Technische Universität Kaiserslautern

Fachbereich Informatik
AG Datenbanken und Informationssysteme
und
AG Heterogene Informationssysteme

Komposition und Choreographie von Diensten

Sprachen und Frameworks zur Spezifikation von Geschäfts- und Koordinationsprotokollen für e-Business

Seminararbeit
21.07.2004

Benjamin Horak

Betreuer
Prof. Dr. Stefan Deßloch

Gliederung

Einführung

Betrachtet man unter dem Aspekt der Dienstorientierung moderne Unternehmen, so lassen sich folgende Tendenzen erkennen:
Betriebliche Leistungen werden zunehmend zu autonomen Diensten mit definierten Schnittstellen modularisiert. Betriebliche Abläufe werden nun so definiert, dass sie in ihrer Abarbeitung solche Dienste aufrufen.
Durch diese Methodik können selbst sehr komplexe wirtschaftliche Abläufe stark abstrahiert, kontrolliert und letztendlich optimiert werden.
In Folge dieser Tendenzen entstand der Wunsch, gemäß dem Plug & Play Prinzip, vorhandene wirtschaftliche Aktivitätsbausteine (Dienste) zu nahtlosen Prozessketten zu kombinieren. Um diese Wünsche zu erfüllen, entstanden Konzepte und Systeme wie Enterprise Ressource Planning (ERP) für den innerbetrieblichen Bereich und Electronic Data Interchange (EDI) für den zwischenbetrieblichen Austausch.[1] Allerdings zeigte sich im Praxiseinsatz, dass diese Konzepte weiterhin eine bestehende Inflexibilität in der Kommunikation mehrerer heterogener Aktivitätsbausteine besitzen. Es folgte ein weiterentwickelter Absatz unter dem Namen Enterprise Application Integration (EAI), welcher die Flexibilität in heterogenen Umgebungen erhöhen sollte. Leider war der Praxiseinsatz mit diesen Systemen sehr komplex und beschränkte somit die interne Modellierung von Abläufen.

In der folgenden Ausarbeitung werden Möglichkeiten beschrieben, standortunabhängige Dienste in verteilten Prozessen flexibel zu komponieren.

(1) Im 1.Abschnitt veranschaulicht ein Fallbeispiel die Thematik einer Service-Komposition. Im Verlauf der Ausarbeitung wird öfter darauf verwiesen, um abstrakte Beschreibungen zu konkretisieren.
(2) Der 2.Abschnitt definiert benötigte Begriffe und klärt die Vorteile verteilter Geschäftsprozesse. Es werden die Vorteile von Web Services in Bezug auf die Positionierung einer möglichen Choreographie in schon bestehende Technologien geklärt.
(3) Die daraus resultierenden Anforderungen an eine Choreographie von verschiedenen Web Services gliedert der 3.Abschnitt. Hier wird öfter in Auszügen auf das Fallbeispiel im 1.Abschnitt zurückgegriffen, um komplexe Anforderungen und Standards zu erläutern.
(4) Es wurden in vergangener Zeit viele Spezifikationen entworfen, um die Anforderungen einer konkreten Beschreibung von Choreographien umzusetzen. Eine Auswahl bestehend aus BPEL und YAWL, sowie der alternative Ansatz von RosettaNet werden im 4.Abschnitt beschrieben.

Um komplexe Definitionen und Abläufe lesbarer darzustellen, wurden falls möglich und sinnvoll deutsche Synonyme für Fachbegriffe verwendet. Da jedoch bestehende Dokumentationen über diese Thematik vorwiegend in englischer Sprache geschrieben werden, folgt jedem Synonym der englische Originalbegriff in runden Klammern.
Codebeispiele unterscheiden sich von Erläuterungen, indem sie in COURIER Schriftart gestaltet sind und sind im Wesentlichen im Anhang positioniert.
Um den Lesefluss nicht zu unterbrechen werden Zitate, Codebeispiele und Quellenverweise im folgenden lediglich mit einer Nummer oder einem Buchstaben in eckigen Klammern gekennzeichnet und am Ende der Ausarbeitung im Anhang näher erläutert

1 Fallbeispiel einer Choreographie

Das folgende erweiterte Szenario aus [2] beschreibt eine skizzierte Choreographie und wird im Verlauf dieser Ausarbeitung mehrmals wieder referenziert werden, um verschiedene Konzepte zu veranschaulichen:

Herr Mayer möchte eine günstige Digitalkamera kaufen. Dazu besucht er das Onlineportal www.cheapdigicams.com, da hier versprochen wird, durch eine einfache Suchanfrage über viele Onlineshops das günstigste Angebot zu finden, ohne direkt mit den jeweiligen Onlineshops in Kontakt treten zu müssen. Sogar der Kaufauftrag kann von Herrn Mayer direkt über das Portal www.cheapdigicams.com abgeschlossen werden.
Im Folgenden wird der entstehende Kaufprozess **Buy_Cheap_Digi_Cam** genannt. Der Ablauf von **Buy_Cheap_Digi_Cam** definiert sich wie folgt:

(1) Herr Mayer sucht sich ein Kameramodell seiner Wahl aus
(2) www.cheapdigicams.com schickt jeweilige Anfragen zu registrierten Onlineshops.
(3) Jeder Onlineshop verarbeitet die Anfragen und antwortet mit Informationen über Preis und bereitgestellte Kaufkonditionen oder der Nachricht, dass das gesuchte Modell nicht im Sortiment vorhanden ist.
(4) www.cheapdigicams.com bereitet die Informationen auf und liefert Herr Mayer das günstigste Modell.
(5) Falls Herr Mayer das Modell kauft, wird seine Kreditkarte überprüft.
(6) www.cheapdigicams.com schickt einen Kaufauftrag zu dem gewählten Onlineshop.
(7) Dieser Onlineshop verarbeitet den Kaufauftrag mit seinem eigenen, internen Kaufprozess namens **Buy_Digi_Cam**:
    a) Der interne Finanzabrechnungsdienst wird über die Summe des Kaufauftrags benachrichtigt.
    b) Der Lagerbestandsdienst wird über die gekaufte Menge informiert. Falls der Bestand die Marke M unterschreitet wird bei dem Hersteller nachbestellt.
    c) Das Produkt wird zu Herr Mayer verschickt.
    d) Der interne Finanzabrechnungsdienst regelt die Überweisung von Herrn Mayers Konto und zahlt die Rechnung beim Onlineshop.

Im Wesentlichen enthält dieses Szenario die folgenden Merkmale:
Es besteht ein koordinierter Prozessablauf, in dem es möglich ist, weitere Unterprozesse auszuführen;
Herr Mayer erhält keinerlei zusätzlichen Informationen darüber wie sein Kaufauftrag intern verarbeitet wird;
www.cheapdigicams.com besitzt eine bestimmte Anzahl bereits registrierter Onlineshops;
Es besteht ein geschäftliches Vertrauensverhältnis zwischen www.cheapdigicams.com in der Rolle des Käufers und den registrierten Onlineshops in den jeweiligen Verkäufer - Rollen.

2 Virtuelle Unternehmungen

Das Fallbeispiel beschreibt einen vertikalen Verbund zwischen verschiedenen Anbietern von Produkten. Charakterisierend für dieses spezielle gemeinschaftliche Auftreten verschiedener Organisationen (die man hier durchaus als eine Sammlung bereitgestellter Dienste abstrahiert betrachten kann) ist die Kapselung der gesamten Choreographie zu einer so genannten „virtuellen Unternehmung“.[3]
Dem Benutzer wird in einer virtuellen Unternehmung die interne Komplexität der oft heterogenen Abläufe und Datentransfers erspart. Er benutzt diesen gekapselten Dienst als eine Art „Blackbox“ mit definierter Ein/Ausgabe – Schnittstelle und betrachtet innerhalb seiner Perspektive www.cheapdigicams.com als eigenständige Unternehmung.
Es ist anzunehmen, dass www.cheapdigicams.com eine sehr dynamische Registrierung von Onlineshops aufweist, oder dass die Produktpalette von Digitalkameras bei guten Absatzzahlen auf weitere Elektronikartikel erweitert wird. In diesem weitergehenden Denkschritt stellt sich heraus, dass virtuelle Unternehmungen eine enorme Flexibilität verlangen, da sich Strukturen und Inhalte schnell ändern können.

2.1 Geschäftsprozesse

Die folgende Definition der Workflow Management Coalition (WfMC) von 1998 verdeutlicht den Prozessbegriff im Kontext einer Geschäftsumgebung[4]:

Geschäftsprozess – Eine Menge einer oder mehrerer verbundener Prozeduren oder Aktivitäten, welche im Normalfall im Kontext organisatorischer Strukturen mittels funktionaler Rollen und Relationen in ihrem Zusammenwirken eine konkrete Zielsetzung realisieren.

