfasst wird, muss verschiedenen Ansprüchen genügen.

Das SWWS[24] [SWWS2], stellt zwölf Punkte zusammen, mit denen sich Web Services beschreiben müssen, um ihre Interoperabilität zu gewährleisten. [vgl. WSMF1]:

1. Name.
2. Ziel des Service.
3. Vor und Nachbedingungen: Welche Informationen müssen in welchem Format vorliegen, damit der Service ausgeführt werden kann. Beispielsweise muss der Standort des Nutzers bekannt sein, bevor er nach dem nächsten Restaurant fragen kann.
4. Ein- und Ausgabedaten: Anlehnend an dem Restaurant Beispiel unter Punkt 3 wird hier festgelegt, dass eine geographische Größe eingegeben, und eine andere zurückgegeben wird.
5. Fehlermeldungen: Nur teilweise Erfüllung oder die Notwendigkeit weiterer Eingaben müssen an den Nutzer gemeldet werden.
6. Dynamische Nutzung weiterer Web Services zur Laufzeit (invoked web service proxy).
7. Datenfluss: Der Fluss der Informationen zwischen den einzelnen Aktivitäten.
8. Kontrollfluss: Die Reihenfolge und Anordnung der einzelnen Aktivitäten.
9. Ausnahmebehandlungen wie Abbruchbedingungen oder Wiederholung,
10. Kompensation von Ausnahmen wie zum Beispiel der Rollback einer Transaktion.
11. Wie setzt ein Web Service die Anforderungen an sein Nachrichtenaustauschprotokoll um. Wie garantiert er Authentifizierung, Verlässlichkeit, Nicht Wiederholbarkeit und Sitzungs-Management.
12. nicht-funktionale Eigenschaften wie geographischer Wirkungskreis u.ä.

Einschränkung der Anforderungen des SWWS

Auf fast alle dieser Anforderungen kann durch die Festlegung des Systems auf bestimmte Technologien und einen bestimmten Zweck, eine Markttransaktion, verzichtet werden.

Das Ziel besteht selbstverständlich in der Abwicklung einer Markttransaktion, was den Geschäftspartnern klar ist, und daher nicht beschrieben werden muss. So ist die Beschreibung der Ein- und Ausgabedaten und auch des Datenflusses im Falle einer Nutzung des openTRANS-Formates ebenfalls unnötig. Eine dynamische Nutzung weiterer Web Services wäre nur dann von Vorteil, wenn der Prozess zusätzliche Aufgaben wie die Überprüfung der Kreditfähigkeit eines Kunden enthielte. Diese Ziele werden in diesem Lösungskonzept jedoch nicht verfolgt.

[24] SWWS ist ein Projekt des Information Society Technology (IST) Programms für Forschung, Technologie Entwicklung und Demonstration unter dem "5th Framework Programm" der Europäischen Kommision. [SWWS1]

Die obigen Punkte sind im Falle der Beschreibung von openTRANS-Web Services nur von Vorteil, wenn sie auch von anderen e-Procurement Anwendungen genutzt werden sollten.

Es ist vorstellbar, dass ein noch mächtigeres Integrationssystem als das hier vorgestellte anhand dieser Beschreibungen erkennt, dass ein openTRANS-Web Service beispielsweise mit einem RosettaNet-Web Service kommunizieren möchte.

Da die Geschäftsdokumente der beiden Spezifikationen eine völlig andere Struktur besitzen, müsste dann unter anderem in den generierten Prozess ein e-PSP eingesetzt werden, der bei jedem Senden und Empfangen die Dokumenten übersetzt.

In diesem Beispiel wäre eine exakte Beschreibung der Ein- und Ausgabedaten und der Nutzung von dynamischen Web Services wie dem Übersetzter zwischen den Dokumenten notwendig.

Nicht-funktionale Eigenschaften sind prinzipiell nur in der Wissensphase einer Markttransaktion von Interesse.

Fehlermeldungen, Ausnahmebehandlungen und Kompensation

Die unter Punkt 5. genannten Fehlermeldungen werden in einer Prozessbeschreibung für den Austausch von openTRANS-Geschäftsdokumente (kurz openTRANS-Prozessbeschreibung) benötigt, wenn das Zusenden eines Geschäftsdokuments ausbleibt bzw. den Timeout, den ein Geschäftspartner angegeben hat, überschreitet. In einem solchen Fall würde ein Geschäftspartner, darüber informiert werden, dass der Eingang seines Geschäftsdokuments überfällig ist. Das Lösungskonzept wird solche Fehlermeldungen allerdings noch nicht berücksichtigen.

Ein Flow-Service nimmt das gesendete Dokument allerdings gar nicht erst an, wenn es nicht in den Prozessablauf passt. Wird das Dokument vom Flow-Service nicht angenommen, erkennt der Geschäftspartner dies am Response auf seine Nachricht.

In diesem Lösungskonzept wird nur die Kompensation für Dokumente erlaubt, die ein Geschäftspartner öfter sendet, als der andere sie empfangen kann bzw. will. Dabei kann der die Kompensation deklarierende Partner eine Web Service Operation angeben kann, die gerufen wird.

Fehlerbehandlungen und auch Ausnahmebehandlungen für ein Transaktionsmanagement werden in diesem Lösungskonzept nicht berücksichtigt.

Nachrichtenaustauschprotokoll

Die Anforderungen an ein Nachrichtenaustauschprotokoll werden in diesem Lösungskonzept mit Ausnahme vom Sitzungs-Management vernachlässigt. Wichtig für einen ablaufenden Prozess ist vor allem, dass die eingehenden Nachrichten exakt einer Sitzung (einer laufenden Instanz einer openTRANS-Prozessbeschreibung) zugeordnet werden können. Dies kann durch das Nachrichtenaustauschprotokoll oder über Informationen in den übertragenen Geschäftsdokumenten geschehen. Für dieses Lösungskonzept wird eine Vorauswahl auf eine Art des Sitzungs-Management getroffen, weshalb sie in der potenziellen Prozessbeschreibung nicht

beschrieben werde muss. Sie ist daher nur für die ausführbare Prozessbeschreibung von Interesse (siehe nächstes Kapitel).

Letzten Endes können die folgenden zwei Anforderungen an die PDL extrahiert werden, die für die hier zu realisierende automatische Prozessintegration erfüllt sein müssen. Hinzu kommen weitere sechs übergreifende Anforderungen.

1. Kompensationsfähigkeit.
2. Kontrollfluss. Hierbei sind vor allem die noch zu analysierenden Anforderungen an die Darstellung einer openTRANS-Prozessbeschreibung zu beachten.

Zudem sind folgende übergreifende Anforderungen zu beachten, die sich nicht direkt auf die Kontrollflusskonstrukte der PDL beziehen:

3. Eignung der PDL für die Beschreibung von Web Services.
4. Darstellbarkeit der Zusammenarbeit verschiedener Partner und der Rollen, die sie dabei einnehmen.
5. Neutralität. Die Prozessbeschreibung sollte möglichst auf open source Technologien und Standards beruhen, um eine möglichst breite Anwendbarkeit garantieren zu können und nicht von kommerziellen Entscheidungen abhängig zu sein.
6. Entwicklungsstand und Erweiterbarkeit hinsichtlich im Folgenden nicht weiter behandelter Fehlerbehandlungen und übergreifender Integrationssysteme wie oben beispielhaft erwähnt.
7. Da die potenzielle Prozessbeschreibung die einzige Komponente des Lösungskonzepts ist, auf deren Gestaltung die Geschäftspartner direkt Einfluss nehmen, sollte die Technologie einfach anwendbar sein, um eine Anwendung nicht zu erschweren oder Fehler zu provozieren.
8. Semantische Technologien sollen an dieser Stelle verwendet werden, um die Bedeutung der Web Service Operationen, die in einer Prozessbeschreibung genutzt werden zu beschreiben.

4.4.2 Allgemeine Anforderungen an die EPD

Die ausführbare Prozessbeschreibung (EPD) wird in einer der bekannten Process Execution Languages (PEL) geschrieben. Teilweise überschneiden sich die Anforderungen für potenzielle und ausführbare Prozessbeschreibung. So müssen beide Kontrollflusskonstrukte bereitstellen.

Der Unterschied der beiden Beschreibungssprachen liegt darin, dass die Beschreibungen in der PDL noch aufeinander abgestimmt werden müssen, und die Beschreibungen in der PEL sich auf konkrete Operationen eines Web Service oder andere Aktivitäten beziehen, die reale Auswirkungen haben.

Hier sind die für dieses System wichtigen Anforderungen zusammengestellt, die auch für die Beschreibung eines B2B Prozesses allgemein notwendig sind [vgl. ORiordan1].

1. Die Zusammenarbeit verschiedener Parteien muss dargestellt werden können

2. Transaktions-Management auch für langlebige Prozesse, die im B2B oft auch Tage oder Wochen dauern können.
   ⇨Eine Umsetzung ist in diesem Lösungskonzept nicht vorgesehen.
3. Nachrichtenaustauschprotokoll-Anforderungen (siehe oben).
4. Sitzungs-Management.
   Die über die Web Service Operationen eingehenden Geschäftsdokumente müssen einer Prozessinstanz zugeordnet werden können.
5. Kontrollfluss. Die Reihenfolge von Aktivitäten muss ausgedrückt werden können.
   ⇨ Hierzu werden noch die speziellen Anforderungen einer openTRANS-Prozessbeschreibung herausgestellt.
6. Kompensationsmöglichkeiten werden benötigt.
7. Eignung für Web Services. Die Prozessbeschreibung sollte leicht für kooperierende Web Services genutzt werden können.

Audit Trail – die Rückverfolgbarkeit ist in B2B-Anwendungen aus rechtlichen Gründen wichtig. So könnten vom Prozess digital signierte Antworten auf einen Nachrichteneingang an die Geschäftspartner versendet werden, um einem Prozess die Bestreitbarkeit seines abgeschlossenen Ablaufs zu nehmen.

Transaktions-Management dient der Ausnahmebehandlung von ungewünschten Zuständen in laufenden Prozessen und geht über die reine Kompensation hinaus, die im vorherigen Kapitel beschrieben wurde. Müssen Aktionen wiederholt oder zurückgenommen werden (Rollbacks), sind dabei wichtige Aspekte.

Audit Trail und Transaktions-Management werden in dieser Arbeit nicht berücksichtigt, da sie für die reine Realisation einer Prozessintegration nicht von Bedeutung sind.

4.4.3 Inhaltliche Anforderungen an die Prozessbeschreibungen

Die Anforderungen an eine Prozessbeschreibungssprache, egal ob PPD oder EPD, in der ein openTRANS-Prozessesbeschreibung[25] umgesetzt wird, muss alle benötigten Kontrollflusskonstrukte bereitstellen, bzw. dazu erweiterbar sein. Dazu werden in diesem Kapitel die benötigten Konstrukte vorgestellt.

In der PPD dürfen zudem nur bestimmte Strukturierungen der Geschäftsdokumente zugelassen werden, um eine sinnvolle und ausführbare openTRANS-Prozess-beschreibung zu gewinnen. Auch die Strukturierungsmöglichkeiten werden hier erläutert und führen zu Mindestbedingungen, die ein Geschäftspartner bei der Erstellung einer PPD beachten muss.

Es wird nun geklärt, welcher Sende- und Empfangs-Ablauf der openTRANS-Geschäftsdokumente sinnvoll ist.

Dazu wird in Erfahrung gebracht,

1. welche Abhängigkeiten zwischen den Dokumenten bestehen,

[25] Eine Prozessbeschreibung für web Services die dem Austausch von openTRANS-Geschäftsdokumenten dienen

2. welche Kardinalitäten für jedes Geschäftsdokument in einer openTRANS-Prozessbeschreibung gelten,
3. welche Dokumente gesendet und empfangen werden müssen und
4. ob sie parallel oder sequentiell übermittelt werden können.

Hieraus lässt sich ein openTRANS-Metaprozessbeschreibung gewinnen, die festlegt, welche inhaltlichen Anforderungen die PPD einer openTRANS-Prozessbeschreibung mindestens erfüllen muss, z.B. dass eine RFQ vor einer QUOTATION erfolgen muss.

So kann das Integrationssystem die Gültigkeit der Selbstbeschreibungen der Web Services überprüfen.

Die Geschäftsdokumente

Es werden nun kurz die Bedeutungen der openTRANS-Geschäftsdokumente näher erläutert, die in einer Markttransaktion Verwendung finden:

- RFQ: Eine Angebotsanforderung des Einkäufers über eine oder mehrere Waren.
- QUOTATION: Ein Angebot des Verkäufers über eine oder mehrere Waren.
- ORDER: Eine Bestellung des Einkäufers von einer oder mehreren Waren.
- ORDERRESPONSE: Eine Reaktion des Verkäufers auf eine Bestellung. Damit wird zum einen signalisiert, dass die Bestellung vom System des Verkäufers verarbeitet wurde. Des weiteren können hier Informationen, wie die Ablehnung der Bestellung enthalten sein.
- ORDERCHANGE: Änderung einer Bestellung durch den Einkäufer. Eine Änderung ist im weitesten Sinne auch der Abbruch einer Markttransaktion.
- DISPATCHNOTIFICATION: Die Versandbenachrichtigung des Verkäufers über die bestellten Waren.
- RECEIPTACKNOWLEDGEMENT: Die Eingangsbestätigung des Einkäufers über den Erhalt der bestellten Waren.
- INVOICE: Die Rechnung über die bestellten Waren.

openTRANS-Interaktionsablauf

Eine ausführbare Prozessbeschreibung für den Austausch von openTRANS-Geschäftsdokumenten zur Durchführung einer Markttransaktion wird als ausführbarer Prozess, als Flow-Service, bereitgestellt. Ein Interaktionsablauf sei hier die Instanz eines solchen Prozesses.

e-Procurement-Beziehungen können beliebig komplex werden. Eine ORDER kann abgeschickt, widerrufen und erneut abgeschickt werden. In dieser Arbeit ist die ORDER zentrales Dokument eines Interaktionsablaufs. Für jede ORDER wird ein eigener Interaktionsablauf gestartet.

Zudem können georderte Waren in verschiedenen Lieferungen verschickt werden. Der Einfachheit halber wird in dieser Arbeit von einer Lieferung pro ORDER ausge-

gangen. Da Kardinalitäten höher als 1 prinzipiell berücksichtigt werden, ist ein Ausbau des Lösungskonzepts auf mehrere Teillieferungen leicht möglich.

Für den Interaktionsablaufs werden hier folgende Annahmen getroffen:

1. Der Interaktionsablauf startet entweder mit einer RFQ, auf die eine QUOTATION folgen muss, oder einer ORDER. RFQs und QUOTATIONs können beliebig oft vorkommen, eine ORDER nur einmal.
2. Auf eine ORDER kann genau eine DISPATCHNOTIFICATION folgen. Wenn sie vom Verkäufer gesendet wird, muss sie zeitlich nach einer ORDER erfolgen.
3. Auf eine DISPATCHNOTIFICATION kann genau eine RECEIPTACKNOWLEDGEMENT folgen. RECEIPTACKNOWLEDGEMENT wird nicht durch eine DISPATCHNOTIFICATION bedingt, aber sie erfolgt in jedem Fall nach einer DISPATCHNOTIFICATION, da die Waren nicht ankommen können, bevor sie versendet wurden.
4. Auf eine ORDER kann genau eine INVOICE folgen.
5. Nach der ORDER können ein oder mehrere ORDERCHANGE folgen, auf die wiederum eine zugehörige ORDERRESPONSE folgen kann, so lange die Waren nicht verschickt wurden.
6. Nach einer INVOICE sind eine oder mehrere ORDERCHANGE samt ORDERRESPONSE noch erlaubt. Nach einer DISPATCHNOTIFICATION oder RECEIPTACKNOWLEDGEMENT macht sie keinen Sinn mehr. Der Einkaufsprozess kann nicht mehr abgebrochen werden, da die Waren schon verschickt wurden. Jetzt wäre nur noch ein Widerruf der Markttransaktion möglich. Dieser Fall wird durch die openTRANS-Spezifikation nicht abgedeckt, da er nach der eigentlichen Markttransaktion stattfindet.
7. INVOICE und DISPATCHNOTIFICATION sind die einzigen Aktivitäten in einem openTRANS-Interaktionsablauf, die nebenläufig sein bzw. von den Geschäftspartnern in verschiedene Reihenfolgen gesetzt werden können. So kann ein Einkäufer verlangen eine INVOICE vor einer DISPATCHNOTIFICATION zu erhalten, oder umgekehrt. Er könnte auch festlegen, dass ihm die Reihenfolge gleichgültig ist.

Abhängigkeiten

Die Abhängigkeiten beziehen sich nur auf den nach außen sichtbaren Prozess. In den internen Systemen der Geschäftspartner können sehr wohl Beziehungen und Abhängigkeiten zwischen beispielsweise RFQ und ORDER bestehen. Aber für den nach außen sichtbaren Prozess ist eine solche Beziehung nicht von Bedeutung. Eine ORDER kann auch ohne eine vorhergehende RFQ in dem Prozessablauf vorkommen.

Im Vordergrund stehen die Mindestanforderungen an die Abhängigkeiten der Dokumente, die in der potenziellen Prozessbeschreibung eines Geschäftspartners erfüllt sein müssen. Darauf aufbauend ist festzulegen, in wieweit diese Abhängigkeiten abgeändert bzw. erweitert werden können. Tabelle 4.1 zeigt die Abhängigkeiten der Geschäftsdokumente.

Geschäftsdokument	**abhängig von**
RFQ	unabhängig
QUOTATION	RFQ
ORDER	unabhängig
ORDERRESPONSE	ORDER oder ORDERCHANGE
ORDERCHANGE	ORDER
DISPATCHNOTIFICATION	ORDER
RECEIPTACKNOWLEDGEMENT	ORDER
INVOICE	ORDER

Tabelle 4.1 Abhängigkeiten der Geschäftsdokumente

Mit den bis hier erzielten Ergebnissen lässt sich der Aufbau des openTRANS-Interaktionsablauf, in einem UML-Aktivitätsdiagramm beschreiben. Abbildung 4.4 zeigt den Interaktionsverlauf, der die Mindestanforderungen an die Abhängigkeiten und die Reihenfolge der Geschäftsdokumente berücksichtigt.

Ein Geschäftspartner muss seine potenzielle Prozessbeschreibung an diesem Modell orientieren, um zu einer gültigen Prozessbeschreibung zu gelangen. Dabei kann er Geschäftsdokumente weglassen und nebenläufige Sequenzen in eine gemeinsame Sequenz stellen. Im Fall einer openTRANS-Prozessbeschreibung bedeutet dies, dass eine Sequenz, in der eine INVOICE empfangen wird, in eine Sequenz, in der eine DISPATCHNOTIFICATION samt RECEIPTACKNOWLEDGEMENT empfangen wird, eingeordnet werden kann. Eine INVOICE kann also entweder vor oder nach einer DISPATCHNOTIFICATION gesendet werden oder erst nach einer RECEIPTACKNOWLEDGEMENT.

In dieser Arbeit wird eine Einordnung von INVOICE zwischen DISPATCHNOTIFICATION und RECEIPTACKNOWLEDGEMENT nicht erlaubt, um Deadlocks im Prozessablauf zu vermeiden. Ein Deadlock würde auftreten, wenn ein Verkäufer eine RECEIPTACKNOWLEDGEMENT vor dem Versenden einer INVOICE verlangt, und ein Einkäufer eine RECEIPTACKNOWLEDGEMENT erst versendet, wenn er eine INVOICE erhalten hat (siehe auch Kapitel 4.6.8).

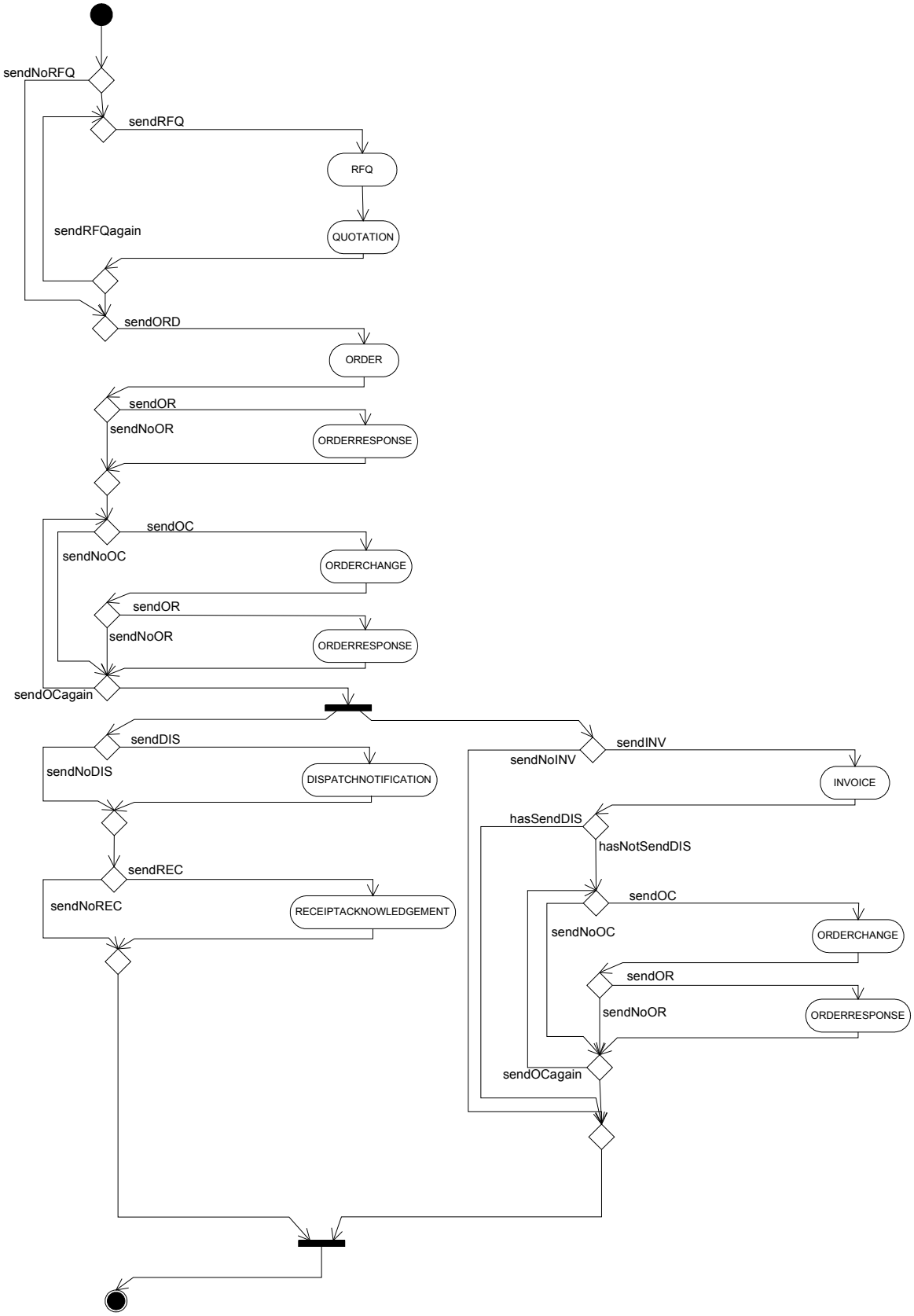

Abbildung 4.4 openTRANS-Interaktionsablauf

Jede Aktivität des Diagramms entspricht dem Senden und Empfangen eines openTRANS-Geschäftsdokuments. Jedes Geschäftsdokument kann genau einem Geschäftspartner als Sender und dem anderen als Empfänger zugesprochen werden (siehe Tabelle 4.2).

Geschäftsdokument	**gesendet von Geschäftspartner**
RFQ	Einkäufer
QUOTATION	Verkäufer
ORDER	Einkäufer
ORDERRESPONSE	Verkäufer
ORDERCHANGE	Einkäufer
DISPATCHNOTIFICATION	Verkäufer
RECEIPTACKNOWLEDGEMENT	Einkäufer
INVOICE	Verkäufer

Tabelle 4.2 Sender der Geschäftsdokumente

Änderungen der Abhängigkeiten durch eine potenzielle Prozessbeschreibung sind immer nur aufbauend möglich. Das bedeutet, dass keine Abhängigkeiten übergangen werden, sondern nur zusätzliche eingefügt werden dürfen. Dazu könnte ein weiterer Regelsatz definiert werden, zum Beispiel die Erlaubnis eine RECEIPTACKNOLEDGEMENT von einer DISPATCHNOTIFICATION abhängig zu machen, eine INVOICE aber nicht von einer RFQ. Solche Regeln sind in der Praxis vermutlich irrelevant, da Geschäftspartner so wenig Abhängigkeiten wie möglich definieren werden, um mit sovielen Geschäftspartnern wie möglich interagieren zu können. Daher wird in dieser Arbeit darauf verzichtet.

Kardinalitäten

In Abbildung 4.4 ist schon der Hinweis darauf enthalten, dass eine ORDERCHANGE mehrmals gesendet werden kann. (Zu sehen ist das an den rückläufigen Übergangspfeilen von „nach" der Ausführung einer ORDERCHANGE zu „vor" der Ausführung einer ORDERCHANGE) Dabei handelt es sich nicht um Rückschritte innerhalb des Ablaufs, sondern die Möglichkeit des Sendens einer neuen ORDERCHANGE.

Hier wird nun festgelegt, wie oft ein jeweiliges Geschäftsdokument in einer Prozessinstanz vokommen darf.

Im Gegensatz zu den Abhängigkeiten, die eine Minimalanforderung an die Strukturierung einer potenziellen Prozessbeschreibung darstellen, geben die Kardinalitäten den Rahmen vor, wie oft ein Dokument mindestens und höchstens gesendet bzw. empfangen werden kann.

Die Kardinalität abhängiger Dokumente wird von ihrem Abhängigkeitsziel bestimmt. Eine QUOTATION kann mehr als einmal gesendet werden, aber nur wenn ihr eine

RFQ vorausging. Daher ist es sinnvoll die Kardinalität abhängiger Dokumente gebunden an ihr Abhängigkeitsziel anzugeben.

Es bedarf eines Kontrollflusskonstrukts in der PDL wie auch der PEL, um die Struktur von WENN RFQ, DANN QUOTATION abzubilden. Dieses Konstrukt wird bei den von ORDER abhängigen Dokumenten nicht benötigt, da eine ORDER genau einmal vorkommen muss. Ein beispielsweise WENN ORDER, DANN INVOICE ist somit obligatorisch in der Prozessstruktur enthalten.

Es lassen sich folgende Kardinalitäten festlegen:

1. RFQ kann kein-, ein- oder mehrmals gesendet werden.
2. Auf eine RFQ folgt genau eine QUOTATION.
3. Eine ORDER wird genau einmal gesendet.
4. Eine ORDERCHANGE kann kein- bis n-mal gesendet werden, n>0.
5. Eine zur ORDER oder ORDERCHANGE gehörige ORDERRESPONSE wird kein- oder einmal gesendet.
6. Eine DISPATCHNOTIFICATION wird kein- oder einmal versendet.
7. Eine RECEIPTACKNOWLEDGMENT wird kein- oder einmal versendet.
8. Eine INVOICE wird kein- oder einmal versendet.

Mit Hilfe der Kardinalitäten kann ein Geschäftpartner auch festlegen, welche Dokumente er

1. in jedem Fall senden wird (Kardinalität n-m, mit m ≥n>0),
2. möglicherweise sendet (Kardinalität 0-m, mit m >0) und
3. nicht sendet (Kardinalität 0),

beziehungsweise, welche Dokumente er

4. nicht empfangen kann (Kardinalität 0),
5. empfangen kann (Kardinalität 0-m, mit m>0) und
6. unbedingt empfangen muss (Kardinalität n-m, mit m ≥n>0).

n-m Kardinalitätsfestlegungen mit m ≥n>1, machen bei den openTRANS-Geschäftsdokumenten kaum Sinn. RFQ und ORDERCHANGE und deren abhängige Dokumente sind die einzigen Dokumente, die öfter als einmal vorkommen dürfen. Es ist unsinnig, festzulegen, dass eine RFQ öfter als einmal gesendet werden muss, sowie es auch nicht schlüssig ist, weshalb eine ORDERCHANGE öfter als einmal gesendet werden muss, da ORDERCHANGE seiner Verwendung nach ein optionales Dokument ist. Es mag Spezialfälle geben, bei denen eine solche Regel notwendig erscheint. Diese werden hier aber vernachlässigt.

Es ergeben sich folgende Wahlmöglichkeiten für die Kardinalitäten der Geschäftsdokumente, aus denen ein Geschäftspartner seine potenzielle Prozessbeschreibung aufbauen kann.

Geschäftsdokument	Kardinalität
RFQ	0, 0-n, 1, 1-n
QUOTATION	0, 0-1, 1
ORDER	1
ORDERCHANGE	0, 0-n
ORDERRESPONSE	0, 0-1, 1
DISPATCHNOTIFICATION	0, 0-1, 1
RECEIPTACKNOWLEDGEMENT	0, 0-1, 1
INVOICE	0, 0-1, 1

Tabelle 4.3 Kardinalitäten der Geschäftsdokumente

Folgende Kardinalitäten können also vorkommen:

- 0
- 0-n, mit n>0
- 1
- 1-n, mit n>1

Die Kardinalitäten der Geschäftsdokumente, die ein Geschäftspartner für seine PPD aussucht, können nur in einem Bereich variieren, der für sie in Tabelle 4.3 festgelegt ist. Weitergehende Erweiterungen der Kardinalitäten eines Geschäftsdokuments sind nicht erlaubt.

Kardinalitätsbeziehungen

Die festgelegten Kardinalitäten für das Senden bzw. Empfangen eines Dokuments müssen für einen komponierten Prozess aufeinander abgestimmt werden.

Dabei ergeben sich zahlreiche Kombinationen. Eine Nachricht wird gesendet, aber nicht empfangen, oder sie wird nicht gesendet, muss aber empfangen werden usw. Die folgende Tabelle 4.4 stellt die denkbaren Beziehungen dar. Ist die Kompositionierbarkeit „OK“ ist eine Umsetzung in eine EPD ohne weiteres möglich. Ein „ERROR“ zeigt an, dass diese Beziehung nicht umgesetzt werden kann. Ein „OK (ERROR ?)“ besagt, dass grundsätzlich eine Umsetzung erreicht werden kann, gesendete Geschäftsdokumente jedoch verloren gehen können.

Sendungs Kardinalität	Empfangs-Kardinalität	Beschreibung	Kompositionier-barkeit
0	0	WIRD NICHT gesendet und empfangen :	OK
0	0-n, mit n>0	WIRD NICHT gesendet werden, und MUSS NICHT empfangen werden.	OK
0	1-m, mit m>0	WIRD NICHT gesendet, aber MUSS empfangen werden.	ERROR
0-n, mit n>0	0	WIRD VIELEICHT gesendet, aber WIRD NICHT empfangen	OK (ERROR ?)
0-n, mit n>0	0-n, mit n>0	WIRD VIELEICHT gesendet, und MUSS NICHT empfangen werden.	OK (ERROR ?)
0-n, mit n>0	1-m, mit m>0	WIRD VIELEICHT gesendet, aber MUSS empfangen werden	ERROR
1-m, mit m>0	0	WIRD gesendet, aber WIRD nicht empfangen	OK, (ERROR ?)
1-m, mit m>0	0-n, mit n>0	WIRD gesendet, und MUSS NICHT empfangen werden.	OK, (ERROR ?)
1-m, mit m>0	1-m, mit m>0	WIRD gesendet und MUSS empfangen werden.	OK, (ERROR ?)

Tabelle 4.4 Kardinalitätsbeziehungen der Geschäftsdokumente

Hier ist zu erkennen, dass eine Kompositionierbarkeit von Kardinalitäten genau dann nicht erreicht werden kann (bei allen ERRORs), wenn der Empfänger den Eingang eines Geschäftsdokuments erwartet, aber der Sender niemals sendet. In diesem Fall kann keine Übereinstimmung gefunden werden, da der Geschäftspartner das Dokument benötigt, um den Prozess fortzusetzen.

Die Kardinalitätsbeziehungen, die in Tabelle 4.4 mit „OK“ bewertet wurden sind trivial, da sie in einer komponierten Prozessbeschreibung nicht umgesetzt werden.

Kompensation für verlorengegangene Nachrichten

Bei der Bewertung in Tabelle 4.4 der Kompositionierbarkeit durch „OK (ERROR ?)“ ist zu beachten, dass zwar eine Komposition umgesetzt werden kann, es dabei aber zu Beschränkungen bei einem der Geschäftspartner kommen kann.

So erlaubt ein Verkäufer beispielsweise nur eine RFQ, ein Einkäufer will aber unbegrenzt viele versenden. Eine Komposition kann hier erreicht werden, aber nur unter den minimalen Bedingungen des Verkäufers.

Hier muss die PPD die Möglichkeit bieten, dass derjenige, der eine Kardinalität über der minimalen (z.B.: sende bis zu 10 RFQs statt 0-1) verlangt, eine Kompensationsmöglichkeit anbietet, welche der ausführende Prozess ausführt, wenn ein Zustand erreicht wird, den der Geschäftspartner nicht wünscht.

Dies könnte den Einkäufer betreffen, der seine zweite RFQ sendet, obwohl der Verkäufer nur eine empfangen kann. In diesem Fall könnte der Einkäufer angegeben haben, dass der Prozess eine Operation seines Web Service ruft, die ihn darüber informiert, dass die 2.RFQ nicht gesendet wird.

Nicht nur bei zu vielen, sondern auch bei zu wenig verschickten Geschäftsdokumente, kann eine Kompensation nötig sein. So will ein Geschäftspartner nicht unbegrenzt lange auf das Eintreffen eines Geschäftsdokuments warten und muss damit auch die Möglichkeit besitzen, Timeouts für seine Operationen festzulegen. Aktivitäten mit MUSS-Status müssen einen Timeout besitzen. Wenn sie nicht eintreffen, wird der Prozess automatisch beendet.

Kompensationen können theoretisch beliebig komplex werden und die verschiedensten Operationen mit den unterschiedlichsten Parametern rufen. Für dieses Lösungskonzept soll es aber genügen, dass eine Kompensation dem Aufruf einer bestimmten Web Service-Operation des Geschäftspartner gleichkommt, der die Kompensation anbietet.

Damit ist für die EPD eine Kompensation für verlorengegangene Nachrichten nichts weiter als der Aufruf einer Operation, wenn eine bestimmte Kardinalität überschritten wird. Durch das schon erwähnte WENN-DANN Konstrukt kann diese Anforderungen ebenfalls umgesetzt werden.

KANN, WERDE, MUSS Status von Geschäftsdokumenten

Empfangserwartungen eines Geschäftsdokuments mit einer Kardinalität von 1 oder höher sind immer verpflichtend für den Sender. Sie müssen von ihm gesendet werden, da der Prozess sonst nicht fortgesetzt werden kann: MUSS-Status eines Geschäftsdokuments.

Andererseits ist das Empfangen eines gesendeten Dokuments mit einer Kardinalität größer 1 nicht verpflichtend. Es kann vom ausführbaren Prozess empfangen, aber dann ohne Weitersenden verworfen werden: WERDE(-Senden)-Status eines Geschäftsdokuments.

Möchte ein Sender allerdings, dass sein Dokument in jedem Fall empfangen wird, soll also beispielsweise der Verkäufer RFQs und ORDERCHANGEs empfangen können, muss der Einkäufer dies explizit in der PPD festlegen können. Er fügt seinem WERDE-(VIELLEICHT)-Senden-Geschäftsdokument den Status MUSS-Empfangen-werden hinzu.

Zusammenfassend bedeutet dies, dass die Geschäftspartner einem Dokument je nach Kardinalität den Status

- KANN NICHT-Senden (Kard. 0),
- WERDE VIELLEICHT-Senden (Kard. 0-n) oder

- WERDE-Senden (ab Kard. 1-n)

vergeben können muss, sowie zusätzlich den Status

- MUSS-oder
- MUSS NICHT-Empfangen werden.

Die Beziehungen der Geschäftsdokumente, die durch die Kardinalitäten schon umfangreich waren, werden hier noch komplexer.

Es kann beispielsweise sein, dass eine RFQ mit MUSS Status beim Einkäufers und mit MUSS-Status beim Verkäufers deklariert sind. Dennoch wird hier eine Kompensation nur erreicht, wenn zusätzlich die Kardinalitätsbeziehung der Geschäftsdokumente stimmig ist.

In Abbildung 4.5 ist eine Komposition nicht möglich, obwohl beide Geschäftspartner das Dokument RFQ unterstützen. Der Einkäufer verlangt, dass eine RFQ empfangen wird. Diese Anforderung wird erfüllt. Der Verkäufer verlangt, dass eine RFQ in jedem Fall gesendet wird. Diese Anforderung unterstützt der Einkäufer nicht, da er für das Senden einer RFQ eine Kardinalität von 0-n festgelegt hat und daher unter Umständen keine RFQ sendet.

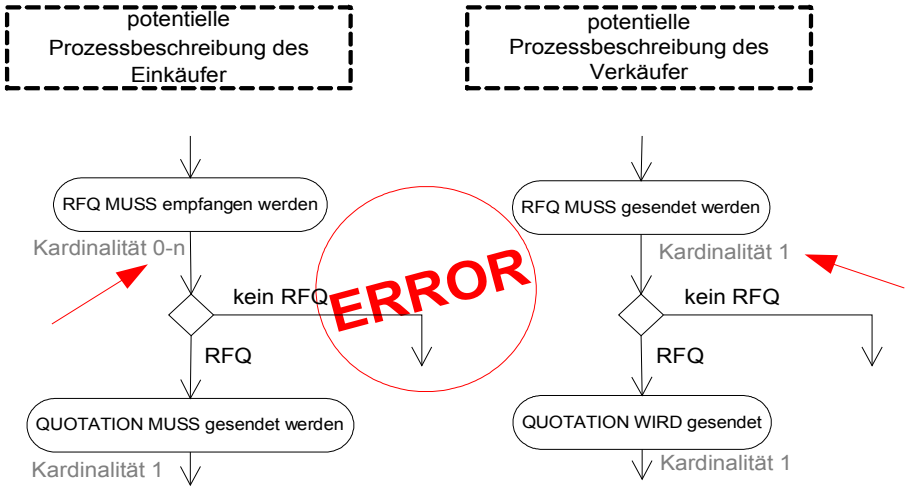

Abbildung 4.5 Statusbeziehungen 2

In Abbildung 4.6 hingegen ist eine Komposition möglich, da die Wünsche des Einkäufers mit dem Angebot des Verkäufers übereinstimmen: Eine RFQ MUSS vom Verkäufer angenommen werden können, und der Verkäufer kann RFQs empfangen.

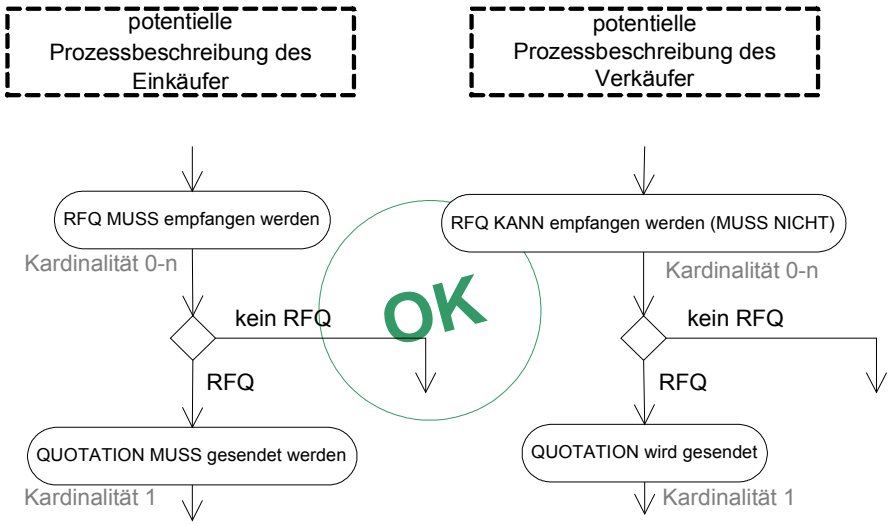

Abbildung 4.6 Statusbeziehungen 1

Ersetzung eines MUSS Geschäftsdokuments

Findet sich zu MUSS-Status-Geschäftsdokumenten beim Gegenüber des Geschäftspartner keine Entsprechung, kommt eine EPD nicht zu Stande.

Die Geschäfspartner müssen hier Ausnahmen definieren können. So kann ein Einkäufer beispielsweise definieren, dass er eine ORDERRESPONSE erhalten MUSS. Diese kann aber entfallen, wenn auf jeden Fall eine INVOICE gesendet wird. Um diesen Bezug abzubilden, muss ein Konzept vorgesehen werden, welches die mögliche Ersetzung von Operationen vorsieht.

Mit Hilfe dieser Anforderung können Geschäftspartner die Kompositionsfähigkeit ihrer Prozessbeschreibungen vergrößern.

Beenden des Interaktionsablaufs und Timeouts

Ein openTRANS-Interaktionsverlauf sieht keine Möglichkeit vor, eine zweite ORDER zu schicken, wenn beispielsweise die erste vom Verkäufer abgelehnt wurde. Der Einkäufer müsste für eine neue Bestellung eine neue Instanz des Prozesses starten.

Damit die erste Instanz nicht unendlich weiterläuft, müsste sie entweder explizit oder nach einer gewissen Zeit automatisch beendet werden. Da alle Empfangs-Aktivitäten eines Prozesses auch Timeouts definieren, ist dieser Zeitpunkt automatisch erreicht, wenn ein MUSS-Dokument nicht eintrifft. Für dieses Lösungskonzept soll das Auslaufen eines Interaktionsablaufs ausreichend sein.

Sequentiell oder Parallel

Paralleles Senden und Empfangen von openTRANS-Geschäftsdokumenten bedeutet nicht ihre zwangsläufige Gleichzeitigkeit, sondern dass es gleichgültig ist, welches der Dokumente als erstes vom Geschäftspartner gesendet oder empfangen wird. Sequentielles Senden und Empfangen bedeutet, dass zwei Dokumente in der festgelegten Reihenfolge übermittelt werden müssen.

DISPATCHNOTIFICATION und INVOICE sind die einzigen Geschäftsdokumente, die laut dem Interaktionsablauf in Abbildung 4.4 parallel ausgeführt werden können.

Anforderungen an die EPD

Es ergeben sich folgende zusätzliche Anforderungen die ausführbare Prozessbeschreibung:

1. Sie muss WENN-DANN-Beziehungen von Aktivitäten ausdrücken können.
   Bsp.: WENN ich eine RFQ sende, DANN will ich eine QUOTATION empfangen.
2. Sie muss die Möglichkeit bieten, Reihenfolgen von Aktivitäten auszudrücken.
   Bsp.: Erst RFQ, danach QUOTATION, schließlich ORDER.
3. Nachrichten müssen auch parallel geschickt werden können.
   Bsp.: Erst ORDER, dann erst INVOICE und danach DISPATCHNOTIFICATION oder dann erst DISPATCHNOTIFICATION und dann INVOICE.
4. Die Bedingung von Timeouts muss umgesetzt werden können.
5. Sie muss die Möglichkeit bieten, Kardinalitäten von Aktivitäten auszudrücken.
   Bsp.: Eine RFQ darf null bis unbegrenzt oft gesendet werden, eine ORDER nur einmal.
6. Eine TERMINATE-Aktivität muss vorhanden sein.

Anforderungen an die PPD

Eine PPD muss die o.g. und einige zusätzliche Bedingungen erfüllen:

1. Sie muss WENN-DANN-Beziehungen von Aktivitäten ausdrücken können.
   Bsp.: WENN ich eine RFQ sende, DANN will ich eine QUOTATION empfangen
2. Sie muss die Möglichkeit bieten, Reihenfolgen von Aktivitäten anzugeben.
   Bsp.: Erst RFQ, danach QUOTATION, danach ORDER.
3. Nachrichten müssen auch parallel geschickt werden können.
   Bsp.: Erst ORDER, dann erst INVOICE und danach DISPATCHNOTIFICATION oder dann erst DISPATCHNOTIFICATION und dann INVOICE.
4. Sie muss die Möglichkeit bieten, Kardinalitäten von einzelnen Aktivitäten auszudrücken.
   Bsp.: Eine RFQ darf null bis unbegrenzt oft gesendet werden, eine ORDER nur einmal.
5. Sie muss die Möglichkeit bieten, Kompensationen für eine nicht ausführbare Aktivität anzugeben.
6. Es muss möglich sein Timeouts für den Empfang bzw. das Senden eines Geschäftsdokuments angeben zu können.
7. Sie muss die Definition verpflichtender Aktivitäten erlauben. Ist eine Operation ein MUSS, oder KANN sie auch weggelassen werden?
   Bsp.: Eine ORDERCHANGE MUSS vom Geschäftspartner empfangen werden.

8. Sie muss erlauben verpflichtende Aktivitäten durch andere zu ersetzen.
   Bsp.: Eine ORDERRESPONSE MUSS nicht gesendet werden, wenn eine INVOICE vom Verkäufer in jedem Fall versendet wird.

4.4.4 Process Integrator

Der Process Integrator (PI) ist die verwaltende Komponente des Integrationssystems. Er hat die Aufgabe, die eingereichten PPDs zur Generierung der EPDs weiterzureichen und die Komposition, Transformation und Installation zu überwachen, sowie den Einkäufer über den Zustand der Integration zu informieren.

Der PI entscheidet zudem anhand der eingehenden PPDs, ob eine Komposition und Transformation überhaupt notwendig sind oder schon durchgeführt wurden. Sind die PPDs bereits komponiert worden, prüft er, ob eine transformierte Fassung der komponierten Prozessbeschreibung (CPD) vorliegt, und ob der entsprechende Flow-Service noch auf der Process Engine installiert ist. Wenn nötig, stößt er die Transformation und Installation erneut an und gibt die Adresse des Flow-Service an den Einkäufer zurück, damit dieser den Prozess starten kann. Für diese Funktionalität benötigt der PI ein Caching von bereits komponierten und transfomierten Prozessbeschreibungen sowie er Informationen der PE über installierte Prozesse benötigt.

4.4.5 Process Composer

Grundsätzlich soll die Komponente erst einmal nur openTRANS-Prozessbeschreibungen komponieren können. Ein generischer Ansatz soll dabei aber nicht aus den Augen verloren werden, um eine mögliche Erweiterung des Systems zu erlauben.

Die Komponente ist stark von der gefundenen Technologie zur Erstellung der potentiellen Prozessbeschreibung (der PDL) abhängig, die sie analysieren und verarbeiten können muss.

Zusätzlich zu den Prozessbeschreibungen der Geschäftspartner muss es möglich sein eine Metaprozessbeschreibung einzulesen, anhand derer die Gültigkeit der beiden Prozessbeschreibungen überprüft werden kann.

Die vom Process Composer erstellte komponierte Prozessbeschreibung (CPD) soll aus Verständnisgründen mit der gleichen semantischen Technologie und im gleichen Format erfolgen, auf der auch die potenziellen Prozessbeschreibungen beruhen. Dabei werden "mögliche" Operationen, Fehlerbehandlungen und Abhängigkeiten, die in den potenziellen Prozessbeschreibung vorhanden sind, umgewandelt in "reale" Prozessbeschreibungen.

4.4.6 Process Transformer

Der Process Transformer (PT) wandelt die in der PDL erstellte komponierte Prozessbeschreibung (CPD), in eine ausführbare Prozessbeschreibung (EPD) um, die auf

einer Process Execution Language (PEL) basiert. Diese Komponente sollte die Prozesskomposition grundsätzlich in jede mögliche PEL erlauben. Das bedeutet, dass die Komponenten, die der Umwandlung dienen, architektonisch so im Process Transfomer eingebettet sein sollen, dass ein Austausch leicht möglich ist.

Eine übergreifendes Interface muss hier definiert sein, über das der Aufbau einer EPD geregelt wird. Ein solches Interface stellt dann Methoden wie startSequence und endSequence oder addWSDLOperation bereit, mit denen eine EPD systematisch aufgebaut werden kann.

4.4.7 Prozess Engine

Auf einer Process Engine wird die Installation und die Ausführung einer ausführbaren Prozessbeschreibung durchgeführt, die in einer bestimmten Process Execution Language (PEL) geschrieben ist. Als Process Engine (PE) soll keine Eigenentwicklung dienen. Grundsätzlich sollen im Integrationssystem eine Nutzung aller Engines und aller PELs erlaubt sein, da der PT eine mögliche Umwandlung in jede PEL erlauben soll.

Sitzungs-Management

Durch die standardisierte Form der openTRANS-Geschäftsdokumente befinden sich die Identifikationsdaten eines Dokuments immer an der gleichen Stelle (z.B. RFQ_ID im RFQ_HEADER), und auch die Bezüge zu der erfolgten Bestellung können in den Geschäftsdokumenten gefunden werden. So bezieht sich das Feld ORDER_REFERENCE z.B. im INVOICE-Header des INVOICE-Geschäftsdokuments auf die vorausgegangene ORDER (Feld ORDER_ID). So könnte mit der Definition von CorrelationSets in BPEL4WS der Bezug der einzelnen Dokumente zueinander und damit ihre Zugehörigkeit zu einer laufenden Instanz bzw. Sitzung eines Interaktionsablaufs auf der PE hergestellt werden.

Dadurch wären die Sitzungsdaten allerdings abhängig von den Geschäftsdokumenten. Eine Änderung der Geschäftsdokumente würde dann auch eine Änderung der Prozessbeschreibungen nach sich ziehen müssen. Eine unabhängige Variante, die die Sitzungsinformationen im Header speichert wäre hier von Vorteil. WS-Addressing ist ein dafür geeigneter Mechansimus.

Für dieses Lösungskonzept soll eine der beiden Varianten für das Sitzungs-Management genügen. Setzt die Process Engine beispielsweise WS-Addressing ein, ist das ausreichend.

Anforderungen im Überblick

Die PE wird nach folgenden Kriterien bewertet:

1. Die Prozessbeschreibung in der PEL muss dynamisch und ohne menschliche Interaktion an die Engine übergeben werden können.
2. Die PE sollte bereits ausreichend getestet und/oder im Einsatz sein, um Fehler in dieser Komponente, die nicht auf Eigenentwicklung beruhen, zu minimieren.

3. Sie sollte wenigstens in ihrer Anwendbarkeit frei verfügbar sein, bestenfalls Open Source, um Schwierigkeiten mit Trial Versionen oder dem Einkauf von Software zu vermeiden.
4. Die PE muss für eine Weiterentwicklung sowohl die Fähigkeit zum Sitzungs- und Transaktionsmanagement besitzen. Für dieses Lösungskonzept reicht Sitzungsfähigkeit aus.
5. Audit Trail: Die abgelaufenen Prozesse müssen zurückverfolgbar sein. Für dieses Lösungskonzept ist dies kein notwendiges Kriterium.
6. Weiterhin sollten eine fehlerfreie und einfache Erstellung, Überprüfung und Kontrolle der Prozesssteuerung gegeben sein.

4.5 Auswahl der Technologien

Im Folgenden werden die Technologien ausgewählt, die für die Umsetzung des Lösungskonzepts nötig sind.

4.5.1 Process Description Language (PDL)

Als PDL sollen Technologien des Semantic Web, wie RDF und darauf aufbauende Ontologien genutzt werden. Auf diese Weise kann mit einem Streich die Prozessbeschreibung und die Klärung der Semantik der beschriebenen Web Service Schnittstellen abgehandelt werden, wie in Kapitel 4.2 bereits angesprochen.

Die in Kapitel 3.3.2 vorgestellte Technologie OWL-S ist momentan die einzige Ontologie zur Beschreibung von Web Services und deren Prozessen. Da Ontologien leicht zu erweitern sind, ist es sinnvoll auf dieser Ontologie aufzubauen. Dennoch muss überprüft werden, inwieweit OWL-S die Anforderungen an die PDL erfüllt, um festzustellen an welchen Stellen OWL-S erweitert werden muss.

OWL-S

Wichtig für die Umsetzung einer potenziellen Prozessbeschreibung sind die beiden Teilontologien

- Process.owl
- ServiceGrounding.owl

In der Process.owl sind Begriffe für die Strukturierung eines Prozesses und ihre Beziehungen zueinander enthalten, die für eine Prozessbeschreibung instanziiert werden können.

In der ServiceGrounding.owl werden Begriffe bereitgestellt, mit denen die Beziehungen der Aktivitäten aus einer Prozessbeschreibung zu Operationen von einem oder mehreren Web Services über WSDL-Dokumente beschrieben werden können.

Die Übertragung eines Geschäftsdokuments entspricht dem Aufruf einer Operation des Web Service eines Geschäftspartners. Eine solcher Operationsaufruf wird in der

Process.owl als `AtomicProcess` bezeichnet. Eine Komposition von mehreren `AtomicProcesses` ist ein `CompositeProcess`. Beide sind Subklassen der `Process`-Klasse.

Es existieren keine impliziten Konstrukte in der Process.owl, die es erlauben Kompensation von ungültigen Zuständen zu definieren. Hier muss die Ontologie erweitert werden.

Die Deklaration von Timeouts ist allerdings implizit möglich, da die Process.owl sich mit ihren Hauptklassen (`Process` und `ControlConstruct`) auf eine Ontologie der Zeit bezieht. Um einem Timeout eine Kompensation hinzuzufügen, ist allerdings eine Erweiterung nötig.

Die Zusammenarbeit von verschiedenen Parteien lässt sich nicht explizit ausdrücken. Allerdings besteht die Möglichkeit in einem ServiceGrounding für verschiedene Operationen verschiedene WSDLs verschiedener Parteien zu referenzieren.

Die Anforderungen von Neutralität und Erweiterbarkeit erfüllt OWL-S ohne Vorbehalte. Beim derzeitigen Entwicklungsstand sind zwar noch einige Fragen offen, und vorgesehene Teilaspekte von OWL-S sind noch nicht spezifiziert, aber für die hier umzusetzende Anwendung ist OWL-S genügend weit entwickelt.

Die gewünschten Kontrollflusskonstrukte werden alle erbracht. Es gibt ein `If-Then-Else` Konstrukt, ein `sequence` Konstrukt, eine `unordered` Konstrukt für parallele Ausführung von Aktivitäten sind in der Process.owl vorhanden.

Kardinalitäten von Atomic Process können nicht explizit angegeben werden. Ein entsprechendes `Repeate-While` - `ControlConstruct` bezieht sich auf die tatsächliche Ausführungshäufigkeit eines `Process`, und nicht auf seine potenziell mögliche, bzw. verlangte.

MUSS-KANN-Aktivitäten können ebenso nicht festgelegt werden, und daher auch keine ersetzenden MUSS-Aktivitäten.

Im Folgenden werden die übergreifenden und die beschreibenden Anforderungen in Tabellenform aufgezeigt (Tabellen 4.5 und 4.6):

übergreifende Anforderungen an die PDL	**erfüllt**[26]
Zusammenarbeit verschiedener Parteien und deren Rollen	◉
Eignung für die Beschreibung von Web Services	●
Neutralität	●
Entwicklungsstand und Erweiterbarkeit hinsichtlich der gewünschten Funktionalität	●
einfach anwendbar	●
semantische Technologie	●

Tabelle 4.5 PDL - übergreifende Anforderungen und OWLS

beschreibende Anforderungen an die PDL	**Umsetzung möglich**[27]
Kompensation	○
Timeout	◉
Kontrollfluss	●
WENN-DANN Konstrukt (openTRANS - Anforderung)	●
Sequentielle Aktivitäten (openTRANS - Anforderung)	●
Parallele Aktivitäten (openTRANS - Anforderung)	●
Kardinalitäten von Aktivitäten (openTRANS - Anforderung)	○
MUSS - KANN Aktivitäten (openTRANS - Anforderung)	○
Ersetzen einer MUSS-Aktivität	○

Tabelle 4.6 PDL - beschreibende Anforderungen und OWLS

Die nötigen Erweiterungen und Anpassungen für die Umsetzung der beschreibenden Anforderungen werden im Systementwurf noch beschrieben.

[26] Bedeutung:
○ = nicht erfüllt
◉ = teilweise erfüllt
● = erfüllt

[27] Bedeutung:
○ = nicht erfüllt
◉ = teilweise erfüllt
● = erfüllt

4.5.2 Process Execution Language (PEL)

Die Auswahl von der Process Engine und der PEL sind eng miteinander verknüpft.

Als Grundvoraussetzung für die Auswahl der PEL bestehen die oben genannten Anforderungen, an die Umsetzbarkeit einer openTRANS-Prozessbeschreibung und eine für die PEL implementierte Process Engine.

Zwei Ausführungssprachen, ebBPSS und BPEL4WS, wurden in Kapitel 3.4 erläutert. Ihre Eignung für die Beschreibung einer ausführbaren openTRANS-Prozessbeschreibung wird nun untersucht.

ebBPSS

Verschiedene zusammenarbeitende Parteien lassen sich in ebBPSS sehr gut darstellen. Ebenso ist ebBPSS für das Transaktions-Management und Fehlerbehandlungen bestens ausgestattet. Konstrukte wie `Success` und `Failure` sind für diese Aufgaben entwickelt worden.