Hieraus folgt die erweiternde Definition des verteilten Geschäftsprozesses[5]:

Verteilter Geschäftsprozess – Ein Verteilter Geschäftsprozess besteht aus lokalen Geschäftsprozessen, welche räumlich und/oder organisatorisch getrennt ablaufen und lediglich über Kommunikationskanäle miteinander verbunden sind.

Im Folgenden wird der Begriff **Choreographie** stellvertretend für Komposition verwendet und leitet sich aus der Modellierung eines verteilten Geschäftsprozesses ab. Eine **Orchestrierung** beschreibt den Kontrollfluss einer Choreographie. Hinter dem sehr abstrakten Begriff **Dienst** können sich nun sämtliche aufrufbaren und erreichbaren Geschäftsprozesse verbergen. Wird nach einem Aufruf eines bestimmten Dienstes durch den entstehenden Prozess kein weiterer Dienst aufgerufen, so nennt man ihn **Basisdienst** (engl.: basic service). Andernfalls betrachtet man ihn als **zusammengesetzten Dienst**. (engl.: composite service)[6].
Bei Choreographien, in denen verschiedene Prozesse von unterschiedlichen Organisationen verknüpft sind, geben die jeweiligen Organisationen die interne Realisierung ihrer eigenen Geschäftsprozesse nur ungern bekannt. Es folgt eine Unterscheidung von **privaten bzw. internen Prozessen**, in denen die internen Aktivitäten durchgeführt werden und **öffentlichen Prozessen,** welche durch definierte Prozessschnittstellen die Möglichkeit besitzen, private Prozesse aufzurufen. Da laut dieser Definition die **öffentlichen Prozesse** lediglich den Nachrichtenaustausch zwischen mehreren internen Prozessen kontrollieren, spricht man hier auch von **abstrakten Prozessen** oder Geschäftsprotokollen.

2.2 Definierte Schnittstellen

Durch das breite Spektrum an Möglichkeiten, Dienste zu generieren oder zu kapseln folgt die sicherlich schwierigste Forderung, die man sich in einem solch heterogenen Feld vorstellen kann:

„Die Forderung nach einheitlichen und einfachen Schnittstellen!“

Eine Schnittstelle zur Kommunikation kann durchaus als eine gemeinsame Sprachbasis[7] betrachtet werden. Die Tatsache, dass Übersetzungen schwerfällig, fehlerträchtig und verlustbehaftet sind liegt klar auf der Hand, jedoch erscheint es bis heute sehr schwierig, unter den facettenreichen Anforderungen, Implementierungen und Firmenpolitiken, mit herkömmlichen EAI Systemen einen gemeinsamen Standard zu finden.
Verlangt wird eine Schnittstelle, die ähnlich wie SQL in der Datenbankwelt, den Zugang zu den verschiedensten Systemen ermöglicht.

A Kapselung durch Web Services

Die Web Service Technologie bietet der Internet-Gemeinschaft nun die Möglichkeit, diese große Forderung zu erfüllen. Web Services eignen sich für nahezu alle Anwendungsgebiete moderner Informationstechnik: von der Produktionssteuerung und –überwachung bis hin zum E-Commerce. Durch ihren modularen Aufbau mit austauschbaren Schichten können sich Web Services zu einem durchgängig akzeptierten Konzept entwickeln und damit heterogene Strukturen hinter einem homogenen Komponentenkonzept verbergen – sogenannte **virtuelle Komponenten**.[8]
Das aus der Datenbankwelt bereits bekannte und bewährte Konzept der logischen Datenunabhängigkeit kann, transferiert auf Web Services, nun als sogenannte logische Dienstunabhängigkeit von den Realisierungen der Dienste gewährleistet werden. Ein Web Service kann überall positioniert sein: Auf einem fernen Kontinent und nur durch das Internet erreichbar, in einer anderen Abteilung im Intranet oder lokal als Thread im eigenen System. Web Services integrieren sich problemlos, unterstützen und fördern sogar das dienstorientierte Denken, können mit unterschiedlichen Frameworks und Protokollen angesprochen[9] und in den verschiedensten Programmiersprachen implementiert werden.

B Choreographie als Programmierung im Großen

Web Services sind implementierungsunabhängig aufrufbar. Das Zusammenspiel der drei Basiskomponenten: SOAP zur Nachrichtenübertragung; WSDL zur Beschreibung der Schnittstellen und UDDI als Verzeichnisdienst zum Auffinden von Web Services, wird bei der Choreographie als Web Service Technology Stack zugrunde gelegt.

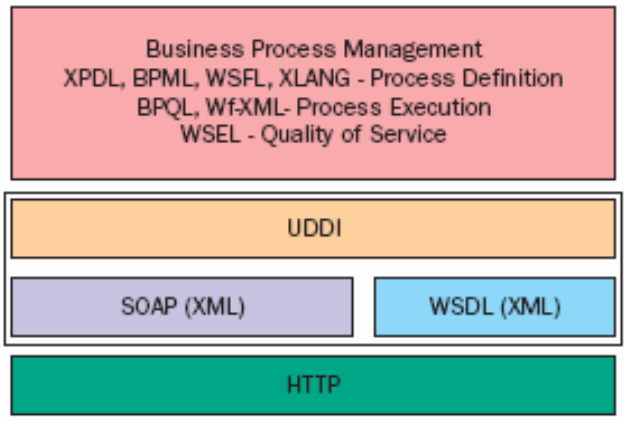


Abbildung 1 - Web Service Technologie Stack

Eine Choreographie definiert einen verteilten Prozess, in dem verschiedene bereits implementierte Basis- oder komponierte Dienste in einem Kontrollfluss aufgerufen werden. Aus dieser Perspektive, die man durchaus als „Programmieren im Großen“ betiteln kann, ergeben sich für Choreographiebeschreibungssprachen (Abb. 1. zeigt

mehrerer Beispiele) Forderungen nach Funktionalität, die in fortgeschrittenen Programmiersprachen mittlerweile standardmäßig bereitgestellt wird.

3 Charakteristika formaler Choreographiebeschreibungssprachen

Die bisherigen Web Service Standards (SOAP, WSDL und UDDI) sind bewusst schlank und einfach definiert worden. Folgende fehlende Eigenschaften sind dem Web Service Technology Stack demnach noch hinzuzufügen, um eine Beschreibung einer Choreographie zu ermöglichen:

- In SOAP bzw. WSDL ist es nicht möglich in einem Web Service einen Zustand zu beschreiben, in dem der momentane Kontext definiert wird.
- SOAP ermöglicht zwar eine standardisierte Fehlergenerierung (zum Teil über HTTP), allerdings lässt sich die Fehlerbehandlung nicht definieren.
- Das Transaktionsparadigma sollte im Bereich der verteilten Geschäftsprozesse untertützt werden, um einen zuverlässigen Kontrollfluss der Prozesse im globalen System sicherzustellen.

Es gilt weiterhin zu Untersuchen, inwiefern sich die Menge aller formaler Beschreibungssprachen für Choreographien klassifizieren lässt, um eine Einordnung verschiedener Sprachkonzepte mit einigen wenigen Charakteristika zu gewährleisten. Zum Teil entstammen diese Charakteristika der Weiterentwicklung der bisherigen Web Service Standards, jedoch reichen die oben erwähnten Mängel nicht aus, um eine umfassende Sprachmächtigkeit einer konkreten Choreographiebeschreibungssprache zu definieren. Die folgenden sechs Charakteristika sollen den Umfang von Choreographien mittels grundlegender Charakteristika beschreiben und die verschiedenen Implementierungsmöglichkeiten zur Problemlösung oben genannter Mängel und weiteren auftretenden Aspekten klassifizieren.

3.1 Komponenten

(engl.: Component Model)

In einer Choreographie müssen gewisse Annahmen über die Art und Weise der Aufrufe der verschiedenen Dienste getroffen werden. Eine restriktive aber durchaus sinnvolle Annahme wäre beispielsweise, lediglich solche Dienste zu verknüpfen, die mit WSDL spezifiziert sind. Auf das bereits erwähnte Beispiel **Buy_Cheap_Digi_Cam** bezogen könnten so im Normalfall alle Bestelldienste von Digitalkameraherstellern benutzt werden, die in UDDI Verzeichnissen eingetragen wurden, da hier WSDL fast ausschließlich verwendet wird. Je restriktiver das Komponentenmodell festgelegt wurde, desto geringer gestaltet sich die Komplexität/Heterogenität, allerdings auch die Kompatibilität zu den zur Verfügung stehenden Diensten der Choreographie.[10]

3.2 Orchestrierung

(engl.:Orchestration Model)

In einer Choreographie wird die Abfolge der Aufrufe verschiedener Dienste durch den Kontrollfluss geregelt. Wie in üblichen Programmiersprachen wird auch hier der Kontrollfluss durch eine Reihe von Anweisungen und Schleifen festgelegt:

Beispiele:

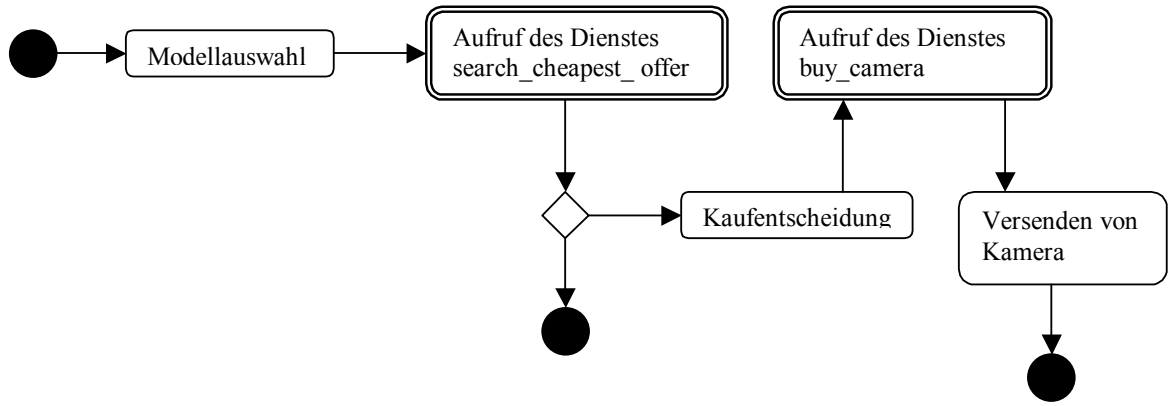
```
IF(BEDINGUNG) THEN ... ELSE;
WHILE(BEDINGUNG);
…
```

Diese interne Logik, im Folgenden **Orchestrierung** genannt, wurde in verschiedenen Modellen zum Ausdruck gebracht, von denen diese Ausarbeitung eine Auswahl am Beispiel der bereits skizzierten Orchestrierung von **Buy_Cheap_Digi_Cam** kurz darstellt.

A UML Activity Diagrams

UML-Activity-Diagramme sind in Orchestrierungen von Web Services weit verbreitet und erfreuten sich auch bei der Modellierung von Workflows großer Beliebtheit. In dieser Definition des Kontrollflusses werden die verschiedenen Aufrufe von Diensten von Prozessbeginn bis Prozessende nacheinander modelliert. Eine Aktivität könnte in diesem Modell den Aufruf eines Dienstes beinhalten. Der Vorteil dieser Beschreibungsmethode liegt in der Einfachheit der Modellierung und der bestehenden großen Werkzeugunterstützung.

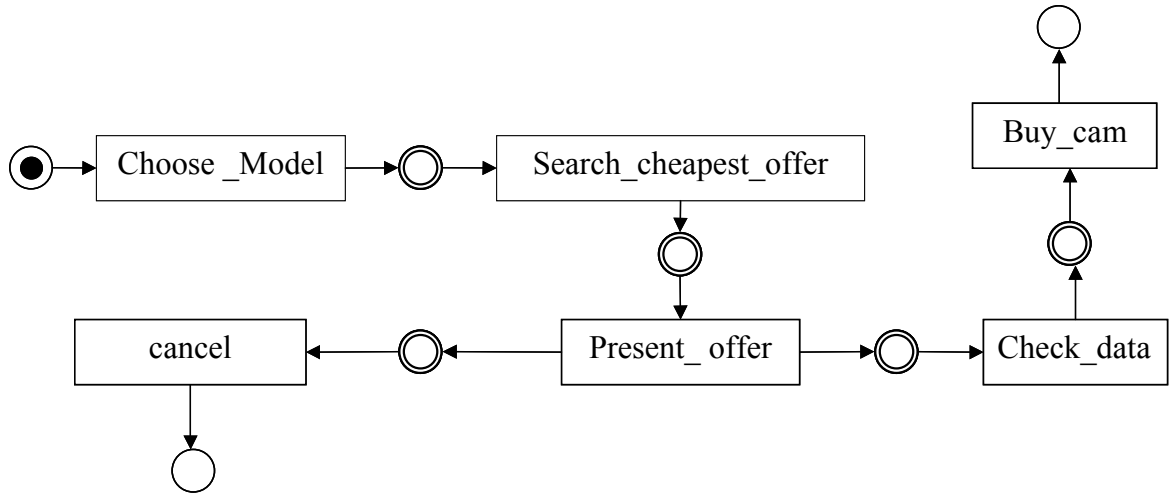
Abbildung 2 - UML Activity Diagramm



B Petrinetze

Petrinetze erweitern das aktivitätsorientierte Paradigma um die zusätzliche Möglichkeit, den Prozesszustand zu beschreiben. Sie werden daher gerne im Workflow - Management - Bereich eingesetzt und sind durch eine sehr formale Definition gekennzeichnet. Im Gegensatz zu UML Diagrammen gibt es in Petrinetzen daher eine klar definierte Semantik mit deren Hilfe eine Verifikation einer modellierten Orchestrierung möglich ist. Auch hier folgt ein weiterer

Abbildung 3 Petrinetz



Vorteil aus der existierenden großen Werkzeugunterstützung.

C Π – Calculus

Die Prozessalgebra Π – Calculus beschreibt in einer formalen Art Orchestrierungen. Ähnlich der Relationenalgebra im relationalen Modell, versucht man hier durch wohldefinierte Terme die Semantik zu modellieren. Es ist in Π – Calculus möglich, Orchestrierungen aufgrund formaler Anforderungen zu verifizieren.[11] Folgende skizzierte Terme veranschaulichen die Notation der Syntax:
Seien A, B Aktivitäten, VAR eine Variable

`A.B`	➔	`A wird vor B ausgeführt`
`A|B`	➔	`A,B werden parallel ausgeführt`
`A+B`	➔	`Es wird A XOR B ausgeführt (Auswahl ist nicht deterministisch)`
`WHILE[VAR = VALUE]A`	➔	`Solange VAR den Wert VALUE besitzt, wird A ausgeführt`

3.3 Datenmodell

(engl.: Data & Data Access Model)

In jeder Beschreibungssprache werden Datenmodelle definiert, um Zugriffe auf Daten und den Transfer von Daten zu gewährleisten. In Orchestrierungen benötigt man zwei unterschiedliche Kategorien an Datenformen. Es existieren zum einen applikationsspezifische Daten, welche die Ein- und Ausgabeparameter des jeweiligen Dienstes repräsentieren und zum anderen Kontrollflussdaten, die im laufenden Kontrollfluss den momentanen Zustand des Prozesses beschreiben. Kontrollflussdaten werden daher auch Prozessdaten genannt und beschränken sich in den meisten Fällen auf wenige Zustandsvariablen, mit denen der Kontrollfluss durch vorhandene Schleifen und Verzweigungen der Orchestrierung geleitet wird.
In Web Services werden applikationsspezifische Daten meistens als Parameter während des Botschaftenaustausches übergen, d.h. sie sind Teil einer XML-Nachricht. Im allgemeinen existieren allerdings zwei Möglichkeiten um applikationsspezifische Daten zu definieren:

(1) Die Parameter eines Dienstes werden in Form eines Unified Resource Identifier (URI) definiert und **referenzieren** damit lediglich auf das entsprechende Datum in Form eines Zeigers. In Web Services benutzt man in diesem Kontext die eindeutige URL (Uniform Resource Locator) als Identifikator. In dieser Methode werden die Daten als Blackbox übergeben, in der keine Information über Struktur und Größe zu finden sind.
(2) Die Datentypen werden **explizit** in einem generischen Schema definiert. Beispielsweise können in XML-basierten Systemen Datentypen mittels eines XML-Schemas definiert werden. Hier erfolgt demnach eine Übergabe des vollständigen Wertes im XML-Dokument. Aufgrund der Tatsache, dass reine XML–Daten durch ihre Größe und zu parsende Struktur jedoch sehr ineffizient zu handhaben sind, gibt es bspw. in SOAP zusätzlich die Möglichkeit, der XML Nachricht binäre Daten als Attachments anzuhängen, wodurch das komplette XML Dokument als Container betrachtet werden kann.(Dieses Prinzip ähnelt den binären, auf MIME-Typen basierenden Anhängen im SMTP Protokoll des E-Mail Verkehrs).