Wie schon beschrieben kommt ebBPSS mit einigen wenigen Kontrollflusskonstrukten aus, mit denen sich ein Kontrollfluss erstellen lässt. Eine Web Service Operation entspricht in ebBPSS einer `BusinessTransactionActivity`. `Transitions` (Übergänge) regeln die Bestimmungen für den Übergang von einer `BusinessTransactionActivity` zur nächsten. Damit lassen sich WENN-DANN Beziehungen sehr gut ausdrücken. Auch die Reihenfolge von `Business-TransactionActivities` lassen sich auf diese Weise festlgegen, ebenso können auch parallele Aktivitäten zugelassen werden. Kardinalitäten, das bedeutet die Definition von Schleifen ist damit auch umsetzbar. Der Übergang einer Aktivität kann wieder zu der gleichen Aktivität führen, bis bestimmte Bedingungen erfüllt werden.

Problematisch ist dabei, dass es kein direktes Konstrukt für Schleifen in ebBPSS gibt, und sie sich erst bei genauer Analyse des Dokuments erkennen lassen. Das liegt an der Tatsache, das die eigentlichen Aktivitäten und ihre Übergänge voneinander getrennt gehalten werden.

BPEL4WS

In BPEL4WS gibt es ebenso ein etabliertes Konzept um die Zusammenarbeit verschiedener Parteien zu ermöglichen: `PartnerLinks` und `PartnerLinkTypes`. Dabei wird auf die PortTypes von WSDL Beschreibungen zurückgegriffen, anhand derer eine Partei seine Erwartungen an die Rolle des Gegenübers definiert. Die in einem PortType definierten Web Service Operationen werden in der Prozessbeschreibung durch Primitve Activities genutzt.

Für das Transaktions-Management und die Fehlerbehandlung stehen mit `Compen-sationHandlers` und `FaultHandlers` gute Umsetzungsmöglichkeiten bereit. Sie funktionieren ähnlich den `try-catch` Scopes in Java.

Timeouts lassen sich über ein `onAlarm` Element definieren.

BPEL4WS wurde speziell für Web Services entwickelt.

Es gibt kein WENN-DANN Kontrollflusskonstrukt. Dieses kann aber mit der einem `pick-onMessage` umgesetzt werden. Wenn ein RFQ ankommt, wird danach auf ein QUOTATION gewartet, bevor der Prozess weitergehen kann.

Sequentielle und parallele Darstellung werden mit `sequence` und `flow` Elementen beschrieben. Kardinalitäten können mit dem Kontrollflusskonstrukt `Repeate-While` beschrieben werden.

Auswahl

Hier nocheinmal die Umsetzungsmöglichkeiten der Anforderungen beider Sprachen im Überblick:

Anforderung	**in ebBPSS**	**in BPEL4WS[28]**
Zusammenarbeit verschiedener Parteien	●	●
Transaktions-Management	●	●
Fehlerbehandlungen	●	●
für kooperierende Web Services geeignet	◘	●
Darstellung eines Kontrollfluss	●	●
Timeout	●	●
WENN-DANN Kontrollflusskonstrukt (openTRANS - Anforderung)	●	●
Sequentielle Aktivitäten (openTRANS - Anforderung)	●	●
Parallele Aktivitäten (openTRANS - Anforderung)	●	●
Kardinalitäten von Aktivitäten (openTRANS - Anforderung)	◘	●

Tabelle 4.7 PEL - Anforderungen an eBPSS und BPEL4WS

Der primäre Unterschied zwischen BPEL4WS und ebBPSS sind die verschiedenen Ansätze den Kontrollfluss darzustellen. In BPEL4WS werden Aktivitäten in einzelnen Blöcken oder Scopes strukturiert (block-orientiert), ähnlich den Scopes einer Programmiersprache. In ebBPSS werden Zustände und Übergänge zwischen diesen definiert (zustand-übergang-orientiert).

[28] Bedeutung:
○ = nicht erfüllt
◘ = teilweise erfüllt
● = erfüllt

Um eine Transformierung von einer PPD in eine EPD zu erleichtern ist es von Vorteil, wenn beide Prozessbeschreibungssprachen entweder block-orientiert oder zustand-übergang-orientiert aufgebaut sind. Da als PDL eine block-orientierte Sprache gewählt wurde, ist BPEL4WS besser geignet.

Zudem erfüllt BPEL4WS das Kriterium der Eignung für Web Services implizit, und muss nicht wie bei ebBPSS explizit angepasst werden.

Für ebBPSS sind Trial Versionen für verschiedene Process Engines verfügbar wie zum Beispiel BUSINESSWARE 4 von VITRIA, der auch die Integration von Web Services unterstützt. Allerdings ist das „Wie" einer Umsetzung für Web Services von System zu System verschieden, da keine direkten Verknüpfungen mit der WSDL Beschreibung eines Web Services in ebBPSS definiert werden kann.

Für die Umsetzung von BPEL4WS wurden zwei funktionierende Process Engines gefunden. Der BPWS4J Engine 2.0 von IBM und der Collaxa BPEL Server 2.0 rc5.

ebBPSS ist als Teil der allgemein anerkannten ebXML Spezifikation auf dem besten Wege zu einer weit verbreiteten Nutzung. Allerdings ist auch unter Hinblick auf die Standardisierungsbemühungen von IBM und Partnern für BPEL4WS zu erwarten, dass diese PEL noch auf längere Sicht unterstützt und gefördert wird.

Da eine BPEL4WS Prozessbeschreibung von einem System problemlos auf ein anderes übertragen lässt, sie Vorteile bei der Umsetzung der Kontrollkonstrukte bezüglich der PDL besitzt und eine implizite Verknüpfung mit den hier zu integrierenden Web Services vorhanden ist, wird ihr hier der Vorzug geben.

4.5.3 Process Engine

Getestet wurden

- die BPWS4J Engine 2.0 von IBM, und
- der Collaxa BPEL Server 2.0 rc5 von Collaxa[29]

BPWS4J

Die IBM Business Process Execution Language for Web Services Java Runtime stellt eine Plattform bereit, auf der Business Prozesse, die in BPEL4WS beschrieben wurden, ausgeführt werden können. Die aktuelle Version 2.0 unterstützt die BPEL4WS v1.1 (März 2003) Spezifikation.

Die Engine ist in Java implementiert und wird als .jar-Datei ausgegeben. Er kann einfach als Webanwendung auf einem Java Servlet Container wie zum Beispiel TOMCAT[30] deployed werden.

29 http://ww.collaxa.com
30 http://jakarta.apache.org/tomcat/index.html

Danach steht eine Web-Administrationsoberfläche bereit (siehe Abbildung 4.7), die die rudimentärsten Funktionalitäten zur Verfügung stellt.

Abbildung 4.7 BPEL4J Administrationsoberfläche

Hier können die aktuell installierten Prozesse angesehen (List), vorhandene deinstalliert (Un-Deploy) und neue installiert (Deploy) werden.

Für die Installation eines neuen Prozesses werden das WSDL („Process' WSDL file“) und die ausführbare Prozessbeschreibung („BPEL file“) auf der Festplatte gesucht und per HTTP-POST in die Applikation geladen. Das WSDL-Dokument darf allerdings noch keinerlei Bindings enthalten, diese werden erst durch den Server hinzugefügt. Nach Analyse der ausführbaren Prozessbeschreibung verlangt die Engine nach den WSDLs der zu komponierenden Web Services, die den einzelnen PartnerLinks in dem BPEL4WS-Dokument zugeordnet werden.

In der vorliegenden Version ist die Installation eines Prozesses über die Kommandozeile nicht realisiert.

Zudem besitzt er keine Optionen eine Prozessablauf zu verfolgen oder gar zurückzuverfolgen.

Sitzungs-Management kann hier ausschließlich mit CorrelationSets aus BPEL4WS realisiert werden.

Rechtlich darf die BPEL4J Runtime Engine nur zu Bewertungs-/Testzwecken genutzt werden. Allerdings ist der Zeitraum der Bewertung nicht begrenzt. Weder durch die beiliegende Lizenz noch durch eine Inaktivierungseinheit.

Trotz der Eignung aufgrund der freien und unbegrenzten Möglichkeit den BPEL4J für diesen Prototypen zu verwenden, ist die BPLE4J Runtime Engine als Process Engine des openTRANS-Integrationssystems nicht geeignet, da sie das Hauptkriterium zur dynamischen und automatischen Prozessintegration nicht erfüllt.

Collaxa BPEL Server

Weitere OpenSource Implementierungen einer BPEL4WS Engines konnten bis zum Zeitpunkt der Fertigstellung diese Dokuments nicht gefunden werden. Daher musste auf ein kommerzielles Produkt ausgewichen werden.

Gewählt wurde der Release Candiadate 5 für die Version 2.0 des BPEL Servers von Collaxa (Collaxa BPEL Server). Er unterstützt ebenfalls die BPEL4WS v1.1 Spezifikation, allerdings die Variante die im Mai 2003 herausgegeben wurde. Sie unterscheidet sich von der März Version hauptsächlich in einigen Umbenennungen von XML-Elementtypen, allerdings verhindert dies eine einfache Portierung der BPEL4WS Prozessbeschreibungen vom BPEL4J Runtime Engine zum Collaxa BPEL Server.

Der Collaxa BPEL Server ist als 30-tägige Trial Voll-Version verfügbar. Die Lizenz muss nach der Installation und Hochfahren des Servers über dessen Web-Interface angefordert werden. Sie wird per E-Mail im zip-Format zugestellt und muss dann in einen Linzenzordner kopiert werden.

Der Collaxa BPEL Server ist ein eigenständiges Produkt und nicht als Plugin Version wie die BPEL4J Runtime Engine verfügbar. Ihr liegt der JBoss AS[31] als Application Server zugrunde.

Von Collaxa wird zusätzlich ein Plugin für die Java IDE Eclipse[32] bereitgestellt - der BPEL Designer, mit dessen Hilfe Prozesse graphisch modeliert werden können.

Die Administration erfolgt über ein Web-Interface (siehe Abbildung 4.8) wie beim BPEL4J, das um ein vielfaches umfangreicher ist. Die Installation von EPDs kann wahlweise über das Web-Interface oder vom BPEL Designer Eclipse Plugins aus per Mausklick durchgeführt, sowie kommandozeilenbasiert angestoßen werden.

Über das Web-Interface wird dazu eine .jar-Datei benötigt, dass die gleichen Konfigurationsdateien enthält, die auch bei der kommandozeilenbasierten Variante benötigt werden. Vom BPEL Designer aus werden diese Dateien automatisch generiert, wenn die entsprechenden Informationen, bei der Modellierung eingetragen werden.

Bei den benötigten Informationen handelt es sich vor allem um die WSDL-Standorte der zu komponierenden Web Services. Diese müssen in einer bpel.xml Datei angegeben werden. Zusätzlich erlaubt der Collaxa BPEL Server eine einfache Installation von Prozessen mit dem Java basierten Build-Tool Apache Ant[33]. Wird eine weitere in der für Ant üblichen XML-Syntax verfassten Datei namens „build.xml“ bereitgestellt, kann über das einfaches Kommando „cxant“ der Installationsvorgang gestartet werden.

31 http://www.jboss.org/
32 http://www.eclipse.org/
33 http://ant.apache.org/

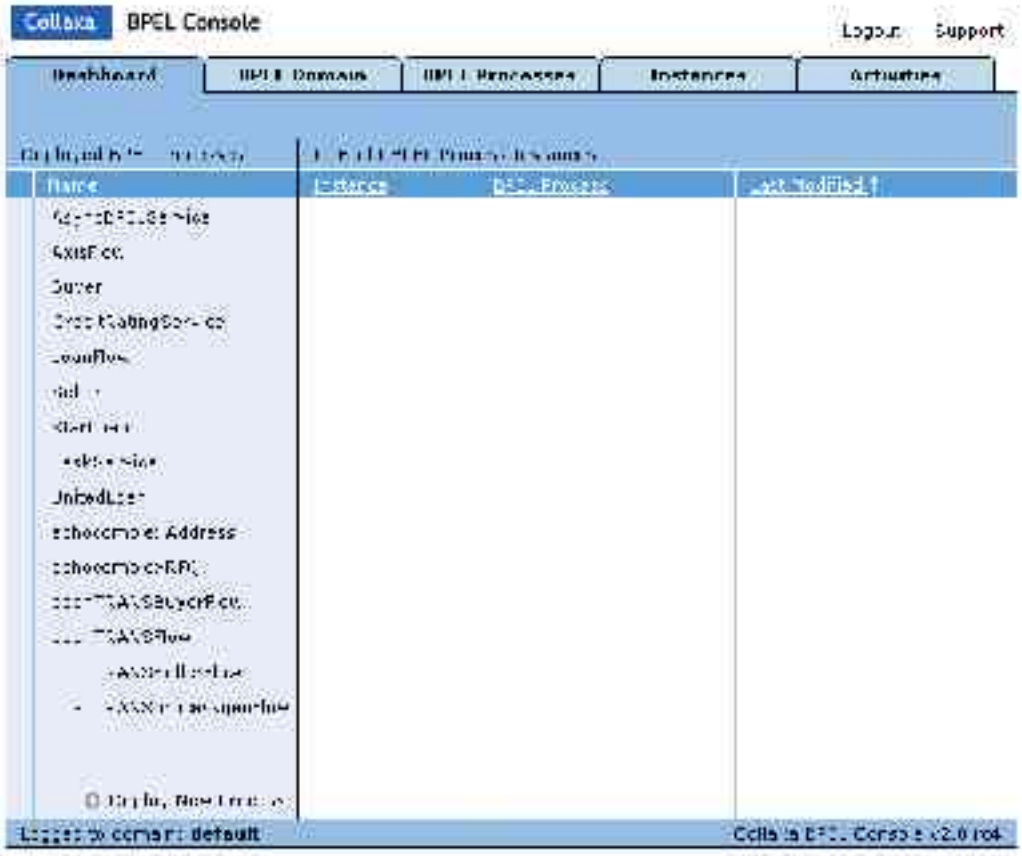

Abbildung 4.8 Collaxa BPEL Server Adminstrationsoberfläche

Der Collaxa Server wird stetig weiterentwickelt und getestet. So durchlief der Collaxa BPEL Server die Release Candidate Stufen 2-6 für die Version 2.0 während der Fertigstellung dieser Arbeit.

Der Collaxa BPEL Server unterstützt neben WS-Transaction, WS-ReliableMessaging, XQuery und XSLT auch die Standardtechnologie WS-Addressing (siehe Kapitel 3.2). Damit lässt sich eine auf SOAP beruhende Addressierung der an einem Prozess beteiligten Web Services vornehmen. BPEL4WS erlaubt mit CorrelationSets das Erkennen von aufeinanderbezogenen Nachrichten. Mit dem Collaxa BPEL Server sind beide Varianten möglich.

Prozesse lassen sich sowohl während des Ablaufs, als auch nach derer Beendigung auf mehrere Arten nachvollziehen, was der Überwachung, der Nachweisbarkeit und der Fehlersuche dient. Es sind eine Audit, sowie eine Debug und eine Flow Ansicht vorhanden, die verschiedene Aspekte des Prozesses hervorheben. Die Flow-Ansicht ist graphisch orientiert und stellt die einzelnen Aktivitäten der EPD als spezielle beschriftete Symbole dar, mit der sich der Ablauf des Prozesses leicht nachvollziehen lässt (siehe Abbildung 4.9).

Abbildung 4.9 Collaxa BPEL Server - Flow Ansicht eines Prozesses

Anforderungen	**Umsetzung in Collaxa**[34]	**Umsetzung in BPWS4J**
dynamische Installation einer EPD	●	○
getestet und/oder im Einsatz	●	●
frei verfügbar oder sogar bestenfalls Open Source	○	○
Sitzungs- wie Transaktionsfähigkeit (Correlationsets und WS-Addressing)	●	◘
zurückverfolgbar	●	○
einfache Erstellung, Überprüfung und Kontrolle der Prozesssteuerung	●	○

Tabelle 4.8 Anforderungen an die Process Engine

In Tabelle 4.8 sind die Anforderungen im Überblick dargestellt.

[34] Bedeutung:
○ = nicht umgesetzt
◘ = teilweise umgesetzt
● = umgesetzt

Die Art der Linzenzzustellung für die Nutzung des Collaxa BPEL Server ist der einzige negative Aspekt, der gegen die Nutzung des Servers spricht. Dadurch wird eine Offline-Präsentation des Prototypen nahezu unmöglich, wenn nicht vorher schon eine Lizenz angefordert wurde.

In Ermangelung von openSource Alternativen muss eine kommerzielle Lösung für die Process Engine herangezogen werden, und der Collaxa BPEL Server eignet sich durch seine umfassenden Administrations- und Überwachungswerkzeuge gut für den vorgesehenen Prototypen. Zudem ist Möglichkeit der kommandozeilenbasierte Installation von Prozessbeschreibungen unabdingbar für eine automatische Integration von Web Services.

4.5.4 Java -APIs

Zur Umsetzung der Komponenten des Systems werden verschiedene APIs benötigt, die die Umsetzung von Web Services ermöglichen, das Lesen der semantischen Technologien (OWL-S) und das Ausgeben von XML-Daten (WSDL, BPEL4WS, u.a.) unterstützen.

Jena 2 Ontology API

Jena[35] ist ein Framework für den Entwurf von Semantic Web Applikationen. Es ist als Open Source Projekt unter sourceforge.net zu finden, welches aus einem Projekt der HP Labs Semantic Web Research Goup hervorgegangen ist.

Angeboten werden

- eine RDF API. Sie stellt zum einen Statement zentrierte Methoden zur Manipulation von RDF-Modellen anhand von RDF-Triples durch, sowie Resource-zentrierte Methoden um eine RDF-Modell als einen Satz von Resources mit Properties zu behandeln und weitere Funktionalitäten, die die Navigation in und die Erstellung von RDF-Dokumenten erleichtern.
- ARP – Jena's RDF/XML Parser, zum Einlesen von RDF-Dokumenten.
- die Anbindung von relationalen Datenbanken durch Backend Database Engines.
- RDQL ist eine RDF Data Query Language. Mit der Jena Implementation werden RDQL Statements in SQL Statements umgesetzt um angebundene relationalen Datenbanken abfragen zu können.
- ein Reasoning Subsystem, dass eine Inference Engine bereitstellt, um Schlussfolgerungen aus den Ontologien zu ziehen.
- ein Ontology Subsystem, mit dessen Hilfe RDF-basierte Ontologien (bisher sind dies OWL, DAML+OIL und RDFS) in Java abgebildet werden können. Ein Satz von Abstraktionen in Java erweitert die generischen RDF Resource und Property Klassen, um die Begriffe und deren Beziehungen, die in Ontologien zu finden sind, direkter nutzen zu können.

[35] http://jena.sourceforge.net/ , http://www.hpl.hp.com/semweb/jena2.htm

Die Jena 2 API wird verwendet, um die potenziellen Prozessbeschreibungen der openTRANS-Web Service Anbieter einzulesen, zu analysieren und aufeinander abzugleichen.

OWLS-API

OWL-S API[36] bietet eine Java API um OWL-S Web Service Beschreibungen zu lesen, auszuführen und zu schreiben.

Sie enthält Datenstrukturen, um die OWL-S Ontologie – vor allem Version 1.0 – abbilden zu können. Dort wo die Ontologie unvollständig ist, bietet auch die API noch keine Lösungen an.

Die zugrunde liegende API zur Abbildung der ontologischen OWL-Begriffe, auf denen OWL-S beruht, ist die oben beschriebene Jena 2 Ontologie API.

Die einzelnen Klassen zur Darstellung der OWL-S Begriffe bieten keine direkten Zugriffe auf die Funktionalitäten der Jena API an, aber über die Methode `getJenaResource()` kann von jeder OWL-S API Klasse die entsprechende Jena Darstellung zur Weiterverarbeitung abgerufen werden.

Apache Axis

Das Open Source Projekt Apache Axis[37] ging aus der IBM SOAP4J API[38] und bietet damit ein Implementation der SOAP[39] Spezifikation des W3C an.

Es bietet zudem ein Framework um SOAP Prozessoren wie Clients, Server und Gateways zu erstellen. Mittlerweile werden eine Java und eine C++ Implementation angeboten.

Weitere Funktionalitäten sind unter anderem:

- Ein einfacher Web Service Standalone Server.
- Ein Web Service Server der als Webapplikation im Apache TOMCAT Servlet Engine gestartet werden kann.
- Ein Tool zur Generierung von Java Klassen aus WSDL.
- Automatisches Deployment von SOAP Web Services (JWS).
  Dabei handelt es sich um reine Java Klassen, die als Quelltext in Dateien mit der Endung .jws vorliegen. Werden diese in den Webapplikationsordner von Axis gestellt, werden ihre Methoden automatisch als Web Service Methoden bereitgestellt.
- Support für alle wichtigen XML Datentypen.
  Zusätzlich bietet Axis ein einfaches erweiterbares Typ-Mapping-System zum Festlegen weiterer Serializer und Deserializer
- Java Beans werden automatisch de- und serialisiert.

36 http://www.mindswap.org/2004/owl-s/
37 http://ws.apache.org/axis/
38 http://www.alphaworks.ibm.com/tech/soap4j/
39 http://www.w3.org/TR/SOAP

Mit Axis JWS werden die Web Services implementiert, die die openTRANS-Dokumente per SOAP senden und empfangen.

Fraunhofer - Packages

Im Fraunhofer IAO Competence Center e-Business wurden einige Packages entwickelt, um Zugriffe auf Datenbanken, Logging, und das De- und Serialisieren von XML-Daten zu vereinfachen. Darauf aufbauend wurden bereits Packages implementiert, die das Verarbeiten von openTRANS-Geschäftsdokumenten ermöglichen.

Diese wurden für das Versenden der Dokumente über einen Axis-Web Services genutzt.

4.6 Systementwurf

4.6.1 Architektur

In den Anforderungen wurde bereits deutlich, dass eine Unterteilung des Process Integrators in verschiedene Komponenten Sinn macht. Das Integrationssystem sollte so unabhängig wie möglich von den Systemen der Einkäufer und Verkäufer realisiert werden. Zudem handelt es sich um die Verknüpfung zweier Web Services. Da es sich bei Web Services um durchgängig unabhängige Komponenten handelt, die sich nur über Interfaces beschreiben um miteinander zu arbeiten, erscheint es nur konsequent auch die Komponenten des Systems als einzelne Web Services anzubieten.

Bei den zu übermittelnden Datenpaketen zur Prozessintegration handelt es sich in allen Fällen um reine XML-Formate (OWL, OWL-S, WSDL, BPEL4WS und die zugehörigen Installationsdateien), welche sich bestens für den Datentransfer per SOAP eignen.

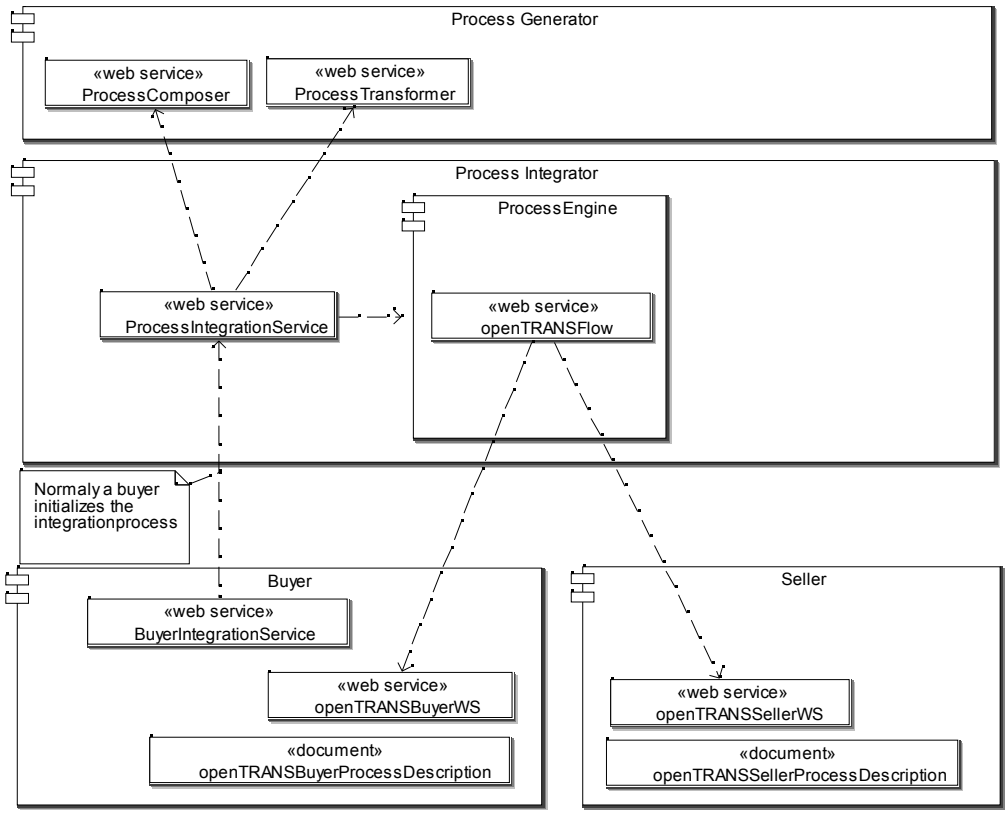

Abbildung 4.10 Systemüberblick Process Integrator

Somit ergeben sich folgende Systemkomponenten:

openTRANS-Web Services

- openTRANSBuyerWS
- openTRANSSeller WS

Diese Web Services bieten die Operationen an, mit denen die openTRANS-Dokumente zwischen den Geschäftspartnern übermittelt werden.

Potentielle Prozessbeschreibungen

- openTRANSBuyerProcessDescription
- openTRANSSellerProcessDescription

In diesen Beschreibungen ist festgelegt, welche Operationen eines openTRANS-Web Service welche Bedeutung besitzen, in welcher Anordnung sie gerufen werden dürfen, und welche Operationen sie von ihrem Gegenüber erwarten.

BuyerIntegrationService

Der Einkäufer benötigt zudem eine Anwendung, mit der er die Integration starten kann. Diese Anwendung hat nur zur Aufgabe die URIs der Prozessbeschreibungen von Ein- und Verkäufer Web Service an den ProzessIntegrationService weiterzurei-

chen und zu empfangen, dass der openTRANS-Flow-Service bereitsteht, und unter welcher Adresse er zu erreichen ist.

Process Integrator

In dieser Komponente sind die sind ProcessIntegrationService und ProcessEngine zusammengefasst, da der Process Integrator direkten Zugriff auf der Process Engine benötigt, um den Prozess zu installieren.

ProcessIntegrationService

Dieser startet, überwacht und verwaltet die Integration der openTRANS-Web Services und informiert den Einkäufer über dessen Zustand.

ProcessEngine

Auf der ProcessEngine wird der vom ProcessIntegrationService gelieferte openTRANS-Flow-Service installiert und vom Einkäufer gestartet.

openTRANSFlow

Die aus der komponierten Prozessbeschreibung der PPDs generierte ausführbare Prozessbeschreibung wird für den Process Engine in einen Web Service umgewandelt. Dieser Flow-Service steuert den Ablauf von Absichts-, Abwicklungs- und Vereinbarungsphase der Markttransaktion zwischen den Geschäftspartnern.

Process Generator

Der Process Generator besteht aus ProcessComposer und ProcessTransformer. In Abbildung 4.10 sind ProcessComposer und ProcessTransformer als jeweils einzelne Web Services in einer Komponente zusammengefasst. Sie könnten jedoch auch in einzelnen Komponenten von verschiedenen Anbietern realisiert werden.

ProcessComposer

Der ProcessComposer komponiert aus den potenziellen Prozessbeschreibungen der Geschäftspartner eine einzelne Prozessbeschreibung, die in einen ausführbaren Prozess in einer PEL umgewandelt werden kann.

ProcessTransformer

Der ProcessTransformer wandelt eine Prozessbeschreibung in eine ausführbare Prozessbeschreibung für eine ProcessEngine um.

Ein- und Verkäufer müssten zusätzlich zu ihrem openTRANS-Web Service nur eine potenzielle Prozessbeschreibung PPD bereitstellen (siehe Abbildung 4.10). Damit sind sie ansonsten unabhängig von der Prozessintegration, und es sind keine weiteren Anpassungen des e-Procurement-Systems der Geschäftspartner notwendig. Wird eine komponierte Prozessbeschreibung der openTRANS-Web

Services erreicht, erfüllt diese die Anforderungen beider Geschäftspartner an einen komponierten Prozess.

Die hier unterteilten Systemkomponenten arbeiten als Web Services und können daher wie in Kapitel 4.4.4 gefordert weitgehend unabhängig miteinander interagieren. Es ist möglich jede der Komponenten – ProcessIntegrator, ProcessComposer und ProcessTransformer – auf jeweils einem eigenen System laufen zu lassen, und die generierten Prozessbeschreibungen (PPD, CPD und EPD) per SOAP zu übermitteln.

Allerdings sind gerade bei der Schnittstelle zur ProcessEngine unter Umständen noch weitere Dokumente erforderlich, die der Installation des EPD dienen.