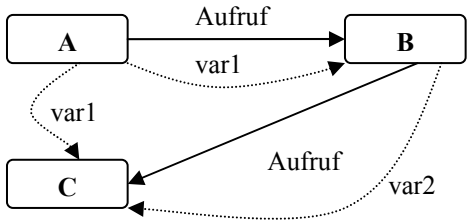
Auch bei der Übertragung von Daten von einem Web-Service-Aufruf A zum nächsten Aufruf B lassen sich zwei Methoden unterscheiden.

(1) Der **Blackboardansatz** speichert alle Daten mit Name und Inhalt in einer Struktur namens Blackboard (analog zum dynamischen Stack im Hauptspeicher). Werden nun Daten als Parameter übergeben, so resultiert der Parameter aus dem Wert des entsprechenden Ein-

trages im Blackboard. Bei Rückgabewerten wird der entsprechende Wert im Blackboard aktualisiert. Jeder Zugriff auf das Blackboard erfolgt atomar. Daraus folgt eine Anforderung an die Serialisierung mehrerer Aktivitäten, welche auf eine überlappende Variablenmenge gleichzeitig schreibend und / oder lesend zugreifen. Eine nebenläufige Ausführung dieser Aktivitäten muss in allen Fällen das gleiche Endresultat vorweisen wie eine serialisierte Ausführung. In allen anderen Fällen wird die Ausführung als unzulässig bewertet.

(2) Der **explizite Datenflussansatz** benutzt spezielle Datenflussverbindungen(Abb.3 gepunktete Pfeile) zwischen zwei Aktivitäten, in denen die jeweilige Datenquelle explizit angegeben werden kann. Der Vorteil dieser Methode ist die größere Ausdrucksmächtigkeit der Sprache, da sich (siehe Abb.3) eine Aktivität A explizit für jeweils eine eindeutige Datenquelle entscheiden kann.[12]

Abbildung 4 expliziter Datenfluss



3.4 Dienstauswahl

(engl.: Service Selection Model)[13]

In einer Choreographie werden verschiedene Dienste aufgerufen. In der Dienstauswahl wird das Konzept definiert, mit dem die jeweiligen Adressen der Dienste an den aktuellen Aufruf gebunden werden. (Im Beispielprozess **Buy_Cheap_Digi_Cam** wird hier das Konzept definiert, mit dem man die verschiedenen Digitalkameraanbieter verwaltet). Man unterscheidet zwischen vier Möglichkeiten:

A Statisches Binden (engl.: static binding)

Beim statischen Binden wird die URI des aufzurufenden Dienstes direkt im Quellcode der Modellierung vermerkt. Der offensichtliche Nachteil dieser Methode ist die Inflexibilität, auf aufkommende, nachträgliche Änderungen der URI des Dienstes zu reagieren.

B Dynamisches Binden mittels Referenzen (engl.: dynamic binding by reference)

Hier bindet man an den Dienstaufruf lediglich eine Variable, in der eine Referenz auf die URI - Adresse des Dienstes verweist. Diese URI - Adresse kann z.B. in einer Datenstruktur mit Web-Service-Adressen (eine Art interne Registrierung) abgelegt sein, aus der jeweils vor einem Aufruf der entsprechende Adresseintrag mit der jeweiligen Variablen verknüpft wird. Diese Liste kann mit diesem Verfahren zur Laufzeit geändert werden, wodurch die Flexibilität und Wartbarkeit der Choreographie deutlich steigt.

C Dynamisches Binden durch Nachschlagen (engl.: dynamic binding by lookup)

Da in UDDI Registrierungen die verschiedenen eingetragenen Dienste mittels WSDL beschrieben werden, und WSDL ein maschinell auslesbares Format besitzt, kann man vor Aufruf eines Dienstes eine Anfrage an die UDDI API schicken, die einem die gewünschten Dienstadressen liefert. Aus dieser resultierenden Menge kann nun ein gewählter Eintrag an den Aufruf gebunden werden.

D Dynamische Operationsauswahl (engl.: dynamic operation selection)

Mit den bisherigen Konzepten beschränkte man das dynamische Binden lediglich auf die Adresse des Dienstes, während der Name der aufzurufenden Operationen statisch definiert wurde. Löst man sich von dieser Denkweise, erhält man die Möglichkeit, sogar die Funktionsaufrufe auf Diensten dynamisch zu organisieren. Die Möglichkeiten Funktionsaufrufe möglichst

dynamisch zu realisieren ähnelt den drei oben genannten und wird hier deshalb nicht mehr wiederholt.

3.5 Fehlerbehandlung

(Exception Handling)

Eine Choreographie zwischen verschiedenen Diensten ist durch die lose Kopplung dieser Dienste charakterisiert. Dienstimplementierungen in SOAP weisen zwar ein Fehlermanagement auf, welches es dem Web Service erlaubt, vordefinierte Fehlernachrichten in auftretenden Fehlerfällen zu generieren, aber eine konkrete Fehlerbehandlung muss in einer Choreographie das weitere Vorgehen im Fehlerfall regeln. Folgende Alternativen sind möglich.

A Flussbasierter Ansatz

Im Abschnitt über Datenmodelle wurden Kontrollflussdaten erwähnt, in denen mittels Statusvariablen der Kontrollfluss des Dienstes durch vorhandene Schleifen und Verzweigungen geleitet wird.
Im flussbasierten Ansatz werden den Statusvariablen eine zusätzliche Menge von Fehlervariablen (Flaggen oder engl.: Flags) hinzugefügt, die anzeigen, ob in dem vorherigen Aufruf ein Fehler aufgetreten ist. Es wird nach jedem Aufruf explizit eine Verzweigung in den Kontrollfluss eingeführt, in der die jeweiligen Fehlerflaggen geprüft werden. Zeigt die Flagge einen Fehler an, so kann explizit im Code kontrolliert werden, welches Verhalten die Orchestrierung einnimmt.
Beispiel im Auszug aus **Buy_Cheap_Digi_Cam**:

```
Boolean     fault;      // Fehlerflag
String      f_type;     // Fehlerbeschreibung
String      product;    // die aktuelle Produktauswahl
String      Service;    // URL des ausgewählten Web Service
```

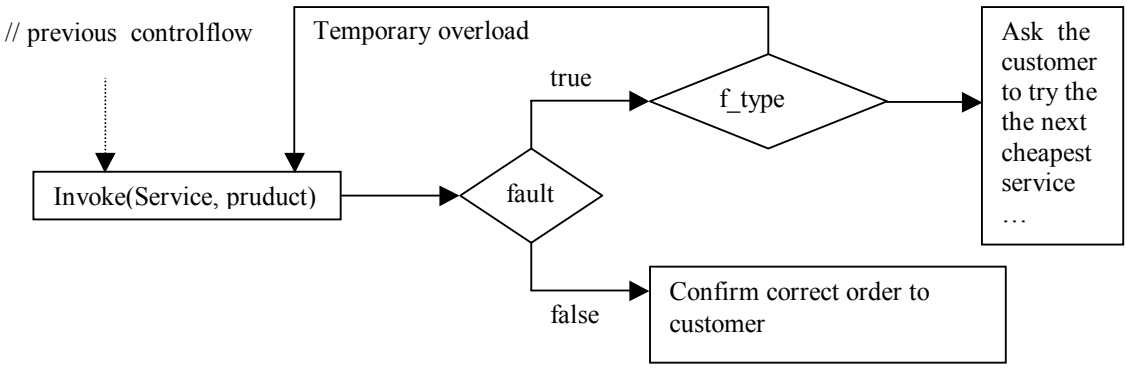


Abbildung 5 Flow based Exception Handling

Der Nachteil des flussbasierten Ansatzes ist die Einbettung der außerordentlichen Fehlerbehandlung in den eigentlichen ordentlichen Kontrollfluss, was zu einem enormen Anstieg der Modellierungskomplexität führt.

B TRY – CATCH – THROW Ansatz

In der Programmiersprache Java wurde die Fehlerbehandlung vom eigentlichen Kontrollfluss getrennt, um eine modulare Modellierung zu ermöglichen. Dieses Prinzip kann auf Orchestrierungen übertragen werden, um die Nachteile des flussbasierten Ansatzes zu beseitigen.
Mittels des try{Anweisung}catch – Konstruktes wird ein Anweisungsblock bestehend aus einem oder mehreren Aufrufen an eine Fehlerbehandlung (**Exception – Handler** mit definier-

ter `CATCH(EXCEPTION e) - Anweisung)` gebunden. Die Fehlerbehandlung überprüft, ob ein Fehlerfall auftritt, und ruft entsprechend eine eigens definierte Fehlerroutine für den jeweiligen Fehler auf. Diese Fehlerroutine unterbricht den aktuellen Kontrollfluss, bis sie abgearbeitet worden ist. Nach der Abarbeitung gibt es mehrere Möglichkeiten fortzufahren:

(1) Der Kontrollfluss führt die nächste Aktivität routinemäßig fort.
(2) Der fehlgeschlagene Aufruf wird wiederholt. (Ein Wechsel der Aufrufparameter ist möglich, um die Konditionen des Aufrufs dem aktualisierten Kontext anzupassen) .
(3) Der Prozess wird beendet.

In geschachtelten Aufrufen ist es möglich, einen Fehlerfall an die darüber liegende Hierarchieebene mit dem THROW Konstrukt hoch zu reichen, wodurch eine Modularisierung der Fehlerbehandlung für Unteraufrufe (Aufruf eines komponierten Dienstes) in einer Hierarchieebene unterstützt wird. Ein Fehlerfall wird implizit in die nächsthöhere Hierarchieebene weitergereicht, wenn kein Exception Handler (`CATCH(EXCEPTION e)` ist nicht definiert) vorliegt. Existiert keine nächsthöhere Hierarchieebene, so wird der Prozess beendet.
Der Vorteil des TRY –CATCH – THROW Ansatzes ist die Kapselung der Fehlerbehandlung in einen eigenen Kontrollflusses mit höherer Priorität.

C Regelbasierter Ansatz (ECA)

In der Datenbankwelt sind ECA Regeln erschaffen worden, welche nach Auftreten eines (operationalen) Ereignisses (engl.: event) E eine Bedingung (engl.: condition) C überprüfen, die angibt ob die Aktion A auszuführen ist oder nicht. Das Tripel (E,C,A) wird Regel genannt. Im Kontext der Fehlerbehandlung prüft die Bedingung C, ob ein Fehlerfall vorliegt und kann mit der Fehlerbehandlung in der Aktion A die Anweisung E rückgängig machen, oder versuchen mit zusätzlichen Anweisungen den Fehlerfall zu beseitigen oder zu umgehen.
Ähnlich dem TRY –CATCH – THROW Ansatz, wird hier klar zwischen der ordentlichen Kontrollflusslogik und der außerordentlichen Fehlerbehandlungslogik beim Einteten der Regel getrennt. Allerdings beinhaltet der Einsatz von ECA Regeln zusätzliche Sprachkonzepte, welche gleichermaßen vom Anwender bzw. der Choreographie verstanden werden müssen.

3.6 Transaktionen

Prozessorientierte Systeme und e-Business Anwendungen benötigen transaktionale Unterstützung, um lose gekoppelte Dienste zu zusammenhängenden Einheiten zu orchestrieren und deren Konsistenz und Zuverlässigkeit zu garantieren [14]. Transaktionale Unterstützung garantiert im engeren Sinne die folgenden vier ACID–Eigenschaften [15]: Atomicy; Consistency; Isolation; Durability. Im Kontext der Orchestrierung von Diensten definiert man Bereiche, deren Kontrollfluss eine sogenannte atomare Semantik besitzen. In einem solchen atomaren Bereich erleiden alle Aktivitäten das gleiche Schicksal. Entweder werden sie erfolgreich beendet, oder sie werden alle abgebrochen. Realisiert werden solche atomaren Bereich durch das 2-Phasen-Commit-Protokoll und sollten von der jeweiligen Middleware der Choreographie unterstützt werden.
Diese Modellierung kann jedoch gerade in den lose gekoppelten Umgebungen, wie man sie in Web-Service-Choreographien vorfindet, zu Problemen führen, da Choreographien über viele Dienste eine relativ lange Laufzeit besitzen. Würden nun bei einem Fehlerfall auch alle bereits erfolgreich abgeschlossenen Dienste zurückgesetzt werden, um den konsistenten Ursprungszustand vor dem Aufruf wiederherzustellen, wären im schlimmsten Fall stundenlange Rechenzeiten umsonst gewesen. Problematisch sind auch die langen Sperren, denen sich die verschiedenen Dienste in lange andauernden Orchestrierungen unterwerfen müssen.

Diese Überlegungen führen zu einer Aufweichung der ACID Eigenschaften, indem bestimmte Bereiche von Dienstaufrufen innerhalb einer Choreographie einen Commit ausführen und somit Ressourcen wieder freigegeben werden können. Außerdem wird sichergestellt, dass die Auswirkungen dieser Bereiche durch zusätzliche kompensierende Aktionen in einem schweren Fehlerfall wieder zurückgesetzt werden können. Differenziert wird zwischen zwei Transaktionstypen:

- **Atomare Transaktion (atomic transaction (AT))**
  typische ACID Transaktionen nach dem „Ganz-oder-gar-nicht“ - Prinzip.
- **Geschäftsaktivität (business activity (BA))**
  lang laufende (LR-) Transaktionen (long running transactions), bestehend aus mehreren aggregierten, hierarchischen und atomaren Bereichen, die selektiv zurückgesetzt werden können.

4 Konkrete Choreographiebeschreibungssprachen

Das Umfeld der Web-Service-Anwendungen ist noch sehr jung, und die Etablierung von WS-Standards (SOAP, WSDL, UDDI) geschah erst vor wenigen Jahren. Zusätzliche Anforderungen, wie das Bilden von Choreographien und transaktionalen Abhängigkeiten zwischen Diensten wurden erst in letzter Zeit ausgesprochen. Um diesem Bedürfnis schnell gerecht zu werden, entschieden sich einige größere Unternehmen, eigene Spezifikationen zu erschaffen. Alle Spezifikationen besitzen jedoch die Gemeinsamkeit, die Extensible Markup Language (XML) als Grundlage zu benutzen.

4.1 Die Babylonische Sprachvielfalt

Um einen Überblick über die verschiedenen entstandenen Spezifikationen zugeben, folgt eine kleine Übersicht aus involvierten Organisationen und dem jeweiligen Erscheinungsjahr.

• BPSS (Business Process Specification Schema)	EbXML	2001
• WSCL (Web Service Conversation Language)	HP	2002
• WSFL (Web Service Flow Language)	IBM	2001
• BPML (Business Process Markup Language)	Intalio	2002
• BizTalk, XLANG	Microsoft	2000, 2001
• PIP (Partner Interface Processes)	RosettaNet	2000
• WSCI (Web Service Choreography Interface)	SAP und Sun	2002

Mit dem Ziel, die eigene Spezifikation zu vervollständigen und zu einem Standard weiter zu entwickeln, entschlossen sich einige Unternehmen, ihre Ansätze zu vereinen oder bei Standardisierungsorganisationen (W3C, OASIS) einzureichen. So verschmolzen WSFL, XLANG miteinander unter dem Namen BPEL4WS und wurden 2003 an OASIS übergeben. Die Sprachkonzepte WSCL und WSCI wurden dem W3C überreicht.

Das folgende Schaubild in Abbildung 5 gibt einen Überblick über den vergangenen Verlauf einiger der entstandenen Spezifikationen.[16]