Hier sind einige Konfigurationskenntnise der Process Engines notwendig, die im Transformer oder zumindest im Integrator enthalten sein müssen, um die EPD entsprechend anpassen zu können. Der ProcessIntegrator lässt sich die EPD vom Process Transformer und alle nötigen Konfigurationsdateien erstellen und übernimmt die Installation der EPD auf der Process Engine.

Im Folgenden werden die Komponenten des Integrationssystems genauer vorgestellt.

4.6.2 Integration

Der ProcessIntegratonService (PIS) des ProcessIntegrator (PI) erhält von einem Einkäufer die potenziellen Prozessbeschreibungen (PPDs) des einkaufenden und des verkaufenden openTRANS-Web Service.

Anhand dieser erkennt der PI, ob schon einmal eine ausführbare Prozessbeschreibung (EPD) für diese PPDs generiert wurde. Ist dies der Fall gibt er die Adresse des bereits installierten Flow-Service an den Einkäufer zurück. Ist dies nicht der Fall gibt er die PPDs weiter an den ProcessComposer.

Von diesem erhält er eine komponierte Prozessbeschreibung (CPD) der beiden openTRANS-Web Services. Diese wird nun an einen ProcessTransformer weitergereicht, der sie in eine EPD für eine bestimmte Process Engine transformiert. Der PI installiert die EPD auf der Process Engine und informiert den Einkäufer über den bereitliegenden Flow-Service.

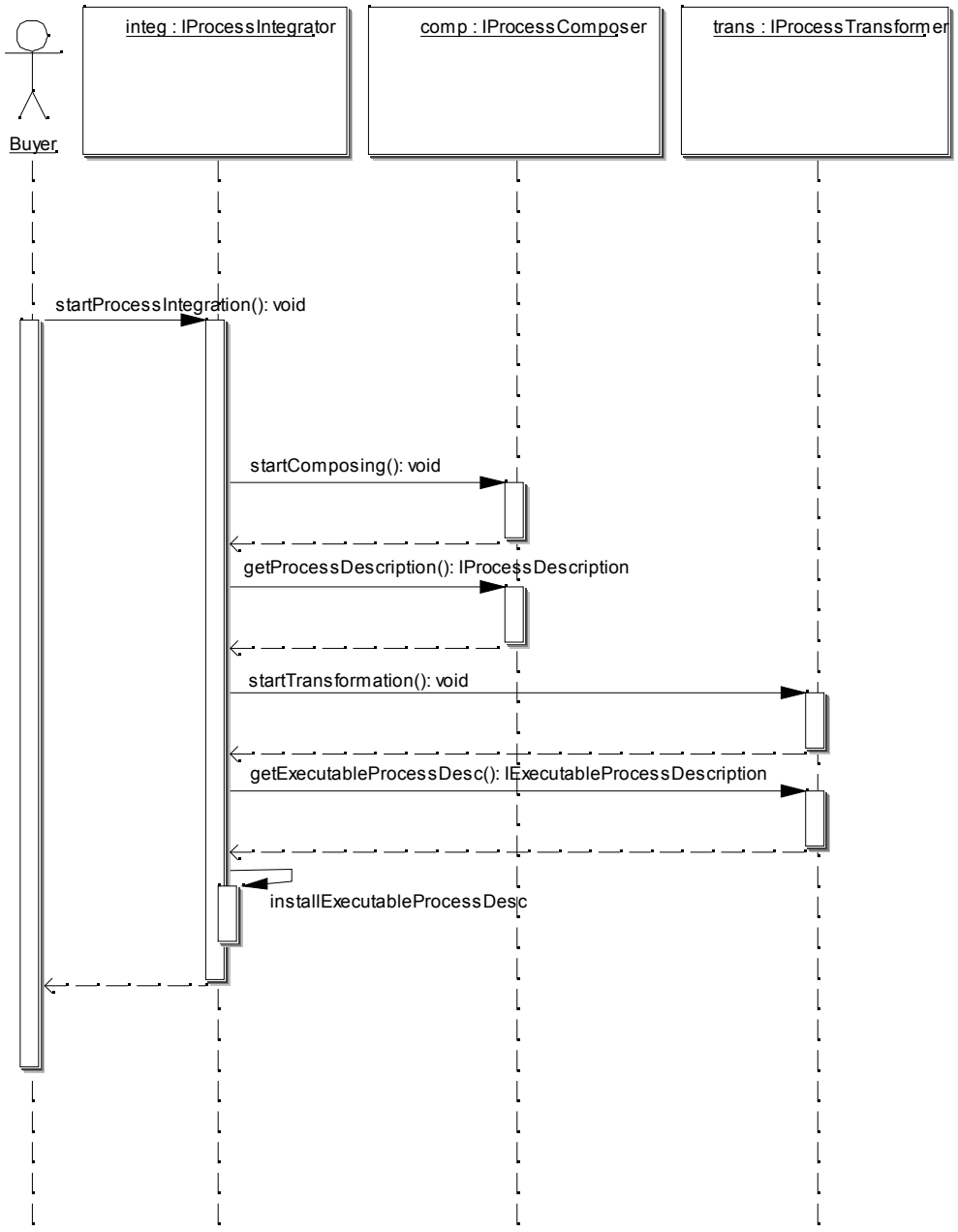

Abbildung 4.11 Ablauf der Process Integration

4.6.3 Potenzielle Prozessbeschreibung und OWL-S

Die Übertragung eines Geschäftsdokuments entspricht dem Aufruf einer Operation des Web Service eines Geschäftspartners. Eine solcher Operationsaufruf wird in der Process.owl als `AtomicProcess` bezeichnet. Eine Komposition von mehreren Atomic Processes ist ein `CompositeProcess`. Beide sind Subklassen der `Process`-Klasse.

Aufbauend auf der OWL-S Ontologie zur Beschreibung von Web Services, müssen Erweiterungen und Spezialisierungen getroffen werden, um eine potenzielle Prozessbeschreibung den Anforderungen entsprechend erstellen zu können.

OWL-S versteht sich als Ontolgie zur Beschreibung von Web Services für eine automatische Komposition.[OWLS1]. Dabei geht OWL-S nicht von einer verschachtelten Nutzung von Web Services aus, wie beim Ablauf eines Einkaufs, bei dem Verkäufer und Einkäufer abwechselnd miteinander kommunizieren, sondern einer sequentiellen Anordnung. Das ist zum Beispiel der Fall wenn ein Service genutzt wird um die momentanen Aufenthaltsort zu bestimmen, und mit dessen Ergebnis nach dem nächstgelegenen Restaurant gesucht wird.

Grundsätzlich ist die Prozessbeschreibungsontologie von OWL-S daher nicht für eine potenzielle, sondern für eine ausführbare Prozessbeschreibung gedacht. Aus diesem Grund sind einige Erweiterungen nötig.

Es fehlen Begrifflichkeiten, die als Anforderungen an die potentielle Prozessbeschreibung gestellt werden, wie Geschäftsdokumenten den Status von KANN und MUSS zuzuweisen (siehe Kapitel 4.4.3).

Trennung von PPD-Konstrukten und EPD-Konstrukten

Bei Kardinalitäten und dem MUSS/KANN-Status eines `AtomicProcess` (AP) handelt es sich rein um potenzielle Prozessbeschreibungs-Konstrukte, und nicht um ausführbare Prozessbeschreibungs-Konstrukte.

Daher wurde ein Konzept entworfen, dass die potenziellen Konstrukte von den ausführbaren Konstrukten trennt.

Die in einer PPD mit Hilfe von OWL-S Begriffen umzusetzenden Anforderungen an die Kontrollflusskonstrukte werden in der folgenden Tabelle als potenzielle oder ausführbare Konstrukte festgelegt.

Weitere Kontrollflusskonstrukte werden nicht benötigt. Die Umsetzung von Kardinalitäten in Schleifen erfolgt erst bei der Komposition.

Als erstes werden die beiden benötigten potenziellen Konstrukte betrachtet, und darauffolgend die ausführbaren Konstrukte.

Anforderungen einer openTRANS-Prozessbeschreibung	**Umsetzung durch potenzielles (P) oder ausführbares (A) Konstrukt**
Fehlermeldungen	A
Ausnahmebehandlungen	A
Kompensation	A
Timeout	A
Zusammenarbeit verschiedener Parteien und deren Rollen	A
Kontrollfluss	A
WENN-DANN Kontrollflusskonstrukt (openTRANS - Anforderung)	A
Sequentielle Aktivitäten (openTRANS - Anforderung)	A
Parallele Aktivitäten (openTRANS - Anforderung)	A
Kardinalitäten von Aktivitäten (openTRANS - Anforderung)	P
MUSS - KANN Aktivitäten (openTRANS - Anforderung)	P

Tabelle 4.9 Umsetzung der Anforderungen in OWL-S

4.6.4 Potenzielle Konstrukte der openTRANSProcess.owl

Außer den in der Anforderung enthaltenen benötigten potenziellen Konstrukten wie Kardinalität und Status einer AP muss auch unterschieden werden, ob ein deklarierter `AtomicProcess` als Sender oder Empfänger AP dient.

Für den ProcessComposer ist bei der Komposition wichtig, ob die Anforderungen an den AP eines Geschäftspartner von einem sendenden oder empfangenden AP gestellt werden.

Legt beispielsweise ein Verkäufer eine RFQ-AP mit einer Kardinalität von 1 fest, so muss der Sender AP eine Kardinalität von 1 oder 1-n besitzen.

Legt der gleiche Verkäufer für eine QUOTATION eine Kardinalität von 1 fest, so bedeutet das nicht, dass der Empfänger die QUOTATION wirklich empfangen können muss.

Der Process Composer benötigt also Informationen, darüber wer einen AP in seiner AP festlegt, und an wen er seine Erwartungen stellt.

Es werden folgende potenziellen Konstrukte für die PPD benötigt:

- Sender-Empfänger-AP,
- Kardinalitäten,

- MUSS-KANN-Status.

Wie schon erwähnt entspricht das Übertragen eines Geschäftsdokuments einem `AtomicProcess` in OWL-S.

Es wird nun eine openTRANSProcess.owl aufgebaut, die die Process.owl von OWL-S erweitert, wie es in Abbildung 4.12 zu sehen ist.

Statt für die einzelnen Übertragungen von Geschäftsdokumenten, wie zum Beispiel „setRFQAtomicProcess", direkt Subklassen von AtomicProcess zu bilden, werden Zwischentypen eingefügt, die die APs exakter definieren. Es wird eine openTRANS-AtomicProcess `openTRANSAP` als Subklasse von `AtomicProcess` deklariert, von dem nun weitere Subklassen gebildet werden.

Sender-Empfänger-AtomicProcess

Von `openTRANSAP` leiten sich die Klassen `openTRANSSellerAP` und `openTRANS-BuyerAP` ab, die die Superklassen für die Ihnen zugeordneten Empfangs-APs stellen. So ist `setRFQ` beispielsweise ein Subklasse von `openTRANSSellerAP`, da RFQs an den Verkäufer übergeben werden.

Dadurch ist es nun möglich immer nur die Empfangs-APs in einer PPD angeben zu müssen, und nicht jeweils eine Empfangs- und eine Sende-AP für jedes Geschäftsdokument. Je nach dem von wem die PPD stammt – Einkäufer oder Verkäufer – weiss der verarbeitende Process Composer bei jeder instanziierten AP der PPD, ob es sich um ein Sende-Gebot oder um eine eine Empfangs-Erwartung an den Geschäftspartner handelt.

Zudem wird dadurch auch definiert, welche APs bzw. Web Service Operationen ein Geschäftspartner von seinem Gegenüber überhaupt erwarten kann.

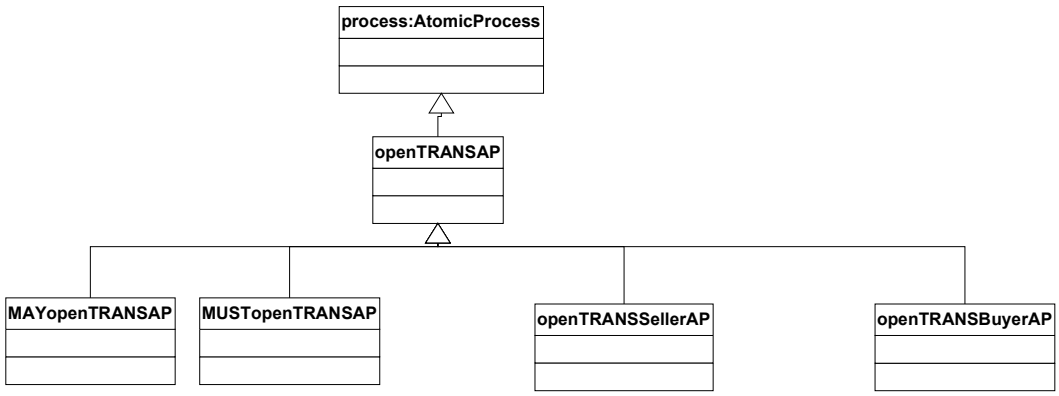

Abbildung 4.12 openTRANSProcess.owl - Hauptklassen

MUSS, KANN, WERDE Status von Geschäftsdokumenten

Weiterhin leiten sich von `openTRANSAP` die Subklassen `MAYopenTRANSAP` und `MUSTopenTRANSAP` ab, die die Superklassen für verpflichtenden und möglichen APs stellen. (MUSS NICHT- und MUSS-Status)

Darauf aufbauend werden nun Klassen wie MAYsetRFQ, die eine Subklasse von setRFQ und MAYopenTRANSAP sind und MUSTsetRFQ als Subklasse von MUSTopenTRANSAP und setRFQ deklariert.

Durch diese Unterscheidung ist es einem Geschäftspartner möglich in seiner PPD ohne grosse Umstände zu deklarieren, ob ein AP einen MUSS- (Subklasse von MUSTopenTRANSAP) oder einen MUSS NICHT-Status (Subklasse von MAYopenTRANSAP) erhält. (vgl. Abbildung 4.13)

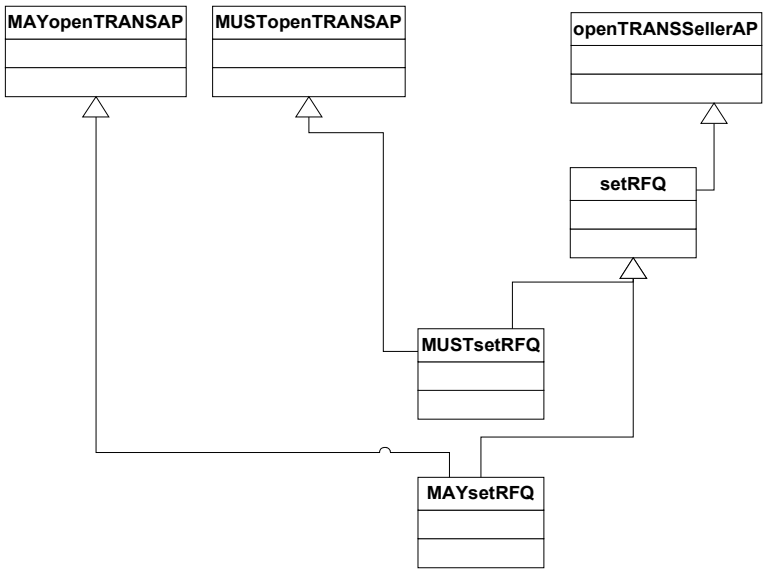

Abbildung 4.13 openTRANSProcess.owl - RFQ Atomic Processes

Ein MUSTopenTRANSAP, die als Empfangs-AP in einer PPD genutzt wird – beispielsweise definiert ein Verkäufer eine MUSTsetRFQAP, was bedeutet, dass sein Geschäftspartner eine RFQ senden MUSS – entspricht grundsätzlich immer einer Kardinalität von größer als 1.

Kardinalitäten

Definiert der Sender eines Geschäftsdokument eine Sende - AP als MUSTopenTRANSAP, kann der Empfänger auch eine Kardinalität von 0-n für diesen AP aufweisen.

Beispiel dafür wäre ein Verkäufer der eine MUSTsetINVOICE als Empfangs-AP – für ihn ist es ein Sende-AP – für einen Einkäufer definiert. Der Verkäufer sagt damit aus, dass ein Einkäufer eine INVOICE empfangen können MUSS. Dies ist schon erfüllt, wenn der Verkäufer eine Kardinalität von 0-n für setINVOICE festgelegt hat.

Zusammengefasst bedeutet dies folgendes:

- Legt der Sender eines Geschäftsdokuments die zugehörige AP als MUSTopenTRANSAP fest, können er und der Empfänger eine Kardinalität von 0-n oder

höher (1, 1-n) aufweisen; es müssen nur beide die Geschäftsdokumente senden und empfangen können.

- Legt der Empfänger eines Geschäftsdokuments die zugehörige AP als `MUST-openTRANSAP` fest, muss der Sender eine Kardinalität von mindestens 1 oder 1-n aufweisen; der Empfänger kann wiederum eine Kardinalität von 0-n festlegen.
  Das ist allerdings unsinnig, da der Sendevorgang, wegen der Kardinalität des Sender-APs, auf jeden Fall stattfinden wird, was einer Kardinalität des Empfänger-APs von mindestens 1 gleichkommt.

Somit ist die Kardinaltät nicht immer gleich, wenn ein AP als `MUST-` oder `MAY-openTRANSAP` bestimmt wird. Daher werden die Kardinalitäten getrennt von `MUST-` und `MAYopenTRANSAP` festgelegt.

Jeder `openTRANSAP` kann über die Property `cardinalityIs` eine Instanz der Klasse `Cardinality` referenzieren.

In dieser sind Minimalkardinalität und Maximalkardinalität über die Properties `minCardinality` und `maxCardinality` als Werte des XML-Schema Types `nonNegativeInteger` festgelegt.

Von `Cardinality` leiten sich wiederum zwei Subklassen, `COULD_Cardinality` und `WILL_Cardinality` ab, wie in Abbildung 4.14 zu sehen.

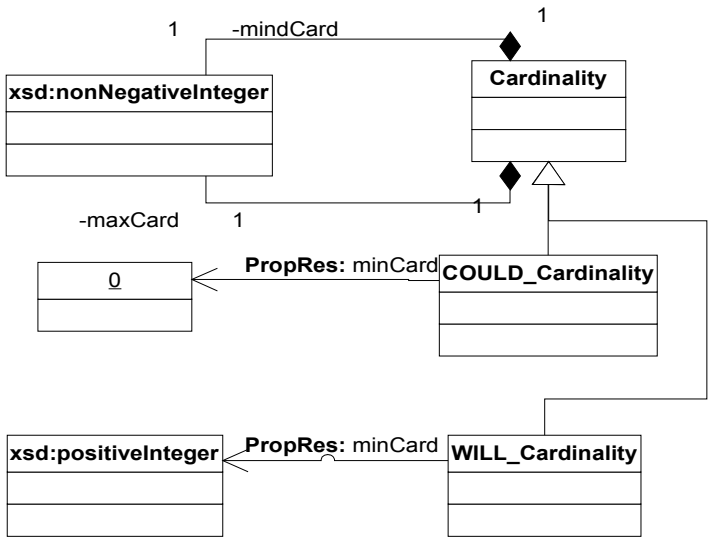

Abbildung 4.14 openTRANSProcess.owl - Kardinalitäten

Für `COULD_Cardinality` wird die Property `minCardinality` über das ontologische Konzept der Property-Restriction auf 0 festgelegt, da diese Klasse die Superklasse aller Kardinalitäten von Geschäftsdokumenten mit dem Status KANN ist, wie zum Beispiel der Klasse `Cardinality_01` (was der Kardinalität 0-1 entspricht) oder `Cardinality_0n` (was der Kardinalität 0-n entspricht).

Für `WILL_Cardinality` wird die Property `minCardinality` auf denXML-Schema Typ `positiveInteger` festgelegt, da diese Klasse die Superklasse aller Kardinalitäten da diese Klasse die Superklasse aller Kardinalitäten von Geschäfts-

dokumenten mit dem Status WERDE ist, wie zum Beispiel der Klasse `Cardinality_1` (was der Kardinalität 1 entspricht) oder `Cardinality_1n` (was der Kardinalität 1-n entspricht).

Abbildung 4.15 stellt das Gesamtsystem der potenziellen Konstrukte der openTRANSProcess.owl dar.

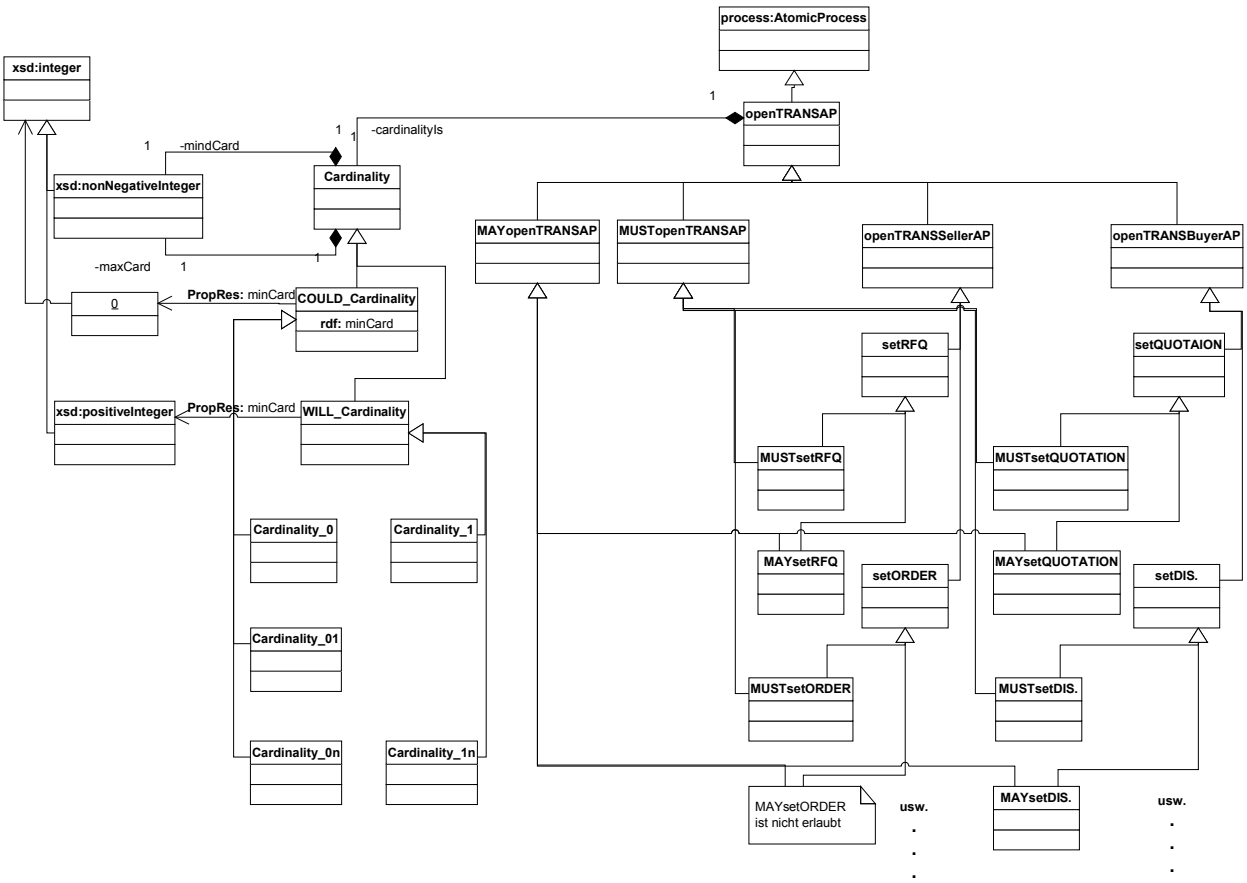

Abbildung 4.15 openTRANSProcess.owl - potentielle Konstrukte

Austausch von MUSTopenTRANSAPs

Wird eine `openTRANSAP` als `MUSTopenTRANSAP` definiert, muss sie in jedem Fall vom Partner unterstützt werden. Was ist aber, wenn ein Einkäufer beispielsweise definieren will, er MUSS eine ORDERRESPONSE erhalten, diese kann aber weggelassen werden, wenn eine INVOICE gesendet wird?

Hier wird eine Object Property für `MUSTopenTRANSAPs` vorgesehen, die diese Definition erlaubt: `replacableBy`, Dessen Deklaration wird in Listing 4.2 dargestellt.

```
1. <owl:ObjectProperty rdf:ID="replaceableBy">
2.   <rdfs:domain rdf:resource="#MUSTopenTRANSAP"/>
3.   <rdfs:range rdf:resource="#MUSTopenTRANSAP"/>
4. </owl:ObjectProperty>
```

Listing 4.1 replaceableBy

Eine MUSTopenTRANSAPs kann dabei von einer oder einer von mehreren MUST-openTRANSAPs ersetzbar sein. MAYopenTRANSAPs kommen als „range“ dabei nicht in Frage, da solche nicht in jedem Fall ausgeführt werden. Die Property darf beliebig oft in einer MUSTopenTRANSAPs vorkommen.

4.6.5 Ausführbare Konstrukte der openTRANSProcess.owl

Ausführbare Konstrukte (ControlConstructs CC), die direkt von OWL-S übernommen werden können sind:

- Sequence CC
  ⇨entspricht der Anforderung für sequentielles Senden und Empfangen von Geschäftsdokumenten.
- Unordered CC
  ⇨entspricht der Anforderung für paralleles Senden und Empfangen von Geschäftsdokumenten.
- If-Then-Else
  ⇨Erweiterungen werden benötigt.
- Timeouts werden mit der Property timeout dargestellt, die für jede Process-Component definiert werden kann.
  ⇨Die zugehörige Kompensation muss ergänzt werden. (siehe untern)
- Zusammenarbeit verschiedener Parteien und deren Rollen wird durch den Bezug der APs auf verschiedene PortTypes des WSDL festgelegt.

Zusätzlich werden noch ein ausführbare Konstrukte für

- Kompensation
- Fehlermeldungen
- Ausnahmebehandlung

benötigt, wobei Fehlermeldungen und Ausnahmebehandlungen in diesem Prototyp vernachlässigt werden wie in Kapitel 4.4.1 erläutert wurde.

If-Then-Else

Das If-Then-Else Konstrukt von OWL-S erlaubt eine Definition voneinander abhängiger Geschäftsdokumenten: WENN eine RFQ erscheint DANN muss auch eine QUOTATION erfolgen.

Das If-Then-Else Konstrukt bedient sich dabei der Condition Klasse, die schon in Kapitel 4.6.3 behandelt wurde. Da sie nicht ausgearbeitet ist, werden Subklassen von Condition gebildet, die für die Bedingungen ausdrücken, die in einem Interaktionsablauf auftreten können. Das sind:

- RFQoccuredCondition, für das Auftreten einer RFQ.

- DISPATCHorRECEIPTACKoccuredCondition für das Auftreten einer DISPATCH-NOTIFICATION oder einer RECEIPTACKNOWLEDGEMENT.

Kompensation

Kompensationen sind immer dann nötig, wenn in einem komponierten Prozess Zustände auftreten können, die einer der Geschäftspartner wünscht, der andere aber nicht erlaubt.

Das ist zum Beispiel der Fall, wenn ein Einkäufer eine zweite ORDERCHANGE senden möchte, der Verkäufer aber nur eine erlaubt. Das ist keine Bedingung, die das Zustandekommen eines komponierten Prozesses insgesamt verhindert, aber wenn dieser Zustand im laufenden Prozess auftritt muss der Teilnehmer, dessen AP nicht durchgeführt werden kann, darüber informiert werden können, wenn er dies wünscht (siehe Kapitel 4.4.3).

Ein `openTRANSAP` referenziert daher null bis ein `Cardinality_Compensation` und null bis eine `Timeout_Compensation`.

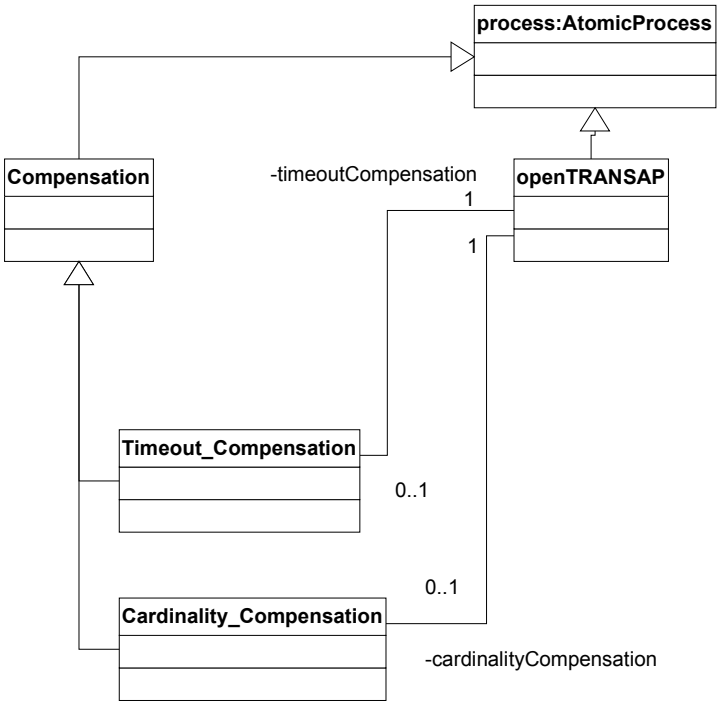

Abbildung 4.16 openTRANSProcess.owl - Compensation

Ein `Cardinality_Compensation` - AP wird immer dann ausgeführt, wenn die gewünschte Kardinalität des zugehörigen `openTRANSAP` vom Geschäftspartner nicht weiter erfüllt werden kann.

Wenn wie im obigen Beispiel eine zweite ORDERCHANGE gesendet wird, wird die `Cardinality_Compensation` ausgeführt. Das ist lediglich der Aufruf einer Web Sercice Operation des Geschäftspartners, der die `Compensation` definiert hat.

Eine Timeout_Compensation wird immer dann ausgeführt, wenn länger auf die Ausführung einer openTRANSAP gewartet werden muss, als in der PPD festgelegt ist.

Bei einer MUSTopenTRANSAP ist die Timeout_Compensation immer gleichbedeutend mit der Beendingung der Prozess-Instanz.

4.6.6 Web Service Operationen und die Atomic Processes

Die Operation, die ein AtomicProcess in einem Web Service referenziert, wird über eine Verknüpfung zum WSDL des Web Service erreicht. Diese Verknüpfung stellt das ServiceGrounding.owl von OWL-S her. Jeder AtomicProcess besitzt ein WsdlAtomicProcessGrounding (siehe Listing 4.3), in dem festgelegt wird, welche Operation welches PortTyps im WSDL einem AtomicProcess zugeordnet wird.

```
1. <WsdlAtomicProcessGrounding
2.   rdf:ID="setRFQonSellerProcessGrounding">
3.
4.   <owlsProcess
5.     rdf:resource="#setRFQonSellerProcess"/>
6.
7.   <wsdlDocument>
8.     file://localhost/c:/openTRANSBuyerFlow.wsdl
9.   </wsdlDocument>
10.
11.   <wsdlOperation>
12.     <WsdlOperationRef>
13.       <portType>
14.         openTRANSBuyerFlowCallback
15.       </portType>
16.       <operation>setRFQ</operation>
17.     </WsdlOperationRef>
18.   </wsdlOperation>
19.   ...
20.</WsdlAtomicProcessGrounding>
```

Listing 4.2 OWL-S - WsdlAtomicProcessGrounding

Der Geschäftspartner definiert einen PortType für die von ihm angebotenen Web Service Operationen und einen PortType für die von seinem Gegenüber erwarteten.

In BPEL4WS – in dem die spätere EPD erstellt wird – ist es notwendig die WSDL-portTypes nach den Rollen zu unterscheiden, um festzulegen, welche Operationen vom Gegenüber erwartet werden können, und welche er selbst anzubieten hat. Diese Unterscheidung wird in der PPD über die Subklassen von AtomicProcesses für Einkäufer- und Verkäufer- AtomicProcesses erreicht.

Dennoch ist es notwendig, dass in der WsdlAtomicProcessGrounding des AP der richtige portType (eigene Operationen oder erwartete Operationen) für den einzelnen AtomicProcess (eigener oder erwartetet AtomicProcess) referenziert wird, damit in der späteren BPEL4WS die richtigen portTypes verwendet werden können.

Um diese Zuordnung im ProcessComposer überprüfen zu können, muss in der PPD bereits bekannt sein, welcher WSDL- `portType` welcher Rolle zugeordnet wird.

Diese werden in diesem Konzept dem `CompositeProcess` zugeteilt, der die einzelnen `openTRANSAPs` zu einem openTRANS-`CompositeProcess` komponiert.

Dazu werden OWL Object Properties erstellt, die die Werte der portTypes enthalten, die später in der `WsdlAtomicProcessGroundings` genutzt werden können.

Dies sind

- `portTypeBuyer` und
- `portTypeSeller`.

```
1. <owl:ObjectProperty rdf:ID="portTypeBuyer">
2.   <rdfs:domain
3.    rdf:resource="#openTRANSCompositeProcess"/>
4.   <rdfs:range
5.    rdf:resource=
6.        "http://www.w3.org/2001/XMLSchema#anyURI"/>
7. </owl:ObjectProperty>
8. <owl:ObjectProperty rdf:ID="portTypeSeller">
9.   <rdfs:domain
10.    rdf:resource="#openTRANSCompositeProcess"/>
11.  <rdfs:range
12.    rdf:resource=
13.        "http://www.w3.org/2001/XMLSchema#anyURI"/>
14. </owl:ObjectProperty>
```

Listing 4.3 portType Bestimmung der Partner

Um diese Object Properties auf den `CompositeProcess` zu beziehen, der die `openTRANSAPs` enthält, muss eine Subklasse `openTRANSCP` von `CompositeProcess` definiert werden, der diese Properties zugewiesen werden können.

4.6.7 Der Metaprozess der potenziellen Prozessbeschreibung

In Kapitel 4.4.3 wurden unter anderem die Mindestanforderungen an eine openTRANS-Prozessbeschreibung bezüglich der Abhängigkeit und des Interaktionsablaufs herausgestellt. Abbildung 4.4 zeigt in einem Aktivitätsdiagramm den gefundenen Mindestablauf an.

Dieser Mindestablauf kann durch die PPDs der Geschäftspartner abgeändert werden. Allerdings nur innerhalb der in Kapitel 4.4.3 gefundenen Bedingungen:

1. Es können Geschäftsdokumente entfernt werden, die eine Kardinalität von 0 haben können.
2. Die Kardinalitäten können sich nur zwischen den pro Geschäftsdokument aufgezählten minimal und maximal Kardinalitäten bewegen.
3. Die Abhängigkeiten der einzelnen Geschäftsdokumente voneinander können nicht verringert sondern nur erhöht werden.

4. Sequentielle Anordnungen von Geschäftsdokumenten dürfen nicht in parallele Anordnungen abgeändert werden, nur umgekehrt.

Weitere Kontrollflusskonstrukte können eingesetzt werden, was aber nur zu weiteren Einschränkungen des Interaktionsablaufs führt, die prinzipiell nicht von den Geschäftspartnern gewünscht werden. Der Grund dafür ist, dass diese mit so zahlreichen Kunden und so vielen Verkäufern wie möglich in Kontakt treten wollen.

Die Metaprozessbeschreibung (Meta Process Description, MPD), die die herausgestellten Mindestanforderungen enthält, wird zusätzlich zu den PPDs in den Process Composer eingelesen. An diesem kann der ProcessComposer nun überprüfen ob eine PPD gültig ist, oder nicht.

Diese Funktionalität ist vergleichbar mit der Validierung eines XML Dokuments anhand einer DTD[40] oder eines XML Schema.

Ferner stellt sich die Frage, wie der ProcessComposer die MPD erhält.

Sie könnte

1. in der openTRANSProcess.owl referenziert werden,
2. sich als unabhängige Prozessbeschreibung wie die PPDs sich aus den Konstrukten der openTRANSProcess.owl zusammensetzen.

Durch letztere Alternative würde die MPD unabhängig von der openTRANSProcess.owl in den Process Composer eingelesen werden, und wäre damit tauschbar.

Die MPD zeichnet sich allerdings gerade durch ihre Nicht-Austauschbarkeit aus, da sie die absoluten Minimalbedingungen für einen openTRANS-Prozessablauf stellt.

Andererseits ist es vorstellbar, dass sich zwei Geschäftspartner auf eine noch restriktivere MPD einigen.

Daher wird eine gemischte Variante bevorzugt, bei der eine Referenz auf die Standard MPD in der openTRANSProcess.owl angibt, die standardmäßig genutzt wird, wenn der ProcessComposer keine andere MPD erhält. Diese neue MPD kann dann ebenfalls anhand der Standard MPD überprüft werden, dessen Bedingungen sie auch erfüllen muss.

Die MPD kann im Anhang studiert werden.

4.6.8 Komposition der potenziellen Prozessbeschreibungen

Der ProcessComposer erhält die PPDs der Geschäftspartner vom ProcessIntegrator.

Beide beschreiben die Integrationsfähigkeit ihres openTRANS-Web Service aus den Konstrukten, die in der oben definierten openTRANSProcess.owl und der OWL-S Process.owl bereitgestellt werden.

Vorbedingungen zur Komposition

[40] Document Type Definition, stellt wie ein XMLSchema Regeln bereit, die Festsetzen, wie die Elemente und Attribute eines XML Dokuments angeordnet werden dürfen.

Zunächst stellt er nun fest, ob es sich um gültige openTRANS-Prozessbeschreibungen handelt.

Dazu prüft er zum ersten die allgemeine Gültigkeit der PPDs:

1. Stammen die verwendeten Konstrukte nur aus der openTRANSProcess.owl oder der Process.owl?
2. Sind die PPDs gültig im Sinne der openTRANS-MPD?

Des weiteren wird die prinzipielle Komponierbarkeit überprüft:

3. Werden alle MUSS-Geschäftsdokumente, die ein Geschäftspartner wünscht vom gegenüber empfangen, bzw. gesendet.
4. Könnten Deadlocks durch die Komposition entstehen,

Deadlocks im laufenden Prozess entstehen, wenn die beiden Web Services auf jeweils ein Geschäftsdokument warten, um fortzufahren, aber diese nicht eintreffen, weil sie erst nach Eintreffen der Geschäftsdokumente gesendet werden, auf die gewartet wird (siehe Abbildung 4.17).

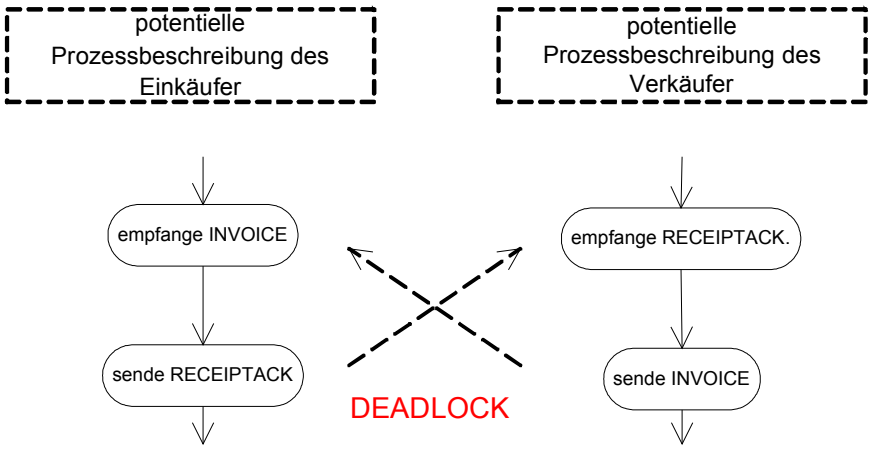

Abbildung 4.17 Deadlock der potenziellen Prozessbeschreibungen

Dies kann allerdings nur dann auftreten, wenn die Geschäftspartner freie Hand bei der Auswahl der Reihenfolge von sendenden und empfangenden `openTRANSAPs` haben. Das ist der Fall, wenn Sende-APs und Empfangs-APs, die in der MPD als parallel angegeben sind, in einer Sequenz definiert werden.

Die einzige Parallelität von Geschäftsdokumenten in einer PPD ist bei DISPATCHNOTIFICATION und INVOICE gegeben, wobei es sich aber um zwei entweder sendende bzw. empfangende – je nach Geschäftspartner – APs handelt. Daher ist hier ein Deadlock nicht möglich (siehe auch Kapitel 4.4.3).

Erst wenn diese vier Bedingungen erfüllt sind, kann eine erfolgreiche Komposition erreicht werden.

Allgemeines zur Komposition

Über die PPDs beschreibt ein Geschäftspartner mit Hilfe der `openTRANSAPs` die Web Service Operationen, mit denen sein Web Service Geschäftsdokumente empfangen kann, bzw. die Operationen, die er bei seinem Geschäftspartner für den Empfang seiner Dokumente erwartet.

In einer komponierten Prozessbeschreibung müssen nun beide Seiten berücksichtigt werden. An der Stelle, wo in einer PPD nur der AP eines Geschäftspartners stand, müssen in der CPD die korrespondierenden APs beider festgelegt werden, wie in Listing 4.4 zu sehen.

Die eine beschreibt dann den Empfang eines Dokuments von einem Sender und die andere die Weiterleitung desselben an den Empfänger.

```
1. <process:CompositeProcess rdf:ID="openTRANSFlowProcess">
2.  <process:composedOf>
3.   <process:Sequence>
4.     <process:components rdf:parseType="Collection">
5.
6.      <opTprocess:openTRANSBuyerAP
7.         rdf:about="#setRFQonProcess"/>
8.
9.      <opTprocess:openTRANSBuyerAP
10.         rdf:about="#setRFQonSeller"/>
11.
12.    </process:components>
13.   </process:Sequence>
14.  </process:composedOf>
</process:CompositeProcess>
```

Listing 4.4 Beispielausschnitt einer komponierten Prozessbeschreibung

Hierbei ist zu beachten, das ein Geschäftsdokument immer erst vom Prozess empfangen wird, bevor es weitergeleitet werden kann. Teilweise müssen der Empfangs-AP und der Sende-AP auch in getrennten Kontrollflusskonstrukten eingebettet werden, wie nachfolgend beschrieben. Können sie direkt nach dem Empfang weitergeleitet werden, kann man sie in einem `CompositeProcess` zusammenfassen. (siehe Listing 4.5, in Zeile 6 ist die Nutzung des definierten CompositeProcess von Zeile 14 dargestellt)

```
1. <process:CompositeProcess rdf:ID="openTRANSFlowProcess">
2.  <process:composedOf>
3.   <process:Sequence>
4.     <process:components rdf:parseType="Collection">
5.
6.      <process:CompositeProcess
7.         rdf:about="#openTRANSsetRFQCP"/>
8.
9.     </process:components>
10.   </process:Sequence>
11.  </process:composedOf>
12.</process:CompositeProcess>
13.
14.<process:CompositeProcess rdf:ID="openTRANSsetRFQCP">
```

```
15. <process:composedOf>
16.  <process:Sequence>
17.    <process:components rdf:parseType="Collection">
18.
19.      <opTprocess:openTRANSBuyerAP
20.        rdf:about="#setRFQonProcess"/>
21.
22.     <opTprocess:openTRANSBuyerAP
23.        rdf:about="#setRFQonSeller"/>
24.
25.    </process:components>
26.   </process:Sequence>
27.  </process:composedOf>
28.</process:CompositeProcess>
```

Listing 4.5 - Beispielausschnitt einer komponierten Prozessbeschreibung

Bei der Durchführung der Komposition geht der Process Composer die APs einer PPD einzeln durch und macht aus, wann die entsprechenden Gegenstücke des APs in der potenzielle Prozessbeschreibung des Partners verlangt werden, und bildet sie dann in der gemeinsamen CPD ab.

Deadlocks können grundsätzlich nicht in einer MPD konformen openTRANS-Prozessbeschreibung vorhanden sein. Zudem ist in der MPD bis auf eine Ausnahme (DISPATCHNOTIFICATION und INVOICE) eine sequentiell Anordnung der APs vorgeschrieben. Daher können kaum Differenzen der potentiellen Prozessbeschreibungen entstehen.

Beachtet werden muss, dass `MAYopenTRANSAPs` vorkommen können, die zwar gesendet aber nicht empfangen werden können. In diesem Fall muss in der EPD ein Empfang des Geschäftsdokuments vorgesehen sein, jedoch keine Weiterleitung an den Geschäftspartner.

Ein durch eine `openTRANSAP` referenziertes Geschäftsdokument kann beispielsweise von Geschäftspartner GP1 bis zu dreimal empfangen, aber von GP2 bis zu zehnmal gesendet werden.

Genauso kann es sein, dass GP1 das Geschäftsdokument bis zu zehnmal empfangen kann aber GP2 das Dokument nur dreimal sendet.

Hier muss im komponierten Prozessbeschreibung immer Kardinalität des empfangenden Geschäftspartners herangezogen werden. Verlangt der Sender eine höhere Empfangskardinalität, kann er einen Kompositions-AP bereitstellen, die der ProcessComposer in die Umsetzung einbauen muss, mit der er über die nicht Zustellbarkeit informiert wird.

Der Einfachheit halber werden sie erst bei der Umsetzung eines CPD in eine EPD – im ProcessTransformer – in Schleifenkonstrukte umgeformt.

DISPATCHNOTIFICATION und INVOICE

Bei DISPATCHNOTIFICATION (D) und INVOICE (I) sind zudem vier Kompositions-Varianten für die komponierte Prozessbeschreibung (CPD) möglich, die durch unterschiedliche Anordnungen in den PPDs entstehen.

1. Einkäufer sequentiell (bestimmte Reihenfolge) - Verkäufer parallel (D,I)

   Ein Einkäufer verlangt, dass DISPATCHNOTIFICATION und INVOICE in einer bestimmten Reihenfolge eintreffen (sequentiell).

   Beim Verkäufer sind D und I allerdings als parallele APs definiert, dass bedeutet, sie können in der vom Einkäufer gewünschten Reihenfolge auftreten, was aber nicht zwingend erforderlich ist.

   Umsetzung in der CPD:
   Hier müssen im komponierte Prozess die Prozesse der Geschäftspartner aufeinander abgestimmt werden, indem er erst auf den Eingang von D und I wartet durch den Verkäufer wartet, und sie dann in der gewünschten Reihenfolge des Einkäufers an diesen weitergibt.

2. Einkäufer sequentiell (D,I) Verkäufer sequentiell (D,I)

   Die Anordnung der APs stimmt überein.

   Umsetzung in der CPD:
   Die Anordnung wird übernommen.

3. Einkäufer sequentiell (D,I) Verkäufer sequentiell (I,D)

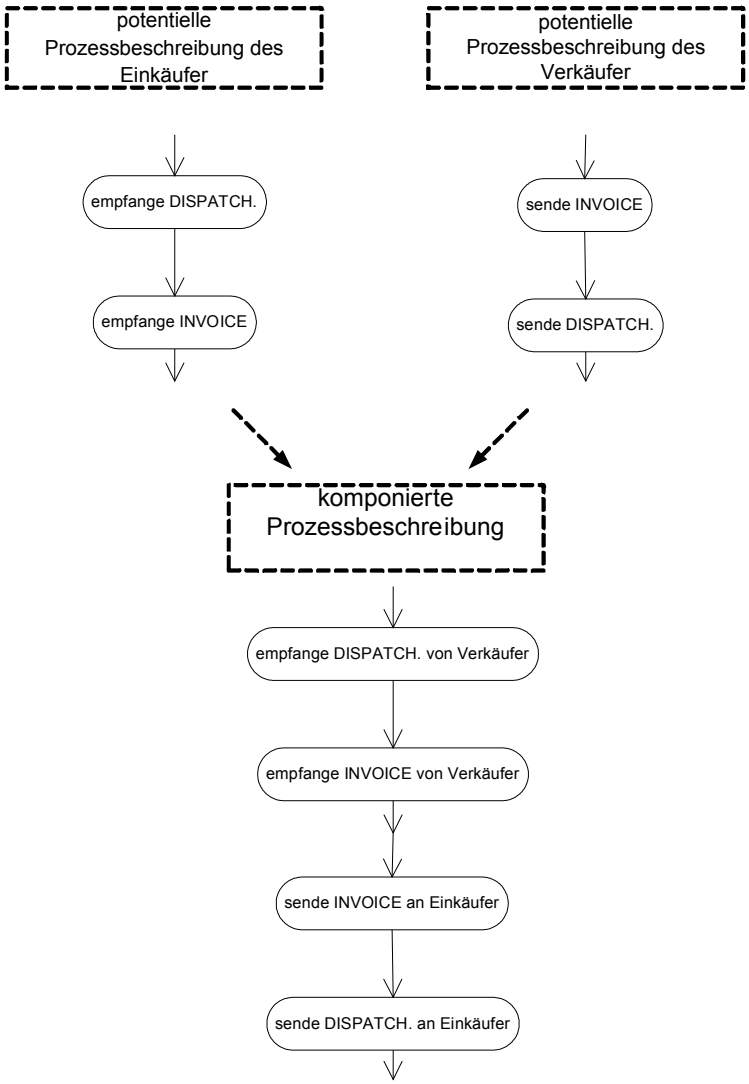

Abbildung 4.18 Komposition gegenläufiger Sequenzen

Dies ist kein Deadlock, da der Verkäufer I und D sendet, und der Einkäufer auf D und I wartet. (siehe auch Abbildung 4.18)

Umsetzung in der CPD:
Wiederum müssen der Eingang der beiden Dokumente des Verkäufers abgewartet werden, und dann erst weitergeleitet werden.

4. Einkäufer parallel (D,I) Verkäuferanordnung gleichgültig

Erlaubt der Einkäufer parallele Ausführung von D und I ist die Reihenfolge der APs des Verkäufers irrelevant.

Umsetzung in der CPD:
Die Anordnung des Verkäufers kann einfach übernommen werden, der Einkäufer nimmt die Dokumente in jeder Reihenfolge an.

Grundsätzlich lässt sich sagen, dass die Anordnung der APs des Empfängers – bei D und I handelt es sich dabei immer um den Einkäufer – maßgebenden Charakter für die Komposition besitzt.

Um einiges komplizierter würde die Analyse der PPDs werden, wenn in Kapitel 4.4.3 die Möglichkeit eine INVOICE zwischen DISPATCHNOTIFICATION und RECEIPTACKNOWLEDGEMENT einzuordnen nicht ausgeschlossen worden wäre.

4.6.9 Transformation der komponierten Prozessbeschreibung

Der ProcessTransformer setzt die eingehende CPD, beschrieben mit OWL-S, in eine EPD, beschrieben mit BPEL4WS, um. Dabei geht er die einzelnen `AtomicProcesses` (APs) des OWLS-Dokuments durch.

Die einzelnen APs beziehungsweise die zusammengesetzten `CompositeProcesses` (wie in Listing 4.5) werden samt der zu Ihnen gehörigen Informationen, wie Kardinalität und Kompensations-APs, in BPEL4WS umgesetzt.

Die Konstrukte Unordered und Sequence können ebenso direkt in Flow und Sequence umgewandelt werden.

Des weiteren wurden einige Umsetzungslösungen erarbeitet um APs mit bestimmter Kardinalität und bestimmten Status in BPEL4WS abzubilden.

Die Wichtigsten seien hier kurz erläutert. Es wurde aus Platz- und Verständnisgründen eine verkürzte Schreibweise für die BPEL4WS Konstrukte genutzt – sie sind so nicht in einer EPD nutzbar.

Timeouts

Um mit Timeouts arbeiten zu können muss statt des einfachen `receive` Elements das `pick–onMessage` Konstrukt genutzt werden, das auch ein `onAlarm` Konstrukt erlaubt, mit dem eine zeitliche Grenze angegeben werden kann, bis auf eine `onMessage` nicht mehr gewartet wird.

```
1.  <pick >
2.    <onMessage Empfange_GD1_Von_GP1>
3.      <sequence>
4.
5.        <invoke Sende_GD1_An_GP2>
6.
7.      </sequence>
8.    </onMessage>
9.
10.    <onAlarm for="PT30S">
11.     ...
12.    </onAlarm>
13.
14.  </pick>
```

Listing 4.6 Arbeiten mit Timeouts

WENN-DANN

Um ein WENN-DANN umzusetzen, wie es bei RFQ-QUTATION nötig ist, bedarf es eines „pick im pick“ (siehe Listing 4.8). Nachdem eine RFQ von Geschäftspartner GP1 (Zeile 2) empfangen und weitergeleitet wurde (Zeile 5), wird auf ein QUOTATION von GP2 gewartet, die im Folgenden an GP1 weitergeleitet wird.

```
1.  <pick >
2.    <onMessage Empfange_RFQ_Von_GP1
3.      <sequence>
4.
5.        <invoke Sende_RFQ_An_GP2>
6.
7.        <pick >
8.
9.          <onMessage Empfange_QUOTATION_Von_GP2
10.           <sequence>
11.
12.
13.             <invoke name="Sende_QUOTATION_An_GP1">
14.
15.           </sequence>
16.         </onMessage>
17.
18.         <onAlarm for="PT30S">
19.          ...
20.         </onAlarm>
21.       </pick>
22.
23.     </sequence>
24.   </onMessage>
25.
26.   <onAlarm for="PT30S">
27.    ...
28.   </onAlarm>
29.
30. </pick>
```

Listing 4.7 WENN - DANN

Kardinalitäten

Kardinalitäten können folgendermaßen abgebildet werden:

```
1. <variable name="MAXIMAL_ITERATION/>
2. <variable name="MOMENTANE_ITERATION"/>
3.
4. <while condition=
5.    "MAXIMAL_ITERATION > MOMENTANE_ITERATION" >
6.
7.   <pick >
8.     <onMessage Empfange_GD1_Von_GP1
9.       <sequence>
10.
11.        <assign >
12.           MOMENTANE_ITERATION ++
13.        </assign>
14.
15.        <invoke Sende_GD2_An_GP1>
```

```
16.
17.       </sequence>
18.     </onMessage>
19.     <onAlarm for="PT30S"> ...</onAlarm >
20.   </pick>
21.
22.</while>
```

Listing 4.8 Kardinalität in BPEL4WS

Solange die maximale Anzahl an Iterationen nicht erreicht ist wird die Message empfangen (Zeile 8) und weitergeleitet (Zeile 15)

Die maximale Kadinalität in Listing 4.9 ist größer 0, da sonst die Schleife gar nicht ausgeführt wird. Sie entspricht somit der Umsetzung einer AP mit minimaler Kardinalität von 1, – einem Geschäftsdokument mit WERDE-Status.

Sequentielle Anordnung von Geschäftsdokumenten

In einer sequentiellen Anordnung bestehen Abhängigkeiten zwischen den Geschäftsdokumenten, wie in Kapitel 4.4.3 bereits festgestellt.

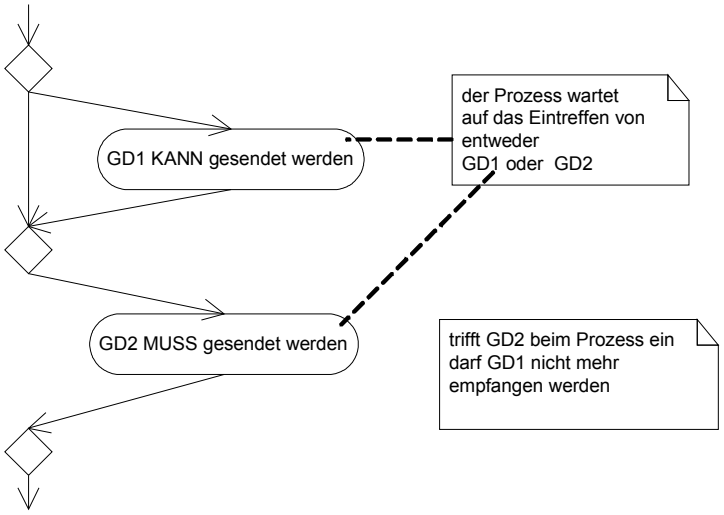

Abbildung 4.19 zwei KANN GDs gefolgt von einem MUSS GD

Bei Dokumenten mit KANN Status ist die Situation dadurch komplexer. Ein KANN Geschäftsdokument (GD1) darf dann nicht mehr empfangen werden, wenn das darauffolgende Geschäftsdokument (GD2) eintrifft (siehe Abbildung 4.19).

Die gleiche Situation tritt auf, wenn Geschäftsdokumente mit MUSS-Status bereits einmal empfangen wurden, aber noch öfter empfangen werden könnten.

Sie dürfen ebenfalls nicht mehr empfangen werden, wenn das folgende Geschäftsdokument eintrifft.

Dazu muss ein zweites Geschäftsdokument (GD2) ebenfalls in das `pick` mitaufgenommen werden, wie es in Listing 4.9 in Zeile 24 gezeigt wird.