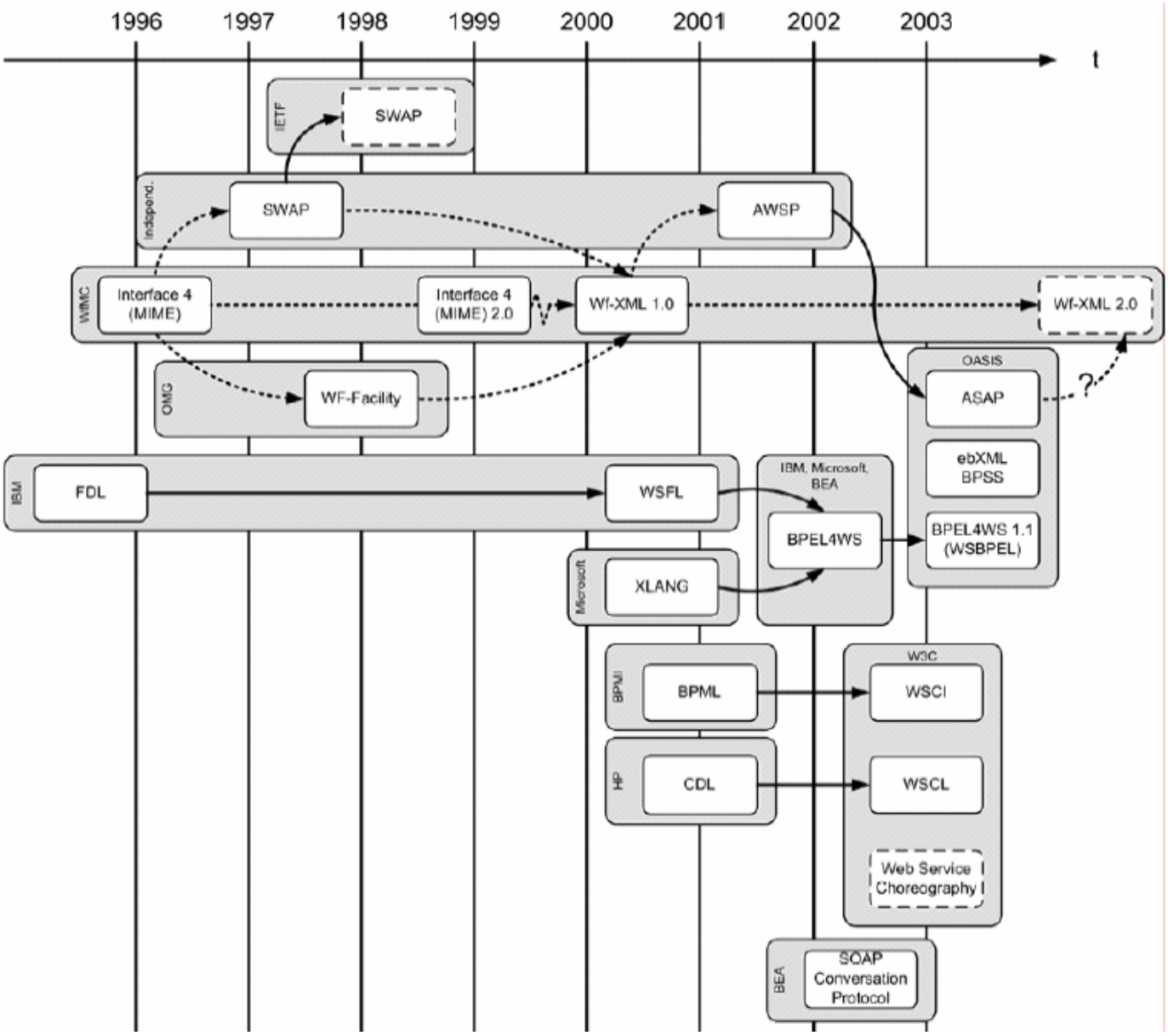


Abbildung 6 Babylonische Sprachvielfalt

Aus den oben genannten Sprachkonzepten wird in den folgenden Abschnitten BPEL4WS anhand der Charakteristika für Choreographiebeschreibungssprachen erörtert. Des Weiteren folgt eine Beschreibung alternativer Sprachansätze wie YAWL (Yet another Workflow Language von W.M.P. van der Aalst) und dem PIP - Konzept aus dem RosettaNet Framework.

4.2 BPEL4WS (Business Process Execution Language for Web Services)

BPEL4WS[17] ist die im Moment gebräuchlichste Choreographiebeschreibungssprache. Zur Abkürzung wird im Folgenden der Präfix BPEL verwendet. Hier wird in der Prozessorientierung unterschieden zwischen „**Executable Business Processes**“ (interne bzw. private Geschäftsprozesse) und „**Abstract Business Processes**“ (abstrakte Geschäftsprozesse), welche auch „**Business Protocols**“ genannt werden. BPEL stellt ein einheitliches Framework für bei-

de Prozesstypen zur Verfügung, welches für den jeweiligen Typus individuell mit zusätzlichen Möglichkeiten erweitert wird. Diese Vorgehensweise erleichtert dem Benutzer eine einheitliche und homogene Prozessmodellierung.

A Komponenten

In BPEL werden die benötigten Komponenten mit Basisaktivitäten angesprochen:

- <invoke> Aufruf einer Operation auf einem Port Type, der von einem Web Services zur Verfügung gestellt wird.
- <receive> Der Prozess blockiert, bis zum Erhalten einer zu definierenden Nachricht.
- <reply> Der Business Prozess sendet eine Nachricht als Antwort auf eine Nachricht, die mit <receive>. empfangen wurde.
- <wait> Die Ausführung wartet eine bestimmte Zeitspanne oder bis zu einem spezifizierten Zeitpunkt
- <throw> generiert eine interne Fehlermeldung
- <empty> ausführen einer „No Op" – Operation, um gegebenenfalls eine Synchronität herzustellen.

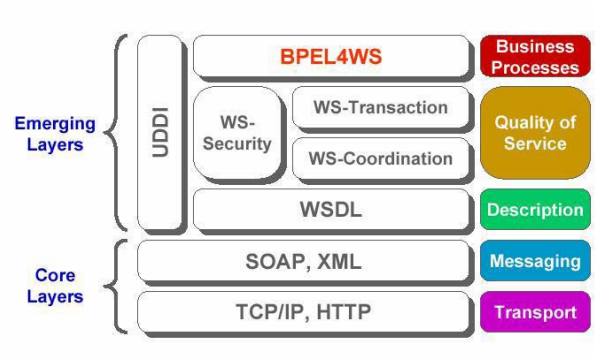


Abbildung 7 Komponentenanforderungen von BPEL4WS

Strukturierte Aktivitäten beschreiben den Kontrollfluss der Orchestrierung eines Prozesses. In BPEL wird im Komponentenmodell vorrausgesetzt, dass die Schnittstellen aller beteiligter Dienste ausschließlich in WSDL spezifiziert sind. Die Abbildung 6 zeigt die Rückgriffe von BPEL im Web Service Technology Stack auf weiter unten befindliche Standards.

B Orchestrierung

Die Orchestrierung in BPEL kombiniert den aktivitätsbasierten Ansatz mit einem hierarchischen Ansatz, da eingebettete Bereiche (engl.: Scopes) definiert werden können, in denen lokale Variablen, eine Fehlerbehandlung und eine Kompensationsbehandlung existieren können.

Der Kontrollfluss wird mit strukturierten Aktivitäten modelliert:

- <sequence> Sequentielle Abarbeitung von Aktivitäten
- <switch> Auswahl eines Kindknotens aus einer Verzweigung mit einer Menge von Aktivitäten
- <pick> Wartet, bis eine spezifizierte Nachricht oder ein „timeout-Alarm" eintrifft. Sodann wird eine festgelegte Aktivität ausgeführt und das <pick> Konstrukt beendet.
- <while> Eine Aktivität wird solange ausgeführt bis eine bestimmte Bedingung erfüllt ist
- <flow> Auswahl einer oder mehrerer Aktivitäten, die nebenläufig ausgeführt werden.

C Datenmodell

BPEL realisiert den **Blackboard-Ansatz**. Angelegte Variablen besitzen einen Namen, einen Datentyp und einen Wert und können wie in herkömmlichen Programmiersprachen als Parameter oder in Ausdrücken benutzt werden, jedoch müssen die Datentypen mittels WSDL oder in einem XML - Schema spezifiziert worden sein. Zur Spezifikation der Variablendeklaration siehe [A]. Das Attribut „messageType“ zeigt auf eine WSDL „message type“ – Definition. Das Element „type“ zeigt auf einen XML Schema Basistyp (engl.: simple type) und „element“ zeigt auf ein XML Schema Element, welches einen komplexen Typ charakterisiert. Nur eines dieser Attribute kann für jeweils eine Variable benutzt werden. Nur Variablen mit dem „message type“ Attribut können als Parameter in <invoke>, <receive>, und <reply> Aktivitäten eingesetzt werden, da hier WSDL vorrausgesetzt wird.
Variablen, die in eingebetteten Bereichen (engl.: scopes) deklariert werden verhalten sich innerhalb des Bereichs wie herkömmliche lokale Variablen. Werden Variablen auf der höchsten Hierarchieebene deklariert, gelten sie als globale Variablen. Mittels der <assign> - Aktivität können Variablen Werte zugewiesen werden (siehe [B]).
In abstrakten Prozessen ist es möglich, bestimmten Variablen die Eigenschaft „opaque“ zu geben, wodurch es erlaubt wird, eventuelle Zuweisungen dieser Variablen innerhalb des abstrakten Prozesses nicht näher zu spezifizieren. Konkrete Zuweisung werden durch den ausführbaren BPEL - Prozess einer Implementierung nachgereicht.