In Zeile 31-34 wird die momentane Iteration von Geschäftsdokument 1 auf das maximum gesetzt. Dadurch kann sie nun nicht mehr empfangen werden, und der Prozess geht über zur Schleife mit Geschäftsdokument 2 in Zeile 44.

```
1. <variable name="MAXIMAL_ITERATION_GD1/>
2. <variable name="MOMENTANE_ITERATION_GD1"/>
3.
4. <variable name="MAXIMAL_ITERATION_GD2/>
5. <variable name="MOMENTANE_ITERATION_GD2"/>
6. ...
7. <while condition=
8.    "MAXIMAL_ITERATION_GD1 >
9.     MOMENTANE_ITERATION_GD1" >
10.
11.   <pick >
12.
13.     <onMessage Empfange_GD1_Von_GP1
14.       <sequence>
15.
16.         <assign >
17.           MAXIMAL_ITERATION_GD1++
18.         </assign>
19.         <invoke "Sende_GD1_An_GP2">
20.
21.      </sequence>
22.     </onMessage>
23.
24.     <onMessage Empfange_GD2_Von_GP2
25.       <sequence>
26.
27.         <assign >
28.            MOMENTANE_ITERATION_GD2++
29.         </assign>
30.
31.         <assign >
32.            MOMENTANE_ITERATION_GD1 =
33.            MAXIMAL_ITERATION_GD1
34.         </assign>
35.
36.         <invoke name="Sende_GD2_An_GP1">
37.
38.       </sequence>
39.     </onMessage>
40.     <onAlarm for="PT30S"> ...</onAlarm >
41.
42.   </pick>
43. </while>
44. <while condition=
45.    "MAXIMAL_ITERATION_GD2 >
46.      MOMENTANE_ITERATION_GD2" >
47.   <pick >
48.     <onMessage Empfange_GD2_Von_GP2
49.       <sequence>
50.
51.         <assign >
52.            MOMENTANE_ITERATION_GD2++
```

```
53.         </assign>
54.
55.         <invoke Sende_GD2_An_GP1>
56.       </sequence>
57.     </onMessage>
58.     <onAlarm for="PT30S"> ...</onAlarm >
59.   </pick>
60.</while>
```

Listing 4.9 Abbruch des Wartens auf ein Geschäftsdokument

Noch komplexer ist die Situation, wenn mehrere KANN Geschäftsdokumente aufeinander folgen.

Beispielsweise könnten DISPATCHNOTIFICATION, RECEIPTACKNOWLEDGEMENT und INVOICE sequentiell angeordnet sein, besitzen aber alle KANN-Status. Wird nun eine INVOICE gesendet, dürfen auch die beiden vorhergehenden Geschäftsdokumente nicht mehr gesendet werden.

Parallele Anordnungen von Geschäftsdokumenten

In parallel angeordneten Geschäftsdokumenten treten diese Schwierigkeiten nicht auf. Die Dokumente in einem Flow sind nicht von einander abhängig. Trifft ein Geschäftsdokument aus dem Flow ein, dürfen die anderen darin enthaltenen immer noch empfangen werden.

Sie dürfen nur dann nicht mehr empfangen werden, wenn das `Unordered` Konstrukt (OWLS) bzw. das `flow` Konstrukt (BPEL4WS), in dem ihre APs enthalten sind in einer `sequence` liegt. Trifft das nächstfolgende Dokument aus der `sequence` nach dem `flow` ein, dürften dann keine Dokumente aus dem `flow` mehr empfangen werden, sofern es sich nicht um MUSS Geschäftsdokumente handelt. Diese Anordnung kommt nach der openTRANS-MPD allerdings nicht vor.

Gegenläufige Anordnungen von Dokumenten

Die oben beschriebenen Szenarien sind allerdings nur so umsetzbar, wenn beide potenziellen Prozessbeschreibungen gleiche Anordnungen vorschreiben.

Viele Unterschiede sind in openTRANS nicht möglich, aber bei DISPATCHNOTIFICATION und INVOICE kann parallele auf sequentielle Anordnung treffen oder sogar in verschiedenen Reihenfolgen, wie im Kapitel 4.6.8 bereits festgestellt wurde. In dieser Situation muss bei der Komposition zuerst der EmpfangsAP und dann der SendeAP eines Geschäftsdokuments umgesetzt werden.

Kardinalitäten höher eins wurden bei einer solchen Konstellation in diesem Lösungskonzept nicht berücksichtigt, da INVOICE und DISPATCHNOTIFICATION laut Anforderungen eine maximale Kardinalität von 1 besitzen (siehe Kapitel 4.4.3).

In folgendem Beispiel sendet der Verkäufer DISPATCHNOTIFICATION und INVOICE in einer Sequenz dieser Reihenfolge. Der Einkäufer erwartet sie in umgekehrter Reihenfolge (siehe auch Abbildung 4.18). Eine Umsetzung sieht hier wie folgt (Listing 4.10) aus.

```
1. <variable name="MAXIMAL_ITERATION_INV/>
2. <variable name="MOMENTANE_ITERATION_INV"/>
3.
4. <variable name="MAXIMAL_ITERATION_DIS/>
5. <variable name="MOMENTANE_ITERATION_DIS"/>
6.
7. <sequence>
8.
9. <sequence>
10.  <while condition=
11.     "1 >
12.      MOMENTANE_ITERATION_DIS" >
13.
14.    <pick >
15.      <onMessage Empfange_DIS_Von_GP2>
16.           ...
17.      </onMessage>
18.    </pick>
19.
20.  </while>
21.
22.  <while condition=
23.     "1 >
24.      MOMENTANE_ITERATION_INV" >
25.
26.    <pick >
27.      <onMessage Empfange_INV_Von_GP2>
28.           ...
29.      </onMessage>
30.    </pick>
31.
32.  </while>
33.</sequence>
34.
35.<sequence>
36.  <invoke Sende_INV_An_GP1>
37.  <invoke Sende_DIS_An_GP1>
38.</sequence>
39.
40.</sequence>
```

Listing 4.10 Umsetzung gegenläufiger Sequenzen

5 IMPLEMENTIERUNG DES PROTOTYPS

Mit dem hier beschriebenen Prototypen wird grundlegend gezeigt, dass die Generierung einer ausführbaren Prozessbeschreibung aus den potenziellen Prozessbeschreibung der Geschäftspartner, wie sie im Lösungskonzept vorgestellt wurde, realisierbar ist.

Statt der komponentenbasierten Lösung wird eine integrierte Lösung angestrebt. Die Erstellung der CPD wurde übergangen. Es wird aus den potenziellen Prozessbeschreibungen sofort eine ausführbare Prozessbeschreibung generiert.

Alle Informationen, die zur Generierung einer ausführbaren Prozessbeschreibung benötigt werden, werden den OWL-S Dokumenten der Geschäftspartner entnommen und in BPEL4WS Konstrukte umgesetzt.

Komponiert werden in der momentanen Version sequentiell angeordnete Prozessbeschreibungen mit 1er Kardinalitäten, deren generierte EPD und die dazugehörigen Installationsdateien vom Integrator ausgegeben und automatisch auf dem Collaxa BPEL Server deployed werden.

Trotz der Einschränkungen des Prototyps wurden die Erweiterungspotentiale im Auge behalten. So weit möglich wurde darauf geachtet, eine Erweiterung des Prototypen auf das eigentliche Lösungskonzept möglich zu machen.

5.1 Systemumgebung

Die Entwicklungs- und Testumgebung setzt sich zusammen aus:

- Java SDK 1.4.2
- Java SDK EE 1.3.1
- Collaxa BPEL Server 2.0 rc5
- Apache Tomcat 4.0
- Apache Axis 1.1 rc2

Das Integrationssystem benötigt zum erfolgreichen Einsatz die in Kapitel 4.5.4 erwähnten Java APIs für semantische Technologien, sowie alle APIs, die in gepackter Version (.jar-Dateien) in der Axis Distribution mitgeliefert werden. Dazu gehören vor allem WSDL4J und SOAP4J.

5.2 Systemübersicht

Die genutzten Komponenten wurden bereits vorgestellt. Es handelt sich um den Collaxa BPEL Server und die TOMCAT Plugin Variante des Apache Axis Web Service Servers.

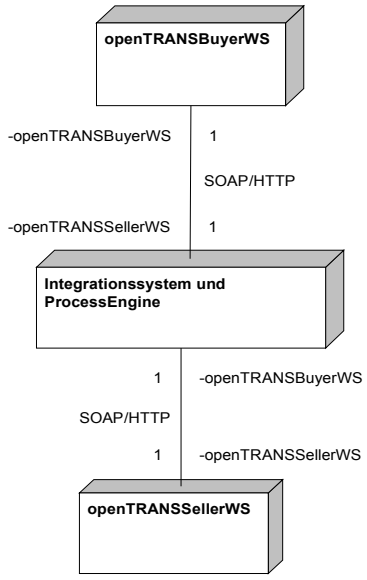

Abbildung 5.1 Systemübersicht des Prototyps

Abbildung 5.1 zeigt eine Übersicht des Prototyps als Deployment Diagramm. Auf dem mittleren Knoten stehen das entwickelte openTRANS-Integrationssystem sowie der ProcessEngine, auf dem die generierte ausführbare Prozessbeschreibung als Flow-Service installiert wird (siehe Abbildung 5.2). Sie kommunizieren mit den Knoten der Geschäftspartner per SOAP/HTTP.

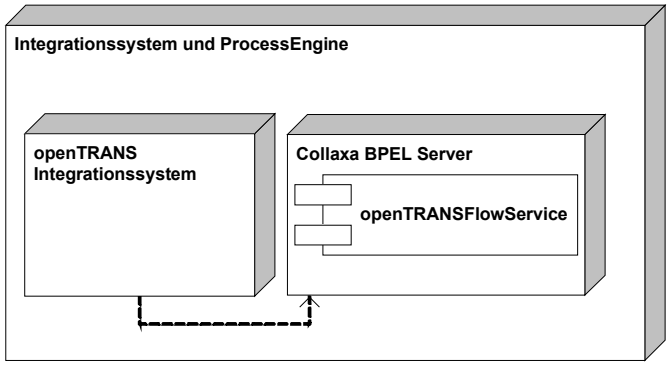

Abbildung 5.2 Integrationssystem und ProcessEngine

Die openTRANS-Web Services der Geschäftspartner bestehen jeweils zum einen aus einem auf Axis installierten Web Service und einem auf dem Collaxa BPEL Server installierten Prozess. In Abbildung 5.3 ist der openTRANS-Web Service des Einkäufers beispielhaft dargestellt.

Die AXIS Services dienen ausschließlich als Lieferanten der Geschäftsdokumente.

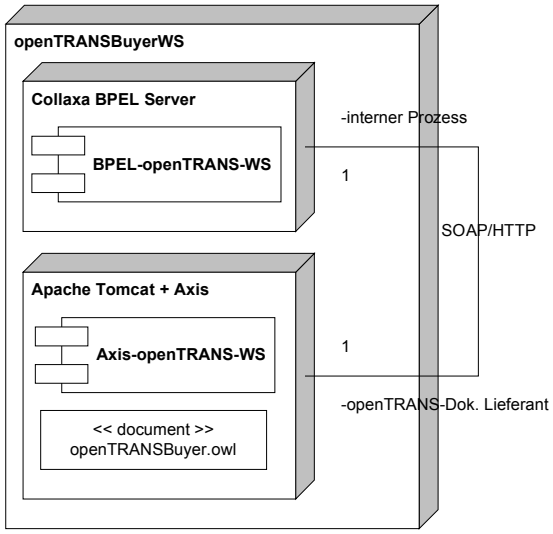

Abbildung 5.3 openTRANSBuyerWS des Prototyps

Der BPEL-openTRANS-WS ist ein Web Service, der aus einer ausführbaren Prozessbeschreibung auf dem Collaxa BPEL Server deployed wurde. Er dient als Repräsentation des internen Prozesses, den der Geschäftspartner in seiner potenziellen Prozessbeschreibung beschreibt. Diese stellen die eigentlichen Web Services dar, die mit dem generierten Flow-Service kommunizieren. Diese Prozesse setzen die potenziellen Prozessbeschreibungen ihres Systems um und dienen so als Repräsentationen deren internen Prozesse.

Zudem wird so das Sitzungs-Management des Collaxa BPEL Servers über WS-Addressing genutzt. Prozesse, die auf Collaxa installiert sind, nutzen automatisch WS-Addressing, um Nachrichten den einzelnen Prozessinstanzen zuordnen zu können.

5.3 ProcessIntegrator

Für den ProzessIntegrator wurden drei Interfaces kreiert, die die eigentliche Integratorklasse `OWL2BPELProcessIntegrator` implementiert, wie in Abbildung 5.4 dargestellt.

Alle Implementationen von `IProcessIntegratorFileBased` zeichnen sich durch eine Ausgabe der erstellten Prozessbeschreibung in Dateien aus. Durch die Implementation von `IProcessIntegratorFileBased` werden Methoden zur Verfügung gestellt, mit der der Ausgabeort des ProcessTransformers gesteuert werden kann.

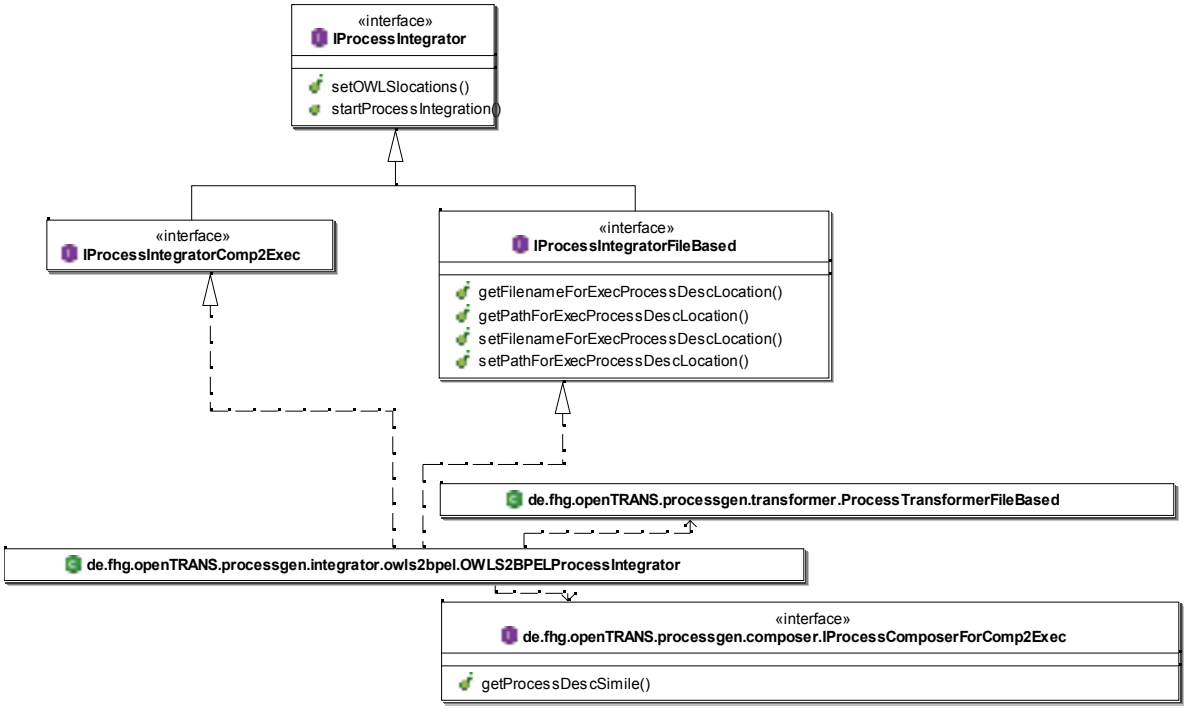

Abbildung 5.4 de.fhg.openTRANS.processgen.integrator

Das methodenlose Interface `IProcessIntegrationComp2Exec` wird implementiert, um darzustellen, dass dieser ProcessIntegrator die integrierte Variante des Integrationssystems darstellt, und nicht die komponentenbasierte.

In Abbildung 5.4 ist veranschaulicht, dass ein ProcessIntegrator einen ProcessComposer und einen ProcessTransformer besitzt. Da der ProcessIntegrator vom Typ `IProcessIntegratorFileBased` ist, referenziert er einen `ProcessTransformerFileBased`.

Durch die Implementation des `IProcessIntegrationComp2Exec` Interface wird verdeutlicht, dass er einen ProcessComposer vom Typ `IProcessIntegration-Comp2Exec` nutzt. Dieser generiert keine komponierte Prozessbeschreibung, sondern wird vom ProcessIntegrator statt einer solchen an den ProcessTransformer weitergegeben. Wie der ProcessComposer dann direkt die Erstellung der ausführbaren Prozessbeschreibung steuert („Comp2Exec"), wird in Kapitel 5.5 gezeigt.

Im Konstruktor des `OWLS2BPELIntegrators` werden die URIs der OWL-S-Dokumente von Ein- und Verkäufer mitgegeben. Diese werden an den ProcessComposer weitergereicht.

Über die Methoden `setFilenameForExecProcessDescLocation` und `setPath-ForExecProcessDescLocation` wird der Ausgabeort der ausführbaren Prozessbeschreibung festgelegt.

Mit `startProcessIntegration` wird der Integrationsprozess eingeleitet. Dieser Aufruf wird an den ProcessTransformer weitergegeben (ProcessTransformer siehe Kapitel 5.4).

Nach Beendigung von Komposition und Transformation der ausführbaren Prozessbeschreibung, generiert der ProcessIntegrator die Dateien, die zur Installation des Flow-Service auf dem Collaxa BPEL Server nötig sind. Danach stösst er die Installation an. Siehe dazu auch Kapitel 5.6.

5.4 ProcessTransformer und komponierte Prozessbeschreibung

Der ProcessTransformer nimmt die Java Repräsentation (`IProcessDescription`) einer komponierten Prozessbeschreibung auf, und übergibt sie einem Parser `IProcessParser`, der aus der Java Repräsentation der komponierten Prozessbeschreibung eine Java Repräsentation der ausführbaren Prozessbeschreibung (`IExecutableProcessDescription`) generiert.

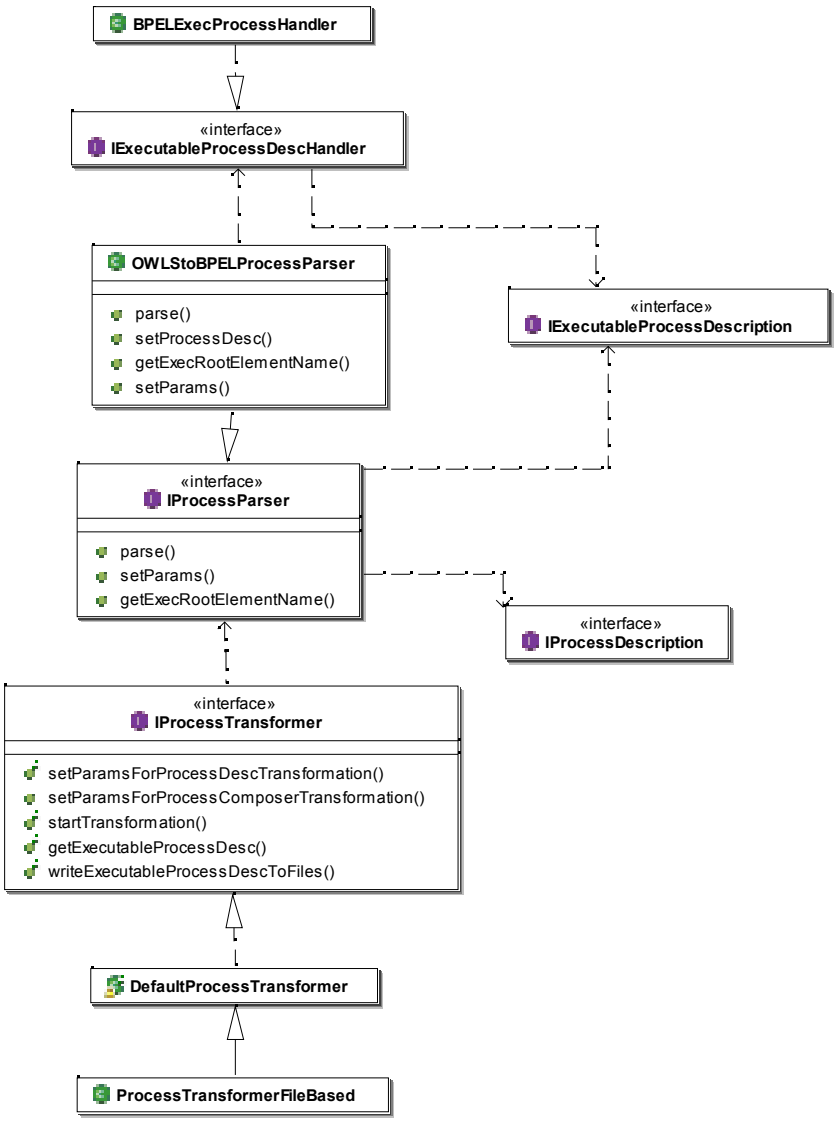

Abbildung 5.5 de.fhg.openTRANS.processgen.transformer

Die hier verwendete Implementation des ProcessTransformer (`ProcessTransformerFileBased`) leitet sich vom `DefaultProcessTransformer` ab, der das Interface des `IProcessTransformers` implementiert. `DefaultProcessTransformer` ist eine abstrakte Klasse, die die Grundfunktionalitäten des ProcessTransformer bietet.

Abbildung 5.5 ist zu entnehmen, welche Klassen ein ProcessTransformer benötigt, um die Erstellung einer ausführbaren Prozessbeschreibung durchzuführen. Dazu gehören das `IProcessDescription`-Objekt, der korrekte `IexecutableProcessDescHandler`, der Methoden bereitstellt, um die XML-Struktur der ausführbaren Prozessbeschreibung zu generieren, und ein Parser `IProcessParser`, der die eigentliche Umsetzung von komponierter Prozessbeschreibung in ausführbare Prozessbeschreibung regelt. Der `IProcessParser` erstellt dann die `IexecutableProcessDescription`.

Die BPEL-Konstrukte

Für die Generierung der einzelnen BPEL-Konstrukte, aus der sich die ausführbare Prozessbeschreibung zusammensetzt, wurden Klassen erstellt, die die einzelnen Konstrukte in Java repräsentieren.

Sie sind im Package `de.openTRANS.processgen.java.bpel` zu finden, zu dem ein Überblick in Abbildung 5.6 gegeben wird.

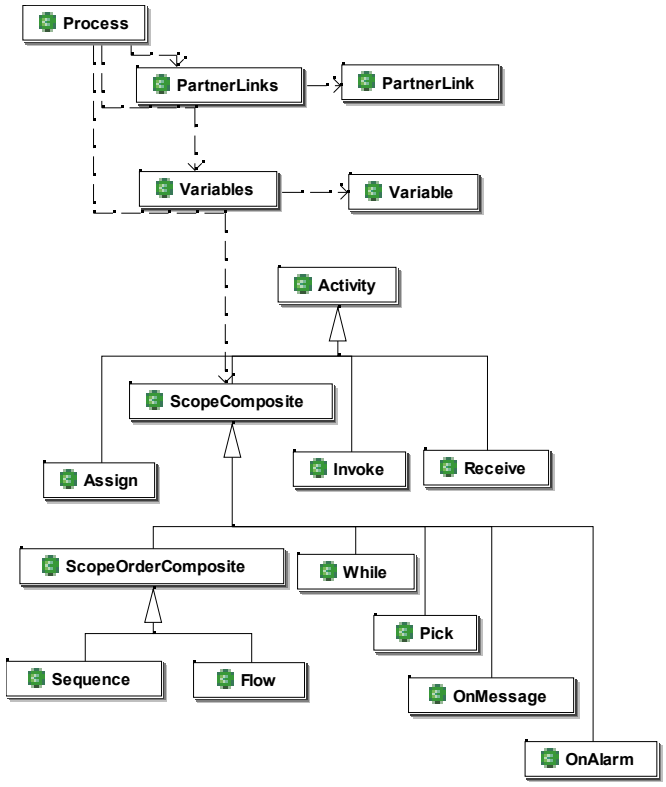

Abbildung 5.6 de.fhg.openTRANS.processgen.java.bpel

Ein ausführbarer `Process` besteht aus `Variables`, `PartnerLinks` und einem `ScopeComposite`, das die eigentliche Prozessbeschreibung enthält. Alle Klassen dieser Prozessbeschreibung leiten sich von der Superklasse `Activity` ab. Zudem wurde nach dem Composite Pattern [Go4] eine Subklasse `ScopeComposite` erstellt, die als einzige weitere `Activities` enthalten darf. Typische Subklassen von `ScopeComposite` sind `Sequence` und `Flow`, die den gleichnamigen Kontrollflusskonstrukten in BPEL4WS entsprechen.

Jede dieser Klassen besitzt ihren eigenen Serializer im Package `de.fhg.openTRANS.processgen.java.bpel.sax`.
Jeder Serializer leitet sich vom `AbstractBaseSerializer` des Fraunhofer Sax-Package ab, das die eigentliche Serialisierung durchführt. Diese Beziehung ist beispielhaft für die Klasse Process in Abbildung 5.7 dargestellt.

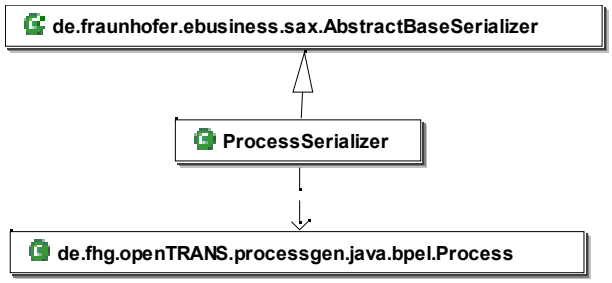

Abbildung 5.7 BPEL-Serializer – de.fhg.openTRANS.processgen.sax.bpel

Der ProcessParser

Der hier genutzte `IProcessParser` wird durch `OWLStoBPELProcessParser` implementiert.

Der `BPELExecProcessHandler` liefert nur Methoden, wie `addPartnerInvoke`, `addPartnerReceive`, `addScope` und `endScope`. Mit diesen Methoden wird die XML-Struktur des BPEL4WS-Dokuments systematisch durch den `IProcessParser` aufgebaut.

Beispielsweise wird eine neue `Sequence` mit `addScope(...)` erzeugt, in die ein `Receive` durch `addPartnerReceive(...)` hinzugefügt wird, woraufhin die `Sequence` wieder geschlossen wird: `endScope(...)`.

Der ProcessTransformer erhält über die `setTransformationParams` Methode den `IProcessParser` und die `SerializerFactory`, mit der die einzelnen Konstrukte des BPEL konstruiert werden können.

In der `IProcessDescription` sind zwei `PreparedPartnerLink`-Objekte enthalten, die die Informationen der WSDL-Dokumente und der OWLS-Beschreibung enthalten, die zur Generierung von Kommunikationskonstrukten, wie `Invoke` und `Receive` bzw. `onMessage`, dieses Geschäftspartners für die ausführbare Prozessbeschreibung benötigt werden, wie Abbildung 5.8 zeigt.

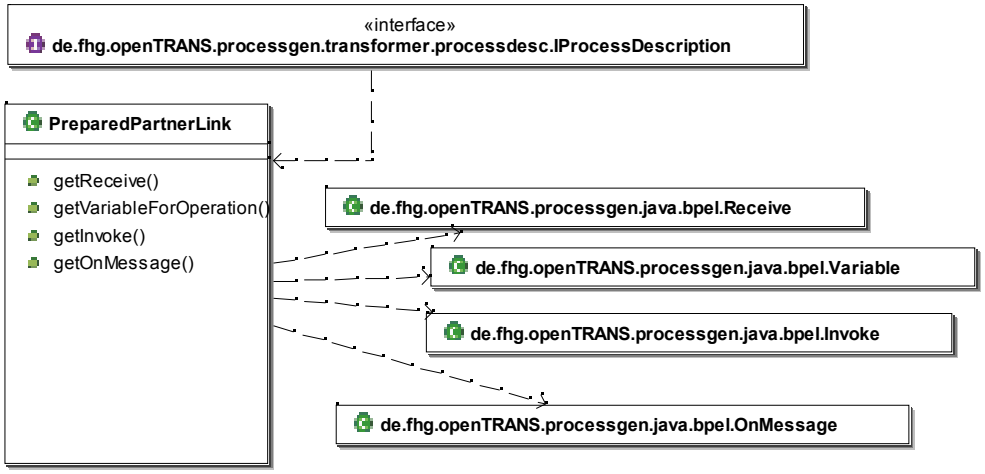

Abbildung 5.8 de.fhg.openTRANS.processgen.transformer.helper

Der Transformer gibt diese `PreparedPartnerLink` Instanzen an den `IProcess-Parser` über die Methode `setParams` weiter.

Mit Hilfe dieser Objekte erstellt der `IProcessParser` `Activity`-Objekte, die er über `addPartnerReceive` und `addPartnerInvoke` an den `BPELExecProcess-Handler` weitergibt.

Abweichung vom Lösungskonzept

Da in der Entwicklung des Prototyps von der Erstellung einer komponierten Prozessbeschreibung, wie im Lösungskonzept beschrieben, abgesehen wurde, und direkt vom ProcessComposer eine ausführbare Prozessbeschreibung generiert werden soll, wurde der ProcessTransformer erweitert.