D Dienstauswahl

In BPEL kann die Dienstauswahl beliebig, statisch oder dynamisch zur Laufzeit, durch den Entwickler implementiert werden. Das Binden an eine Adresse wird durch das sogenannte Partnerkonzept realisiert. Ein „**partnerLinkType**“ definiert eine Beziehung zwischen zwei Rollen. Jede Rolle besitzt eine eigene, in WSDL angelegte Endpunktreferenz genannt „**portType**“, über welche die Kommunikation realisiert wird. Um einen konkreten „**partnerLink**“ zu definieren weist man nun den teilnehmenden Partnern die entsprechende Rolle zu. Zum Anlegen eines portTypes siehe [C], eines partnerLinkType siehe [D] und eines konkreten partnerLinks siehe [E].

E Exception Handling & Transaktionen

Fehlerbehandlung und transaktionale Verarbeitung werden in BPEL4WS Hand in Hand spezifiziert. Durch das „scope“ – Konzept(siehe [F]) können verschachtelt strukturierte Aktivitäten mit einem Kontext versehen werden, in dem ein eigener lokaler Variablenbereich, eine eigene Fehlerbehandlung (hier: fault - handler) und eine eigene Kompensationsverwaltung (compensation - handler) eingebettet sind.
BPEL realisiert einen TRY-CATCH-THROW Ansatz in der Fehlerbehandlung. Eine Fehlerbehandlung existiert innerhalb eines „scopes“, wenn die CATCH – Komponente im Kontrollfluss definiert ist. Der allgemeine Aufbau eine Fehlerbehandlung ist in [G] erläutert.
BPELs Vorgehen bei der Fehlerbehandlung und Kompensation kann gut mit Hilfe des BusinessAgreement-Protokolls aus der WS-Transaction-Spezifikation (siehe Abb.7) beschrieben werden, wobei die Rolle des Koordinators hier einem umgebenden Scope (Vater) zufällt, und jeder eingebettete Scope für sich die Rolle des Participant übernimmt:

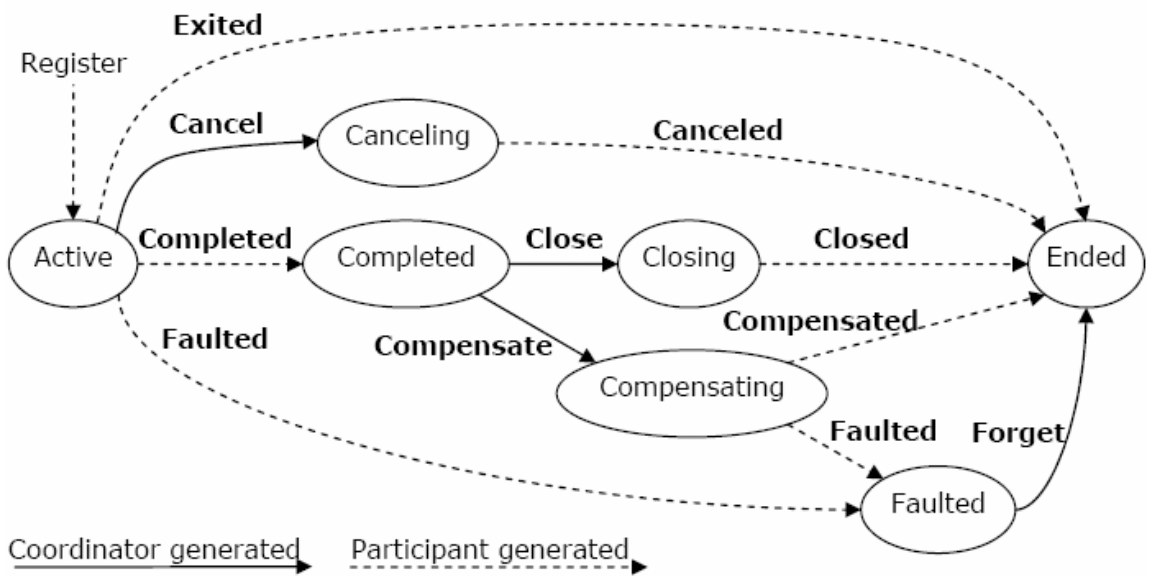


Abbildung 8 BPEL Koordination eines Fehlerfalles in einer transaktionalen Umgebung

(1) Wenn ein eingebetteter Scope erfolgreich beendet ist, wird ein Compensation Handler für diesen Scope angelegt. Ein „**Completed**“ Signal wird an den Scope der nächst höheren Hierarchieebene gesendet. (Vater)

(2) Ein eingebetteter Scope löst einen internen Fehlerfall aus und wird somit nicht erfolgreich beendet.
   a) Falls der Fehler weitergeworfen wird, wird ein „**Faulted**“ Signal zum Vater gesendet.
   b) Wird ein Fehler im eingebetteten Scope behandelt und nicht weitergeworfen, so beendet der Scope seinen Kontrollfluss und versendet an seinen Vater ein „**Exited**“ Signal.

(3) Nach dem erfolgreichen Beenden eines eingebetteten Scopes, kann er durch Aufruf seines Compensation Handlers mittels eines „**Compensate**“ Signals durch den Fault – oder Compensation Handlers des Vaters zurückgesetzt werden.

(4) Nach seinem erfolgreichen Kompensieren schickt der kompensierte Scope ein „**Compensated**“ Signal zum Vater.

(5) Falls während der Kompensation ein Fehler auftritt:
   a) Falls der Fehler nicht innerhalb des Scopes behandelt wird, wird er mittels eines „**Faulted**“ Signals an den Vater weitergeworfen
   b) Falls der Fehler innerhalb des Scopes behandelt wurde, wird eine erfolgreiche Kompensation vorausgesetzt und ein „**Compensated**“ Signal wird an den Vater versendet.

(6) Falls im Vater unabhängig vom laufenden eingebetteten Scope ein Fehler auftritt, so wird mittels eines „**Cancel**“ Signals der aktuell laufende eingebettete Scope abgebrochen.

(7) Ein eingebetteter Scope, der ein „**Cancel**“ Signal empfängt, bricht seinen Kontrollfluss ab, führt nach obiger Beschreibung eine Fehlerbehandlung durch, schickt ein „**Cancelled**“ Signal an den Vater und wird beendet

(8) Falls der Vater eine Kompensierung des eingebetteten Scopes für nicht mehr nötig befindet, schickt er ihm ein „**Close**“ Signal. Nachdem dieser seinen Compensation Handler abgelegt hat, antwortet er mit einem „**Closed**“ Signal.

(9) Im Falle einer Race Situation zwischen einem „**Completed**“ und einem „**Cancel**“ Signal, wird das „**Cancel**“ Signal vom eingebetteten Scope ignoriert.
(10) Falls ein fehlerhafter, eingebetteter Scope ein „**Cancel**“ Signal empfängt, so ignoriert er das Signal und kann mit einem „**Faulted**“ oder einem „**Exited**“ Signal antworten.

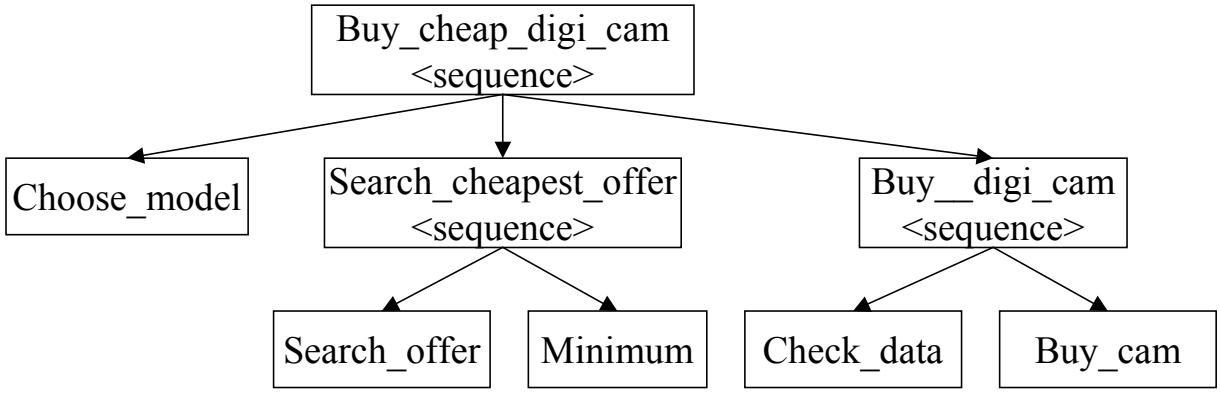
Um transaktionale Unterstützung komponierter Dienste in öffentlichen Prozessen zu gewährleisten, wird in BPEL auf die WS - Transaction Spezifikation verwiesen. Somit lässt sich zusammenfassend sagen, dass Long Running Transactions sich in BPEL im „**Executable Business Process**“ durch das Scope - Konzept umsetzen lassen. Bei „**Business Protocolls**“ hingegen, lassen sich Transaktionen, die sich aus Aktivitäten von mehreren beteiligten Partnern zusammensetzen, lediglich mit der WS-T Spezifikation realisieren.