Er stellt nun bei seiner Initialisierung die Möglichkeit bereit, den ProcessComposer selbst als Nutzer des `IExecutableProcessDescHandler` mitzugeben.

So kann der ProcessComposer selbst und unmittelbar die Methoden zur Erstellung der XML-Strukur, wie `addScope` u.ä. nutzen. Beispielhaft ist die Erstellung einer einfachen Prozessbeschreibung in Abbildung 5.9 dargestellt.

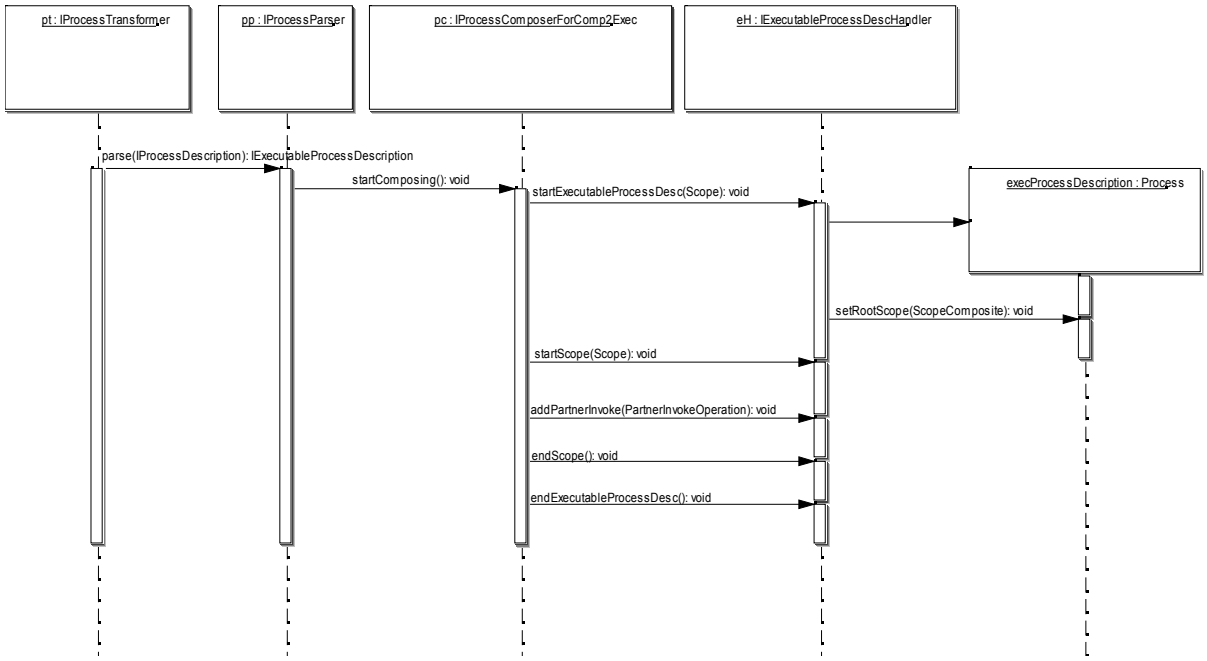

Abbildung 5.9 ProcessTransformer - Konstruktion der EPD

5.5 ProcessComposer und potenzielle Prozessbeschreibung

Durch das Interface `IProcessComposerComp2Exec` wird verdeutlicht, dass der ProcessComposer keine komponierte, sondern direkt eine ausführbare Prozessbeschreibung erstellt.

Der ProcessComposer hält zwei Referenzen auf `OWLHandler` Instanzen, die die OWL-S-Prozessbeschreibungen von Einkäufer und Verkäufer repräsentieren.

In einem `OWLHandler` wird ein OWL-S-Dokument und der darin enthaltenden potenzielle Prozessbeschreibung mit Hilfe der OWL-S API eingelesen (siehe `org.mindswap.owls.process.Process` in Abbildung 5.6), und danach unter Verwendung der Jena API (siehe `com.hp.hpl.jena.ontology.OntModel` in Abbildung 5.6), auf die Konstrukte der openTRANSProcess.owl hin untersucht.

Die OWLS-Konstrukte

Beim Einlesen werden die einzelnen gelesenen OWL-S openTRANSAPs in `Activity`–Instanzen umgesetzt, die die Informationen über die openTRANSAPs halten, wie zum Beispiel welchen openTRANS-Typ (RFQ,ORDER,...), den Status den sie besitzen (MUST/MAY), welche Kardinalität, welchen Timeout usw. (siehe Abbildung 5.10).

Eine `Activity` entspricht entweder einer Web Service-Sende- oder einer Empfangsoperation für ein Geschäftsdokument. Das hängt davon ab, für welchen Geschäftspartner der `OWLHandler` eingesetzt wird, der die `Activity` erhält.

So entspricht eine `Activity` vom openTRANS-Type RFQ im `OWLHandler` eines Einkäufers einer Sendeoperation, im `OWLHandler` eines Verkäufers einer Empfangsoperation.

Auch hier wird wieder das Composite Pattern[Go4] verwendet, um die `Activity` Instanzen in einzelnen `ControlScopes` einbinden zu können. Ein `ControlScope` ist entweder vom Typ `ControlSequence` oder `ControlFlow` (siehe Abbildung 5.10).

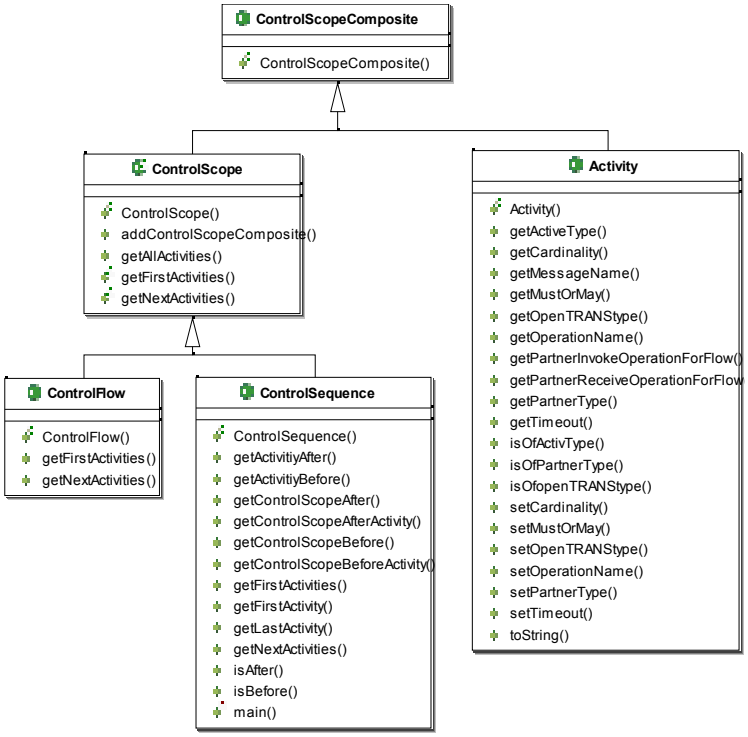

Abbildung 5.10 de.fhg.openTRANS.processgen.composer.activities

Mit `ControlScope.getNextActivities` wird ein Iterator simuliert, der immer die nächsten `Activity`–Instanzen nach den zuletzt ausgegebenen zurückgibt, die in dem `ControlScope` enthalten sind. Bei einer `ControlSequence` ist dies immer nur eine `Activity`, bei einem `ControlFlow` können dies mehrere sein.

Übermittlungskonstrukte von ProcessComposer zum ProcessTransformer

Aus den `Activity` Instanzen gewinnt der ProcessComposer im Verlauf der Komposition die Informationen, die später in BPEL4WS gebraucht werden.

Um diese Informationen an den `IExecutableProcessDescHandler` weitergeben zu können, werden sie von den `Activity` Instanzen auf andere Klassen übertragen, die immens an die einzelnen `Receive` und `Invoke` Element in BPEL4WS erinnern (siehe Abbildung 5.11).

Entsprechend gibt es eine `PartnerInvokeOperation` und eine `PartnerReceiveOperation` Klasse (Subklassen von `PartnerOperation`). Zudem noch eine `MessageActivity` Klasse, die Instanzen beider Klassen enthält.

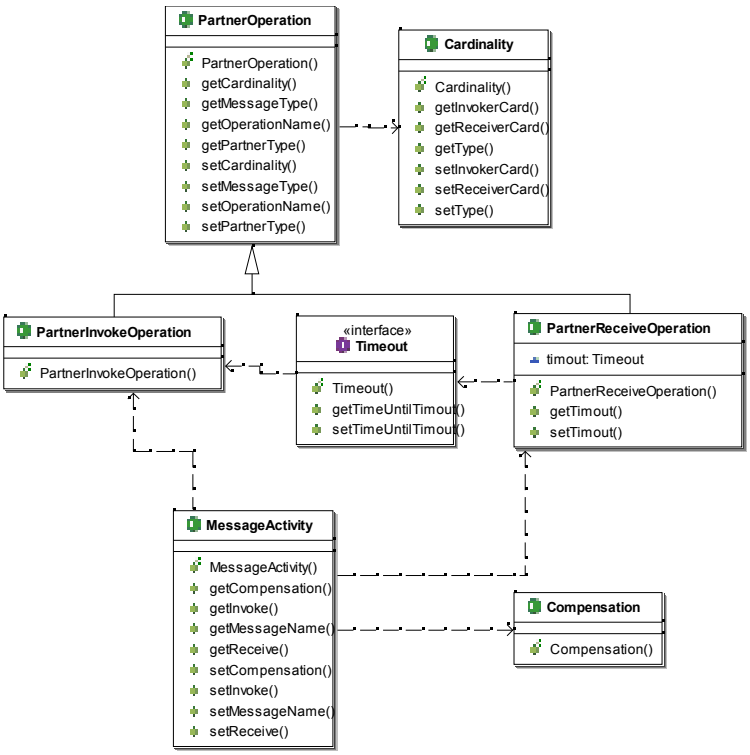

Abbildung 5.11 de.fhg.openTRANS.processgen.activities

`MessageActivity` wird genutzt, wenn eine `Receive` und eine `Invoke` Operation für ein Geschäftsdokument direkt hintereinander ausgeführt werden können, wie es in sequentieller Anordnung meist der Fall ist.

`PartnerOperation`–Instanzen enthalten zudem die `Cardinality` ihrer Ausführung, und gegebenenfalls eine `Compensation`. Eine `PartnerReceiveOperation` enthält zudem noch einen `Timeout`.

In einem `Timeout` ist die Dauer enthalten, nach der der `Timeout` eintritt, und die `PartnerInvokeOperation`, die gerufen werden soll, wenn der `Timeout` eintritt.

Dabei wird aus jeweils einer `Activity` Instanz, je nach dem ob sie als Empfangs- oder als Sende-Aktivität für ein Geschäftsdokument gilt, eine `PartnerInvoke-Operation` oder eine `PartnerReceiveOperation` konstruiert.

Wobei aus `Activities`, die der Geschäftspartner als Empfangsoperation nutzt, eine `PartnerInvokeOperation` kreiert wird und umgekehrt, da es sich um die Konstrukte der ausführbaren Prozessbeschreibung handelt. Das bedeutet, dass eine `Activitiy`, die dem Geschäftspartner als Empfangs `Activity` dient, in der EPD als `Invoke` umgesetzt werden muss.

Ablauf der Komposition

Über `startComposing` wird die Komposition der beiden potenziellen Prozessbeschreibungen, die in den `OWLSHandlers` enthalten sind, gestartet.

Nachdem der ProcessComposer alle Bedingungen überprüft hat, die für eine Komposition nötig sind, wie zum Beispiel, dass alle `MUSTopenTRANSAPs` ein entsprechendes Gegenstück besitzen, beginnt er mit der eigentlichen Komposition.

Dazu holt er sich über `getNextActivities` immer die nächsten `Activities` eines `OWLHandlers` und vergleicht diese mit den `getNextActivities` des Geschäftspartners.

Hat der ProcessComposer zwei korrespondierende `Activities` gefunden, entscheidet er wie und wann sie zu ausführbaren Prozessbeschreibung hinzugefügt werden. Entsprechend gibt er die `MessageActivitiy` oder jeweils eine `PartnerReceiveOperation` und eine `PartnerInvokeOperation` an den `IExecutableProcessDescHandler` weiter, der im Kapitel 5.4 erläutert wurde.

Der Prototyp beschränkt sich bisher auf die Umsetzung der sequentiellen Activities. Bei sequentiell angeordneten `Activities` treffen immer nur zwei `Activities` der Geschäftspartner aufeinander. Diese entsprechen einander als Sende- und Empfangs-openTRANSAP, oder sind von unterschiedlichen openTRANS-Typen.

1. Im einfachsten Fall erhält er von jedem `OWLHandler` nur eine `Activity` und diese stellen ihre jeweiligen Gegenstücke als Sende- und Empfangsoperation dar. Dann kann der ProcessComposer aus den `Activities` einzelne `PartnerInvokeOperation` und `PartnerReceiveOperation` -Instanzen kreiren, in einer `MessageActivity` bündeln, und an den ProcessTransformer weitergeben
2. Die `OWLHandler` beider Geschäftspartner (GP1, GP2) liefern `Activities`, die nicht ein Sende- Empfangs- openTRANSAP Paar darstellen. Da MPD einen festgelegten Ablauf vorgibt, keine Deadlocks vorhanden sein können und alle MUSTopenTRANSAPs ein Gegenstück beim Geschäftspartner besitzen, können in diesem Fall nur zwei Situationen vorliegen:
    1. Es liegt bei einem der Geschäftspartner eine MAYopenTRANSAP vor, die beim Gegenüber keine korrespondierende openTRANSAP besitzt.
    2. Es liegt die gegenläufige Anordnung bei INVOICE und DISPATCHNOTIFICATION vor, wie in Kapitel 4.6.8 beschrieben.

    In der ersten Situation kann eine Empfangs-openTRANSAP vernachlässigt werden; ein Sende-openTRANSAP wird zwar empfangen, aber nicht weitergeleitet.
    In der zweiten Situation muss zuerst das Empfangen, dann das Senden der Geschäftsdokumente abgewickelt werden , wie in Kapitel 4.6.8 erläutert.

Bei der Verarbeitung paralleler Activities wird genauso vorgegangen, mit dem einzigen Unterschied, dass von einem `OWLHandler` mehrere Activities auf einmal geliefert werden können, da die Reihenfolge ihrer Ausführung beliebig ist.

5.6 ProcessEngine und generierter BPEL Prozess

Der ProcessIntegrator generiert alle Dateien, die zur Installation des Flow-Service auf der Process Engine benötigt werden. Neben der eigentlichen Prozessbeschreibung – in diesem Fall der openTRANS.bpel – sind dies bei dem verwendeten Collaxa BPEL Server eine build.xml, eine bpel.xml und drei WSDL Dateien für jeweils einen Partner und den Flow-Service.

Sind diese Dateien vorhanden, wird vom ProcessIntegrator nur noch die Collaxa Batch Datei „cxant.bat" in dem Ordner aufgerufen, in dem die Installationsdateien hinterlegt wurden.

WSDL Dokumente der Partner

Die openTRANS-Web Services der Geschäftspartner besitzen bereits eine WSDL Beschreibung. Allerdings wird für ihre Einbindung in den BPEL4WS Prozess noch die Beschreibung ihrer partnerLinkTypes benötigt, die dann in der openTRANS.bpel Datei als partnerLinks referenziert werden können.

Dazu werden die originalen WSDL-Dokumente der Partner jeweils in den generierten WSDL-Dokumenten importiert und die Beschreibung der partnerLinkTypes, die die möglichen Rollen und ihrer zugehörigen WSDL portTypes festlegen, hinzugefügt.

WSDL Dokument des Flow-Service

In der WSDL Datei des Flow-Service werden die Web Service Operationen, die der Flow-Service aufgrund der potenziellen Prozessbeschreibungen der Partner nach außen hin darbieten soll definiert. Zudem enthält er auch noch eine PartnerLink-Beschreibung, damit er wiederum in eine BPEL4WS-Prozessbeschreibung miteingebunden werden könnte.

build.xml

Die build.xml dient der Verwendung der Apache Ant Build API[41] In einer Ant konformen build.xml können einzelne Schritte definiert werden, wie eine Installation abzulaufen hat. Dazu gehören beispielsweise Kompilieren, Kopieren, Verschieben und anderes.

Basiskommando in build.xml ist der Aufruf der „bpelc.bat", in der ein Java Programm Collaxas gestartet wird,mit dessen Hilfe die openTRANS.bpel Datei, die Prozessbeschreibung, kompiliert wird. Dabei werden die einzelnen Structured und Primitive Activities von BPEL4WS in einzelne Java-Klassen übersetzt, die den aufrufbaren Flow-Service darstellen.

Danach werden die generierten Klassen zusammen mit den anderen Installationsdateien in einer .jar-Datei zusammengepackt und in den deploy Ordner des Collaxa BPEL Servers verschoben. Der gestartete Collaxa Server überwacht diesen Ordner und installiert den Prozess automatisch.

41 http://ant.apache.org/

bpel.xml

Die bpel.xml Datei wird bei der Generierung der Klassen des Flow-Service benötigt. In ihr ist vor allem festgelegt, welchem partnerLink in openTRANS.bpel welches Partner WSDL-Dokument entspricht. Über diese Verknüpfung können bei der Erstellung des Flow-Service erst die receive und invoke Activities der openTRANS.bpel in Web Service Operationsaufrufe umgewandelt werden.

5.7 Architekturübergreifende Aspekte

Das System erhält über eine .properties-Datei alle Informationen, die sie zum Betrieb benötigt. Darin ist zum Beispiel der Pfad zur openTRANSProcess.owl enthalten, sowie die zu nutzenden Bezeichnungen von Buyer und Seller PartnerLinkTypes und PartnerLinks.

Der Prototyp funktioniert kommandozeilenbasiert. Bei Aufruf der Klasse werden dem System die URIs der potenziellen Prozessbeschreibungen der Geschäftspartner und der Pfad der Ausgabe der generierten Dateien mitgegeben.

Darauhin wird der Integrationsprozess automatisch gestartet, und die ausführbare Prozessbeschreibung automatisch in Collaxa deployed.

Nach Abschluss der Integration wird der Prozess mit Hilfe des Collaxa BPEL Server gestartet. Dieser gibt eine Meldung an den referenzierten Buyer Web Service, der daraufhin den Prozess startet.

In der FlowAnsicht des BPEL Servers ist der ablaufende Prozess durch Reload des Browsers zu beobachten. Am Ende des Prozessablaufs werden in der

Der Buyer Web Service sendet sein erstes Geschäftsdokument an den Collaxa Server, der dieses an den Seller Web Service weiterleitet, wodruch auch dessen Prozess

6 FAZIT /AUSBLICK

Ergebnisse

In dieser Arbeit wurden ein Konzept vorgestellt, dass die dynamische Prozessintegration der e-Procurement-Systeme von Einkäufer und Verkäufer zur Ausführung von Markttransaktionen ermöglicht. Diese e-Procurement-Systeme präsentieren sich nach außen als Web Services, die den Austausch von openTRANS-Geschäftsdokumenten erlauben (openTRANS-Web Services).

Für eine Integration müssen sich die Schnittstellen der Web Services hinsichtlich ihrer Bedeutung beschreiben, damit das Integrationssystem erkennen kann, welche Operationen für die Übermittlung welcher Geschäftsdokumente gedacht sind. Hinzukommend müssen detaillierte potenzielle Prozessbeschreibungen der e-Procurement Systeme in einer Process Description Language vorliegen, um die Prozessabläufe von Ein- und Verkäufer aufeinander abgleichen zu können.

Für die Bedeutungs- und die Prozessbeschreibung wurde die semantische Technologie OWL-S genutzt und erweitert, die der Beschreibung von Web Services dient. Für die Validierung einer potenziellen Prozessbeschreibung wurde eine Metaprozessbeschreibung definiert, die die Mindestbedingungen einer openTRANS-Prozessbeschreibung[42] festlegt.

Aufgrund der Analyse der potenziellen Prozessbeschreibungen beider openTRANS-Web Services in OWL-S wird eine komponierte Prozessbeschreibung erstellt. Diese komponierte Prozessbeschreibung wird in eine Process Execution Language übersetzt. Hier wurde BPEL4WS verwendet. Ein BPEL4WS-Dokument wird kompiliert und als Web Service (Flow-Service) auf einer Process Engine (Collaxa BPEL Server) gestartet.

Der Flow-Service dient als Verbindungspunkt der openTRANS-Web Services, der exakt die Anforderungen eines openTRANS-Web Service an sein Gegenüber darstellt. Daher sind keine Anpassungen in den e-Procurement-Systeme der Geschäftspartner nötig, um eine Markttransaktion über den Flow-Service ablaufen zu lassen.

Anmerkungen

Ein Konzept zur Prozessintegration, das alle Eventualitäten des e-Procurement umfasst, hätte den Rahmen dieser Arbeit gesprengt. So mussten einige Einschränkungen getroffen werden, um die Konzentration auf die grundsätzliche Realisation einer automatischen Prozessintegration zu ermöglichen. Für einen realen Einsatz müssten einige Erweiterungen des Lösungskonzepts durchgeführt werden. Dazu zählen:

[42] openTRANS-Prozessbeschreibung: Prozessbeschreibung für den Austausch von openTRANS-Dokumenten über Web Services

- Rechtliche und sicherheitsrelevante Aspekte: Bei der Durchführung von Markttransaktionen ist es wichtig, dass das Versenden von Nachrichten nicht abgestritten werden kann. Dazu wäre eine Signatur versendeter Nachrichten geeignet. Allerdings sollten nicht nur die Geschäftspartner ihre Nachrichten signieren, sondern auch der Flow-Service sollte alle eingehenden Nachrichten signieren und abspeichern, um eine spätere Rekonstruktion des Nachrichtenverlaufs zu erlauben (Audit Trail)
- Terminierung einer Flow-Service-Instanz. Beendet wird die Instanz eines Flow-Service bei Versand der letzten Nachricht im Prozessablauf, oder bei einem Timeout durch Nicht-Eintreffen von Geschäftsdokumenten, deren Eintreffen verpflichtend für einen korrekten Prozessablauf ist. Hier muss eine Termination durch externes Einwirken der Geschäftspartner erlaubt sein. Ein ORDERCHANGE-Dokument kann beispielsweise den Abbruch der Markttransaktion beinhalten. Der Einkäufer sollte daraufhin auch die Instanz des Flow-Service beenden können.
- Teillieferungen. Im Lösungskonzept werden pro Instanz nur eine DISPATCH-NOTIFICATION und eine RECEIPTACKNOLEDGMENT erlaubt. Bei Teillieferungen sind hier mehrere Dokumente möglich. Momentan sieht das Konzept vor, das eine RECEIPTACKNOWLEDGEMENT immer nach einer DISPATCHNOTIFICATION erscheinen muss. Bei Teillieferungen könnte allerdings eine Lieferung L1 vor einer Lieferung L2 abgeschickt werden, jedoch L2 vor L1 beim Einkäufer eintreffen. Da der Prozess im nach dem Lösungskonzept keine Einsicht in die Geschäfts-dokumente hat, kann er die eingehenden Dokumente nicht den einzelnen Lieferungen zuordnen. Wenn dem Flow-Service keine Einsicht in die Dokumente gewährt werden soll, kann dieser nur überprüfen, ob die gleiche Anzahl von RECEIPTACKNOWLEDGEMENT und DISPATCHNOTIFICATION-Dokumenten eingegangen ist.
- Deadlocks. Im Lösungskonzept ist eine Einordnung von INVOICE zwischen DISPATCHNOTIFICATION und RECEIPTACKNOWLEDGEMENT nicht erlaubt. Grundsätzlich müsste dies Möglichkeit den Geschäftspartnern aber offen stehen. Hier ist eine Erweiterung hinsichtlich der Bearbeitung von Deadlocks notwendig.
- Fehlermeldungen. Diese sind zum Beispiel notwendig, wenn eine Nachricht nicht zugestellt werden kann oder wenn Warnungen an die Geschäftspartner wegen Timeouts gesendet werden müssen. Zudem müssen Geschäftspartner informiert werden, wenn sie eine Nachricht zu einem falschen Zeitpunkt des Prozessablaufs an den Flow-Service senden, und unter Umständen mit der Information versorgt werden, was stattdessen von ihnen erwartet wird. Auch sollten die Geschäftspartner über einen Absturz des Flow-Service bzw. seiner Process Engine informiert werden können, und die Geschäftspartner sollten einander über Fehler in ihren Systemen benachrichtigen können.
- Transaktions-Management: Können Nachrichten nicht zugestellt werden, oder sind sie aufgrund von Fehlern versendet worden, muss die Möglichkeit bestehen sie zurückzunehmen, und alle Teilnehmer müssen darüber informiert werden können.

Teilweise sind diese weiteren Anforderungen noch nicht in den genutzten Technologien umsetzbar, bzw. müssten mit komplizierten Eigenlösungen realisiert werden. Der Collaxa BPEL Server bietet zum Beispiel die Möglichkeit mit einem `exec`–Konstrukt Java-Programmcode in BPEL4WS Dokumenten einzufügen, um weitere Funktionalität hinzuzufügen. Gerade bei Sicherheit und Fehlermeldungen sollten jedoch standardisierte Lösungen genutzt werden, um die Interoperabilität der Systeme zu gewährleisten. Hier sind Erweiterungen an den Technologien zur Komposition von Web Services (wie BPEL4WS und deren Engines) notwendig.

Ausblick

Es wurde gezeigt, dass eine dynamische Prozessintegration mit den vorhandenen Technologien und Standards bereits heute möglich ist. Das Lösungskonzept verfolgt einen generischen Ansatz, allerdings sind einige nicht standardisierte Erweiterungen notwendig gewesen, wie der Ausbau der Ontologie für openTRANS-Prozessbeschreibungen.

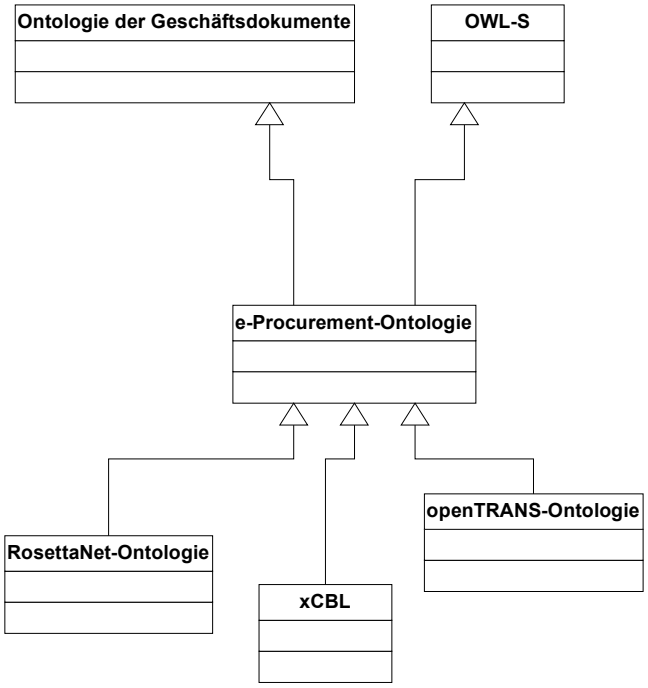

Abbildung 6.1 e-Procurement-Ontologien

Sollte der Ansatz der Semantic Web in Zukunft weiterverfolgt werden, ist zu erwarten, dass bald auch Ontologien für die Beschreibung von Geschäftsprozessen erstellt werden. Darunter könnte sich dann auch eine standardisierte Ontologie zur Beschreibung von e-Procurement-Prozessen finden.

Die erarbeiteten ontologischen Begriffe der openTRANSProcess.owl könnte mit Hilfe ontologischer Properties auf eine solche bezogen werden. Ebenso könnten andere Definitionen von Prozessabläufen sich auf eine solche e-Procurement-Ontologie beziehen (siehe Abbildung 6.1).

So könnte in Zukunft mächtigeren Integrationssystemen, als dem hier erarbeiteten, die Möglichkeit geben werden e-Procurement-Systeme, die verschiedene Spezifikationen nutzen (z.Bsp.: xCBL und openTRANS), dynamisch zu integrieren.

Mit auf Basis semantischer Technologien vergleichbaren Prozessbeschreibung ist es dabei nicht getan. Dazu bedarf es natürlich noch weiteren Komponenten, die zum Beispiel ein openTRANS-Dokument in ein xCBL-Dokument übersetzen. Solche Dienstleistungen könnten von e-PSPs[43] übernommen werden.

Es ist zu erwarten, dass ePSPs bei dynamischen Prozessintegrationen eine große Rolle spielen werden. Sie werden unter anderem die Komponenten der Prozessintegration wie den Process Composer, Process Transformer und den Process Engine bereitstellen und unter Umständen auch Hilfestellung bei der Erstellung der potenziellen Prozessbeschreibungen leisten.

Durch semantische Technologien wird es erstmals möglich die Bedeutung von Komponenten, Web Services und deren Schnittstellen maschinenverständlich zu beschreiben. Semantische Anwendungen können daher den Sinn und die Kompositionsmöglichkeiten dieser Komponenten ohne menschliche Hilfestellung verstehen und miteinander verbinden.

[43] e-Procurement Service Provicer

Abkürzungsverzeichnis

A2Ai	Application to Application Integration
A2C	Adminstration to Customer
B2B	Business to Business
B2Bi	Business to Business -Integration
B2C	Business to Customer
B2A	Business to Administration
BME	Bundesverband Materialwirtschaft, Einkauf und Logistik e. V.
BPEL4WS	Business Process Execution Language for Web Services
BPS	Business Process Specification
CPD	Composed Processdescription
CRM	Customer Relationship Management
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
EAI	Enterprise Application Integration
ebBPSS	ebXML Business Process Specification Schema
e-Business	Electronic Business
e-Commerce	Electronic Commerce
EDI	Electronic Data Interchange
EDIFACT Transport	Electronic Data Interchange for Administration, Commerce and
EPD	Executable Processdescription
e-Procurement	Electronic Procurement
e-PSP	e-Procurement Service Provider
ERP	Enterprise Resource Planning
IKT	Informations- und Kommunikationstechnologien
MPD	Meta Process Description
PC	Process Composer
PE	Process Engine
PIP	Partner Interface Process

PI	Process Integrator
PIS	Process Integration Service
PT	Process Transformer
PPD	Potential Process Description
RPC	Remote Procedure Call
SCM	Supply Chain Management
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
VAN	Value Added Network
WS	Web Service
WSDL	Web Serivce Description Language
WSMF	Web Service Modeling Framework
WWW	World Wide Web
xCBL	XML Common Business Library
XML	eXtensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[BERL1] Dr. Joachim Quantz, Dr. Thorsten Wichmann, "Basisreport Integration mit Web Services", http://www.berlecon.de/output/studien.php#Basisreports, Juli 2003

[Bussler1] Christoph Bussler, "B2B Protocol Standards and their Role in Semantic B2B Integration Engines", http://research.microsoft.com/research/db/debull/A01-mar/issue.htm, 2001

[BPEL4WS1] Microsoft, IBM, u.a., "Specification: Business Process Execution Language for Web Services Version 1.1", http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, Mai 2003

[DAML1] Joint US/EU ad hoc Agent Markup Language Committee "DAML+OIL", http://www.daml.org/2001/03/daml+oil-index.html, März 2001

[DAML2] Roxane Ouellet, Uche Ogbuji, "Introduction to DAML: Part I" http://www.xml.com/pub/a/2002/01/30/daml1.html, 30. Januar 2002

[DC1] Dublin Core "Dublin Core Metadata Element Set, Version 1.1: Reference Description" http://dublincore.org/documents/dces/ 6. Februar 2002

[Dolmetsch1] Ralph Dolmetsch "eProcurement ", Addison-Wesley, 2000

[DynDisc1] Steve Vinoski of Iona, "Web services and dynamic discovery" http://www.webservices.org/index.php/article/articleprint/66/-1/24/ , 03. Dezember 2001

[ebBPSS1] Business Process Project Team, "ebXML Business Process Specification Schema Version 1.01" http://www.ebxml.org/specs/ebBPSS.pdf, 11. Mai 2001

[ebpml1] Jean-Jacques Dubray, "What is a business process anyway?" http://www.ebpml.org/architecture.htm , Ansicht vom 30.Oktober 2003

[ebXML1] ebXML Technical Architecture Project Team, "ebXML Technical Architecture Specification v1.0.4", http://ebxml.org/specs/ebTA.pdf , 16. Februar 2001

[ebXML2] Romin Irani, "An Introduction to ebXML", http://www.webservicesarchitect.com/content/articles/irani02.asp, 11. Juli 2001

[GART1] Michele Cantara, Whit Andrews, Nicole Latimer, "Gartner First Take, FT-19-9047", http://www.gartner.com/resources/114500/114570/114570.pdf, 24. April 2003

[Go4] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns", Addison-Wesley, 1995

[Haertsch1] Patrick Haertsch, "Wettbewerbsstrategien für Electronic Commerce", Dissertation, 2000

[Hentrich1] Johannes Hentrich, "B2B-Katalogmanagement", Galileo Business, 2001"

[Kalakota1] Ravi Kalakota, "e-Business , Roadmap to Success", Addison-Wesley, 1999

[Linthicum1] David S. Linthicum, "B2B Application Integration e-Business-Enable Your Enterprise", 2000

[Merz1] Michael Merz, "Electronic Commerce", dpunkt.verlag, 1999

[MS1] Bill Gates, "Microsofts .NET Today", http://www.microsoft.com/net/downloads/net_today.pdf, Juni 2001

[MS2] Microsoft Corporation, "Microsoft Publishes Key Specification for Integrating Web Services", Pressemitteilung, http://www.microsoft.com/PressPass/press/1999/Nov99/SOAPpr.asp , Microsoft Corporation, Redmond, WA, USA, 1999.

[Möhrstädt1] Detlef G. Möhrstädt, Philipp Bogner, Sascha Paxian "Electronic Procurement planen-einführen-nutzen", Schäffer Poeschel, 2001

[opT1] openTRANS-Expertenkreis, http://www.opentrans.de/, Ansicht vom 25. Februar 2004

[opT2] Oliver Kelkar, Boris Otto, Volker Schmitz, "Spezifikation openTRANS Version 1.0", 7. September 2001

[ORiordan1] David O'Riordan, "Business Process Standards For Web Services", http://www.webservicesarchitect.com/content/articles/oriordan01.asp, 2002

[OWL1] W3C Web-Ontology (WebOnt) Working Group, "Web-Ontology (WebOnt) Working Group Page", http://www.w3.org/2001/sw/WebOnt/ , Ansicht vom 25. Februar 2004

[OWLS1] OWL-S Coalition, "OWL-S: Semantic Markup for Web Services" http://www.daml.org/services/SWWS.pdf, 1.August 2001

[Peltz1] Chris Peltz, Hewlett Packard, Co. " web services orchestration", http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf, Januar 2003

[Riordan1] David O'Riordan, "Business Process Standards for Web Services", http://www.webservicesarchitect.com/content/articles/oriordan01.asp, 10. April 2002

[RDF1] W3C, "Resource Description Framework (RDF)" http://www.w3.org/RDF/, Ansicht vom 25. Februar 2004

[RDF2] W3C Recommendation "Resource Description Framework (RDF) Model and Syntax Specification" http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 22 Februar 1999

[RDF4] W3C Working Draft "RDF/XML Syntax Specification" http://www.w3.org/TR/rdf-syntax-grammar/, 11. August 2003

[RDFS1] W3C Working Draft, "RDF Vocabulary Description Language 1.0: RDF Schema" http://www.w3.org/TR/rdf-schema/ , 5. September 2003

[Scholz1] Scholz, Michael, "Electronic Commerce vs. Electronic Business - Kurze Einführung und Abgrenzung der Begriffe", http://www.competence-site.de/C125693E0068D84A/0/D6EC822CCE5DBBA7C125697B0057F157?Open 2000

[Schubert1] Petra Schubert, Ralf Wölfle, Walter Detling, "Procurement im E-Business" Hanser, 2002

[SemWeb1] Tim Berners-Lee, James Hendler, Ora Lassila, "The Semantic Web", Scientific American, (http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2), Mai 2001

[SemWeb2] Sheila A. McIlraith, David L. Martin, "Bringing Semantics to Web Services", http://www.daml.org/services/pubs/090-093.pdf, Januar /Februar, 2003

[SOAP1] W3C, XML Protocol Working Group, "Simple Object Access Protocol (SOAP)", http://www.w3.org/TR/SOAP/, 8.Mai 2000

[SOAP2] John Ibbotson, „SOAP Version 1.2 Usage Scenarios", http://www.w3.org/TR/2002/WD-xmlp-scenarios-20020626/, W3C Working Group Note, 30 Juli 2003

[SWAD1] SWAD-Europe, "Deliverable 12.1.1: Semantic web applications - analysis and selection" (http://www.w3.org/2001/sw/Europe/reports/chosen_demos_rationale_report/hp-applications-selection.html)

[SWAD2] SWAD-Europe, "Deliverable 12.1.1: Semantic web applications - analysis and selection Appendix B - Application Survey" http://www.w3.org/2001/sw/Europe/reports/chosen_demos_rationale_report/hp-applications-survey.html

[SWIFT1] http://www.swift.com/?item_id=1243 , Ansicht vom 27 Oktober 2003

[SWWS1] http://swws.semanticweb.org/swws , Ansicht vom 03. November 2003

[SWWS2] Juan Miguel Gomez u.a., SWWS – Semantic Web Enabled Web Services, "Draft Version of a Web Services Modeling Framework", http://swws.semanticweb.org/public_doc/D2.1.pdf, 22. April 2003

[TBLee1] Tim Berners-Lee, "Semantic Web - XML2000" http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[TomGr] Tom Gruber, "Toward principles for the design of ontologies used for knowledge sharing" http://ksl-web.stanford.edu/KSL_Abstracts/KSL-93-04.html. August 1993

[Trastour1] David Trastour, Claudio Bartolini, Chris Preist, "Semantic Web Support for the Business-to-Business E-Commerce Lifecycle", 2002

[UDDI] Universal Description, Discovery and Integration http://www.uddi.org

[Williamson1] O.E. Williamson, W.G. Ouchi, "The Markets and Hierarchies and Visible Hand Perspectives. The Markets and Hierarchies Programm of Research: Origins, Implications, Prospects" in:A. Van de Ven, W.F. Joyce (Hrsg.) "Perspectives on Organization Design and Behavior", New York 1981

[WS1] Enrique Castro-Leon of Intel, "A perspective on Web Services" http://www.webservices.org/index.php/article/articleprint/113/-1/24/, 18. Februar 2002

[WS2] Clay Shirky, "Web Services: It's So Crazy, It Just Might Not Work", http://webservices.xml.com/pub/a/ws/2001/10/03/webservices.html, Oktober 2001

[WS3] Hugo Haas W3C, Allen Brown Microsoft (until June 2002), "Web Services Glossary", http://www.w3.org/TR/ws-gloss/#general, W3C Working Draft 8. August 2003

[WS4] "Web Services Addressing", http://www-106.ibm.com/developerworks/library/specification/ws-add/, IBM developerWorks, Ansicht vom 3. Mai 2004

[WSDL1] W3C, Web Services Description Working Group, "Web Services Description Language (WSDL) Version 2.0 Part1", http://www.w3.org/TR/wsdl20/, 10. November 2003

[WSCI] W3C, Web Services Choreography Working Group "Web Service Choreography Interface (WSCI) 1.0" http://www.w3.org/TR/wsci/, 8. August 2002

[WSFL1] Prof. Dr. Frank Leymann und IBM, "Web Services Flow Language 1.0", http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, Mai 2001

[WSMF1] Dieter Fensel und Chris Bussler, "The Web Service Modeling Framework WSMF" http://informatik.uibk.ac.at/users/c70385/ftp/paper/wsmf.pdf , Januar 2002

[x12.org1] http://www.x12.org/ Ansicht vom am 27.Oktober.2003

[xCBL1] http://www.xcbl.org/xcbl40/documentation/structureRef/, Ansicht vom 12. Februar 2004

[XLANG1] Satish Thatte und Microsoft, " XLANG Web Services for Business Process" Design,"http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001

[XML1] W3C, http://www.w3.org/XML/, Ansicht vom 12 Februar 2004,

[XMLS1] W3C Recommendation, "XML Schema Part 2: Datatypes"http://www.w3.org/TR/xmlschema-2/#string , 02 Mai 2001

[Yendluri1] Prasad Yendluri, Principal Architect, webMethods, Inc, " Web Services Choreography ", http://www.webservices.org/index.php/article/view/1178/, 23. September 2003

ANHANG

ANHANG A: openTRANSProcess.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:owl=          "http://www.w3.org/2002/07/owl#"
  xmlns:rdfs=         "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf=          "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd=        "http://www.w3.org/2001/XMLSchema#"
  xmlns:service=      "http://www.daml.org/services/owl-s/1.0/Service.owl#"
  xmlns:process=      "http://www.daml.org/services/owl-s/1.0/Process.owl#"
  xmlns:profile=      "http://www.daml.org/services/owl-s/1.0/Profile.owl#"
  xmlns:grounding=    "http://www.daml.org/services/owl-s/1.0/Grounding.owl#"
  xmlns:opTprocess=   "http://processgen.openTRANS.fhg.de#"
  xml:base=           "http://processgen.openTRANS.fhg.de"
>

  <owl:Ontology rdf:about="">

    <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl"/>
  </owl:Ontology>

<owl:Class rdf:ID="openTRANSCompositeProcess">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl#CompositeProcess"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="portTypeBuyer">
  <rdfs:domain rdf:resource="#openTRANSCompositeProcess"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="portTypeSeller">
  <rdfs:domain rdf:resource="#openTRANSCompositeProcess"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="replacableBy">
  <rdfs:domain rdf:resource="#MUSTopenTRANSAP"/>
  <rdfs:range rdf:resource="#MUSTopenTRANSAP"/>
</owl:ObjectProperty>


<owl:Class rdf:about="openTRANSCompositeProcess">
  <rdfs:comment>
      Restricting the cardinality of portTypes  to one
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction owl:cardinality="1">
        <owl:onProperty rdf:resource="#portTypeSeller"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction owl:cardinality="1">
        <owl:onProperty rdf:resource="#portTypeBuyer"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:ID="openTRANSAP">
  <rdfs:comment>
    The superclass for all openTRANS Atomic Processes, like setRFQ and
setQUOTATION
```

```
    </rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl#AtomicProcess"/>
</owl:Class>

<!--
Cardinality explains, how often a AP should or could be called
-->
<owl:ObjectProperty rdf:ID="cardinalityIs">
  <rdfs:domain rdf:resource="#openTRANSAP"/>
  <rdfs:range rdf:resource="#Cardinality"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="timeoutCompensation">
  <rdfs:domain rdf:resource="#openTRANSAP"/>
  <rdfs:range rdf:resource="#Timeout_Compensation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="timeoutCompensation">
  <rdfs:domain rdf:resource="#openTRANSAP"/>
  <rdfs:range rdf:resource="#Cardinality_Compensation"/>
</owl:ObjectProperty>


<owl:Class rdf:about="openTRANSAP">
  <rdfs:comment>
      Restricting the cardinality of  cardinalityIs to one
    </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction owl:cardinality="1">
        <owl:onProperty rdf:resource="#cardinalityIs"/>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:ID="Cardinality">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#COULD_Cardinality"/>
      <owl:Class rdf:about="#WILL_Cardinality"/>
    </owl:oneOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="minCardinality">
  <rdfs:domain rdf:resource="#Cardinality"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="maxCardinality">
  <rdfs:domain rdf:resource="#Cardinality"/>
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:ObjectProperty>



<owl:Class rdf:ID="NO_Cardinality">
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_0"/>
    </owl:oneOf>
</owl:Class>
<owl:Class rdf:ID="COULD_Cardinality">
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
```

```
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_01"/>
      <owl:Class rdf:about="#Cardinality_0m"/>
    </owl:oneOf>
</owl:Class>
<owl:Class rdf:ID="WILL_Cardinality">
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_1"/>
      <owl:Class rdf:about="#Cardinality_1m"/>
      <owl:Class rdf:about="#Cardinality_nm"/>
      <owl:Class rdf:about="#Cardinality_m"/>
    </owl:oneOf>
</owl:Class>

<!--
possible Cardinalities (where m also be inifinite:
0   -> this Atomic Process can be used 0 times
01  -> this Atomic Process can be used 0 or 1 times
0m  -> this Atomic Process can be used 0 or up to m times
1   -> this Atomic Process has to be used exactly 1 time
1m  -> this Atomic Process has to be used 1 up to m times
nm  -> this Atomic Process has to be used n up to m times
m   -> this Atomic Process has to be used exactly m times


-->
<owl:Class rdf:ID="Cardinality_0">
  <rdfs:subClassOf rdf:resource="#NO_Cardinality"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#minCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#maxCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Cardinality_01">
  <rdfs:subClassOf rdf:resource="#COULD_Cardinality"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#minCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#maxCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Cardinality_0m">
  <rdfs:subClassOf rdf:resource="#COULD_Cardinality"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#minCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">0</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Cardinality_1">
  <rdfs:subClassOf rdf:resource="#WILL_Cardinality"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#minCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#maxCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<!-- m should be greater than 1 -->
<owl:Class rdf:ID="Cardinality_1m">
  <rdfs:subClassOf rdf:resource="#WILL_Cardinality"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#minCardinality"/>
        <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:hasVal
ue>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<!--
At the moment Cardinality_nm and Cardinality_m look equal.
Cardinality_m should be constraint with n = m
Cardinality_nm should be constraint with m>n>0
-->
<owl:Class rdf:ID="Cardinality_nm">
  <rdfs:subClassOf rdf:resource="#WILL_Cardinality"/>
</owl:Class>
<owl:Class rdf:ID="Cardinality_m">
  <rdfs:subClassOf rdf:resource="#WILL_Cardinality"/>
</owl:Class>


<!--
  SUPERCLASSES of openTRANSAPs
-->
<owl:Class rdf:ID="MUSTopenTRANSAP">
  <rdfs:subClassOf rdf:resource="#openTRANSAP"/>
</owl:Class>

<owl:Class rdf:ID="MAYopenTRANSAP">
  <rdfs:subClassOf rdf:resource="#openTRANSAP"/>
</owl:Class>
```

```
<owl:Class rdf:ID="openTRANSBuyerAP">
  <rdfs:subClassOf rdf:resource="#openTRANSAP"/>
</owl:Class>

<owl:Class rdf:ID="openTRANSSellerAP">
  <rdfs:subClassOf rdf:resource="#openTRANSAP"/>
</owl:Class>

<!--
the atomic processes (operations) a SELLER can/has to implement
-->
<owl:Class rdf:ID="setRFQ">
  <rdfs:subClassOf rdf:resource="#openTRANSSellerAP"/>
</owl:Class>
<owl:Class rdf:ID="setORDER">
  <rdfs:subClassOf rdf:resource="#openTRANSSellerAP"/>
</owl:Class>
<owl:Class rdf:ID="setORDERCHANGE">
  <rdfs:subClassOf rdf:resource="#openTRANSSellerAP"/>
</owl:Class>
<owl:Class rdf:ID="setRECEIPTACKNOWLEDGMENT">
  <rdfs:subClassOf rdf:resource="#openTRANSSellerAP"/>
</owl:Class>
<!--
the atomic processes (operations) a BUYER can/has to implement
-->
<owl:Class rdf:ID="setQUOTATION">
  <rdfs:subClassOf rdf:resource="#openTRANSBuyerAP"/>
</owl:Class>
<owl:Class rdf:ID="setORDERRESPONSE">
  <rdfs:subClassOf rdf:resource="#openTRANSBuyerAP"/>
</owl:Class>
<owl:Class rdf:ID="setDISPATCHNOTIFICATION">
  <rdfs:subClassOf rdf:resource="#openTRANSBuyerAP"/>
</owl:Class>
<owl:Class rdf:ID="setINVOICE">
  <rdfs:subClassOf rdf:resource="#openTRANSBuyerAP"/>
</owl:Class>


<!--
  the MUST subtypes of openTRANSAPs
-->
<owl:Class rdf:ID="MUSTsetRFQ">
  <rdfs:subClassOf rdf:resource="#setRFQ"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetORDER">
  <rdfs:subClassOf rdf:resource="#setORDER"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetORDERCHANGE">
  <rdfs:subClassOf rdf:resource="#setORDERCHANGE"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetRECEIPTACKNOWLEDGMENT">
  <rdfs:subClassOf rdf:resource="#setRECEIPTACKNOWLEDGMENT"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetQUOTATION">
  <rdfs:subClassOf rdf:resource="#setQUOTATION"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetORDERRESPONSE">
  <rdfs:subClassOf rdf:resource="#setORDERRESPONSE"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
```

```
</owl:Class>
<owl:Class rdf:ID="MUSTsetDISPATCHNOTIFICATION">
  <rdfs:subClassOf rdf:resource="#setDISPATCHNOTIFICATION"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MUSTsetINVOICE">
  <rdfs:subClassOf rdf:resource="#setINVOICE"/>
  <rdfs:subClassOf rdf:resource="#MUSTopenTRANSAP"/>
</owl:Class>

<!--
  the MAY subtypes of openTRANSAPs
-->
<owl:Class rdf:ID="MAYsetRFQ">
  <rdfs:subClassOf rdf:resource="#setRFQ"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<!-- NOT ALLOWED:
<owl:Class rdf:ID="MAYsetORDER">
  <rdfs:subClassOf rdf:resource="#setORDER"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
-->
<owl:Class rdf:ID="MAYsetORDERCHANGE">
  <rdfs:subClassOf rdf:resource="#setORDERCHANGE"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MAYsetRECEIPTACKNOWLEDGMENT">
  <rdfs:subClassOf rdf:resource="#setRECEIPTACKNOWLEDGMENT"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MAYsetQUOTATION">
  <rdfs:subClassOf rdf:resource="#setQUOTATION"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MAYsetORDERRESPONSE">
  <rdfs:subClassOf rdf:resource="#setORDERRESPONSE"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MAYsetDISPATCHNOTIFICATION">
  <rdfs:subClassOf rdf:resource="#setDISPATCHNOTIFICATION"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>
<owl:Class rdf:ID="MAYsetINVOICE">
  <rdfs:subClassOf rdf:resource="#setINVOICE"/>
  <rdfs:subClassOf rdf:resource="#MAYopenTRANSAP"/>
</owl:Class>^

<!--
Compensations
-->
<owl:Class rdf:ID="Compensation">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl#AtomicProcess"/>
  <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_Compensation"/>
      <owl:Class rdf:about="#Timeout_Compensation"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="Cardinality_Compensation">
  <rdfs:subClassOf rdf:resource="#Compensation"/>
</owl:Class>
<owl:Class rdf:ID="Timeout_Compensation">
  <rdfs:subClassOf rdf:resource="#Compensation"/>
</owl:Class>
```

```
<!--
Conditions for If-Then-Else
-->

<owl:Class rdf:ID="RFQoccuredCondition">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl#Condition"/>
</owl:Class>
<owl:Class rdf:ID="DISPATCHorRECEIPTACKoccuredCondition">
  <rdfs:subClassOf rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl#Condition"/>
</owl:Class>

</rdf:RDF>
```

ANHANG B: Metaprozessbeschreibung

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:owl=          "http://www.w3.org/2002/07/owl#"
  xmlns:rdfs=         "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf=          "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:service=      "http://www.daml.org/services/owl-s/1.0/Service.owl#"
  xmlns:process=      "http://www.daml.org/services/owl-s/1.0/Process.owl#"
  xmlns:profile=      "http://www.daml.org/services/owl-s/1.0/Profile.owl#"
  xmlns:grounding=    "http://www.daml.org/services/owl-s/1.0/Grounding.owl#"
  xmlns:opTprocess=   "http://processgen.openTRANS.fhg.de#"
  xml:base=           "http://metaprocess.openTRANS.fhg.de"
>

  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.0/Process.owl"/>
    <owl:imports
rdf:resource="file://localhost/C:/eclipse/eclipse2.1.1/workspace/openTRANS/temp/
openTRANS_owl"/>
  </owl:Ontology>

<!--
The META PROCESS DESCRIPTION
-->
<process:CompositeProcess rdf:ID="openTRANSMETAProcess">

  <process:composedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">

        <opTprocess:METAsetRFQ rdf:ID="setRFQ"/>

        <process:If-Then-Else>
          <process:ifCondition>
            <opTprocess:RFQoccuredCondition/>
          </process:ifCondition>
          <process:then>
            <opTprocess:METAsetQUOTATION rdf:ID="setQUOTATION"/>
          </process:then>
        </process:If-Then-Else>

        <opTprocess:METAsetORDER rdf:ID="setORDER"/>
        <opTprocess:METAsetORDERRESPONSE rdf:ID="setORDERRESPONSE"/>

        <opTprocess:METAsetORDERCHANGE rdf:ID="setORDERCHANGE"/>
        <opTprocess:METAsetORDERRESPONSE rdf:ID="setORDERRESPONSE"/>

        <process:Unordered>

          <process:Sequence>
            <opTprocess:METAsetDISPATCHNOTIFICATION
rdf:ID="setDISPATCHNOTIFICATION"/>
            <opTprocess:METAsetRECEIPTACKNOWLEDGEMENT
rdf:ID="setRECEIPTACKNOWLEDGEMENT"/>
          </process:Sequence>

          <process:Sequence>
            <opTprocess:METAsetINVOICE rdf:ID="setINVOICE"/>

            <process:If-Then-Else>
              <process:ifCondition>
                <opTprocess:DISPATCHorRECEIPTACKoccuredCondition/>
              </process:ifCondition>
              <process:else>
```

```
                <opTprocess:METAsetORDERCHANGE rdf:ID="setORDERCHANGE"/>
                <opTprocess:METAsetORDERRESPONSE rdf:ID="setORDERRESPONSE"/>
              </process:else>
            </process:If-Then-Else>
          </process:Sequence>

        </process:Unordered>

      </process:components>
    </process:Sequence>
  </process:composedOf>

</process:CompositeProcess>

<!--
  The META openTRANSAPs are subclasses of a specific openTRANSAP
  ---------------------
  Restrictions on the cardinalityIs Property restrict, which
  Cardinalities can be used by the openTRANSAP
-->
<owl:Class rdf:ID="METAsetRFQ">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setRFQ"/></owl:Class>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONEn_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetQUOTATION">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setQUOTATION"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONE_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetORDER">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#MUSTsetORDER"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm
rdf:resource="http://processgen.openTRANS.fhg.de#Cardinality_1"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetORDERRESPONSE">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setORDERRESPONSE"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONE_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetORDERCHANGE">
```

```
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setORDERCHANGE"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROorZEREOn_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetRECEIPTACKNOWLEDGMENT">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setRECEIPTACKNOWLEDGMENT"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONE_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetDISPATCHNOTIFICATION">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setDISPATCHNOTIFICATION"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONE_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="METAsetINVOICE">
  <rdfs:subClassOf
rdf:resource="http://processgen.openTRANS.fhg.de#setINVOICE"/>
  <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty
rdf:resource="http://processgen.openTRANS.fhg.de#cardinalityIs"/>
        <owl:allValuesForm rdf:resource="#ZEROuptoONE_Cardinality"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<!--
  The following Cardinality subclasses

  are only used to specify a group of Cardinality,
  which can be used by the META openTRANSAPs
-->
<!--
Cardinality 0, 0-1, 1
-->
<owl:Class rdf:ID="ZEROuptoONE_Cardinality">
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_0"/>
      <owl:Class rdf:about="#Cardinality_01"/>
      <owl:Class rdf:about="#Cardinality_1"/>
    </owl:oneOf>
</owl:Class>

<!--
Cardinality 0, 0-n
-->
<owl:Class rdf:ID="ZEROorZEROn_Cardinality">
```

```
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_0"/>
      <owl:Class rdf:about="#Cardinality_0n"/>
    </owl:oneOf>
</owl:Class>

<!--
Cardinality 0, 0-n, 1, 1-n
-->
<owl:Class rdf:ID="ZEROuptoONEn_Cardinality">
  <rdfs:subClassOf rdf:resource="#Cardinality"/>
  <owl:oneOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Cardinality_0"/>
      <owl:Class rdf:about="#Cardinality_0n"/>
      <owl:Class rdf:about="#Cardinality_1"/>
      <owl:Class rdf:about="#Cardinality_1n"/>
    </owl:oneOf>
</owl:Class>


</rdf:RDF>
```

ANHANG C: CD-Inhalt

Die beigefügt CD enthält

- den Quelltext des Prototyps
  (Ordner: openTRANSintegrationssystem/openTRANS_ProcessIntegrator/src)
- die genutzten und erstellen Ontologien
  (Ordner: openTRANSintegrationssystem/openTRANS_ProcessIntegrator/src/owl)

sowie alle Dateien, die zu einer Testinstallation des Prototyps benötigt werden.

Dazu gehören die Java-APIs (Ordner: java-APIs) und alle Installations-Dateien zum Aufbau der Systemumgebung (Ordner: systemumgebung)

Die Dateien, die zum Deployment der Geschäftspartner openTRANS-Web Services benötigt werden sind in den Ordnern openTRANSintegrationssystem/openTRANSWebService_Collaxa und openTRANSintegrationssystem/openTRANSWebService_Axis zu finden.

Eine kurze Anleitung zur Installation wird in der Datei installationUndDemo.txt gegeben.