F Correlation Sets

Um innerhalb einer Kommunikation zwischen zwei Instanzen jeder verschickten Nachricht eine eindeutige Identifikation der betreffenden Kommunikationspartner verschiedener Instanzen zu ermöglichen, wird in BPEL ein sogenanntes „**Correlation Set**“ für eine Kommunikation (abstrakt: Correlation) definiert. Hier sind global eindeutige Zuordnungen für die momentanen Kommunikationspartner verzeichnet, in denen bspw. die eindeutige Kunden - ID und Anbieter - ID gekennzeichnet sind, um eine Antwort bzw. eine Weiterverarbeitung der Information den teilnehmenden Prozessinstanzen zuordnen zu können. (siehe[G])

G Buy_Cheap_Digi_Camin

Im anschließenden Beispiel von **Buy_Cheap_Digi_Cam** wird die Orchestrierung anhand eines UML-Activity Diagramms skizziert.



4.3 YAWL (Yet Another Workflow Language)

YAWL[18] ist eine Choreographiebeschreibungssprache, deren Wurzeln im Workflow Management anzusiedeln sind. Das interessante Konzept von YAWL ist die Realisierung eines theoretisch fundierten Unterbaus der Orchestrierung in Form eines erweiterten Petrinetzes (engl.: extended workflow nets (EWF-nets)[19]). Das in Abb.8 folgende Framework von YAWL ist aktuell leider noch nicht vollständig realisiert, da die Implementierung der verschiedenen Module noch nicht abgeschlossen ist.

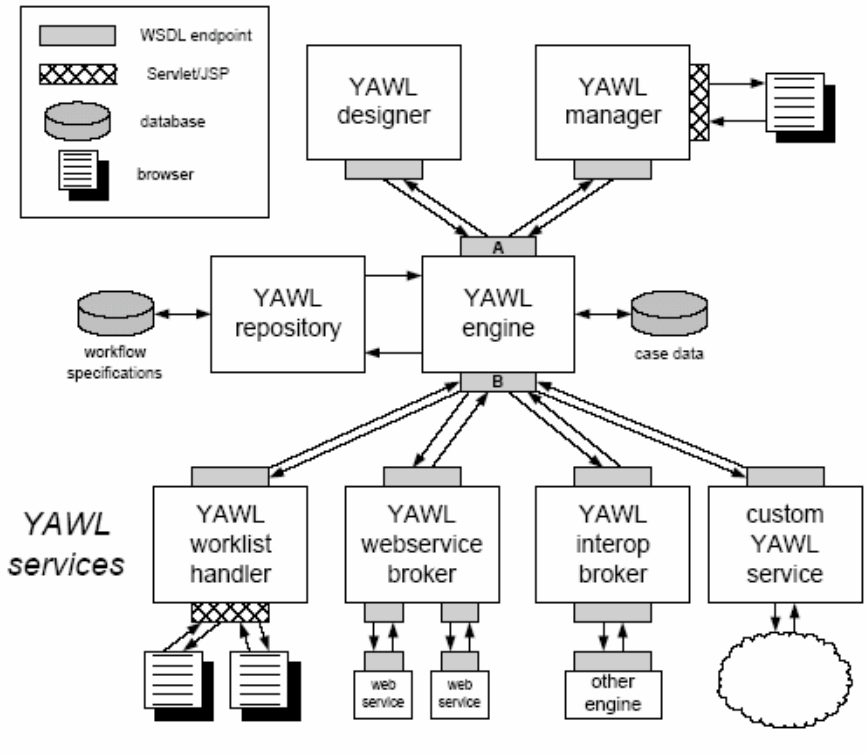


Abbildung 9 Das YAWL Framework

Den Kern des YAWL - Frameworks bildet die **YAWL - Engine**, in der bereits existierende Workflow-Spezifikationen instanziiert und verwaltet werden können. Neue Workflow-Spezifikationen können mit dem **YAWL - Designer** angelegt werden. Bestehende Workflow-Spezifikationen werden in der **YAWL - Repository** verwaltet, welche durch die YAWL - Engine erreicht werden kann. Ein Anwendungsfall (engl.: Case) ist ein Geschäftsprozess in Ausführung und korrespondiert zu einer instanziierten Workflow-Spezifikationen. Der **YAWL - Manager** erlaubt manuelle Kontrolle über Workflowinstanzen, indem er dem Benutzer Informationen über den aktuellen Zustand einer aktuellen Instanz und Details über bereits abgeschlossene Instanzen bereitstellt.
YAWL wurde vom Web-Service-Paradigma inspiriert und abstrahiert Endbenutzer, Applikationen, und Organisationen als Dienste. Folgende Services stellt YAWL zur Verfügung, allerdings können weitere Dienste hinzugefügt werden:

(1) **YAWL - Worklist Handler**
Durch den YAWL - Worklist Handler können Aufgaben bzw. Aktivitäten weiter an Benutzer delegiert werden. Weiterhin können hier Aufgaben angenommen oder das Fertigstellen von Aufgaben signalisiert werden.

(2) **YAWL - Web Services Broker**
Der YAWL - Web Services Broker bildet die Schnittstelle zwischen der YAWL - Engine

und weiteren Web Services. Durch diesen Dienst kann die YAWL - Engine externe Web Services erreichen und einen Aufruf starten.

(3) **YAWL - Interoperability Broker**
Der YAWL - Interoperability Broker ist ein Dienst, der verschiedene YAWL - Workflow - Engines miteinander verbindet. So könnte z.B. eine Aktivität an eine weitere YAWL - Engine delegiert werden, in der diese Aktivität durch einen eigenen Prozess abgearbeitet wird.

(4) **Custom - YAWL Services**
Ein Custom YAWL - Service verbindet die YAWL - Engine mit weiteren Diensten der lokalen Geschäftsumgebung. Es könnte ein eigener Worklist - Handler oder ein zusätzlicher Dienst sein, der eine Verbindung zu mobilen Kommunikationsgeräten (z.B. Handy) herstellt.

A Komponenten

Das YAWL Framework wurde so konzipiert, dass die verschiedenen YAWL Komponenten über Schnittstellen miteinander kommunizieren, die in WSDL spezifiziert sind. Ein YAWL - Web Service Broker benötigt in WSDL beschriebene Web Services, um Zugang zu den öffentlichen Schnittstellenmethoden zu erlangen.

B Orchestrierung

Durch die Wurzeln im Workflow Management wird in YAWL versucht, durch Orchestrierungen bestimmte Workflow Patterns[20] zu realisieren. Workflow Patterns sind standardisierte Ablaufszenarios. Einfache Beispiele von Workflow Patterns wären:

- **Einfache Kontrollfluss Patterns**
  Einfache, häufig vorkommende Konstrukte mit denen der Großteil des Kontrollflusses geregelt wird. Beispiele wären Sequenzen von Aktivitäten, parallele Aktivitäten and bedingte Aktivitäten.
- **Erweiterte Verzweigungen und Synchronisations- Patterns**
  Diese Patterns erlauben fortgeschrittene Operationen wie z.B. eine AND-Verzweigung, bei der immer nur alle oder keine ausgehenden Zweige ausgeführt werden, oder eine OR-Synchronisation, bei der mindestens eine der eingehenden Zweige aktiv sein muss um weiter vorgehen zu können.

Die Syntax einer YAWL-Prozessdefinition wird folgendermaßen (siehe Abb.9) definiert:

Condition - Conditions besitzen eine äquivalente Semantik zu den Stellen der Petrinetztheorie. Sie definieren den aktuellen Zustand des Systems

Input / Output Condition - Jede Prozessdefinition enthält jeweils einen eindeutigen Startzustand (hier: Input Condition) und einen eindeutigen Endzustand (Output Condition).

Atomic / Composite Task - Ein Task besitzt eine äquivalente Semantik zu der Transition in der Petrinetztheorie und ähnelt der **Aktivität** aus BPEL4WS. Im Gegensatz zu einfachen Petrinetzmodellen ist es in EWF-nets möglich, mehrere Transitionen hintereinander zu schalten, ohne jeweils eine explizite Stelle an den Übergängen zu modellieren. Jeder Übergang an sich kann als implizite Stelle interpretiert werden (siehe Abb.10). Composite Tasks sind zusammengesetzte Aktivitäten und referenzieren weitere Prozessdefinitionen, während Atomic Tasks einzelne Aktivitäten darstellen.

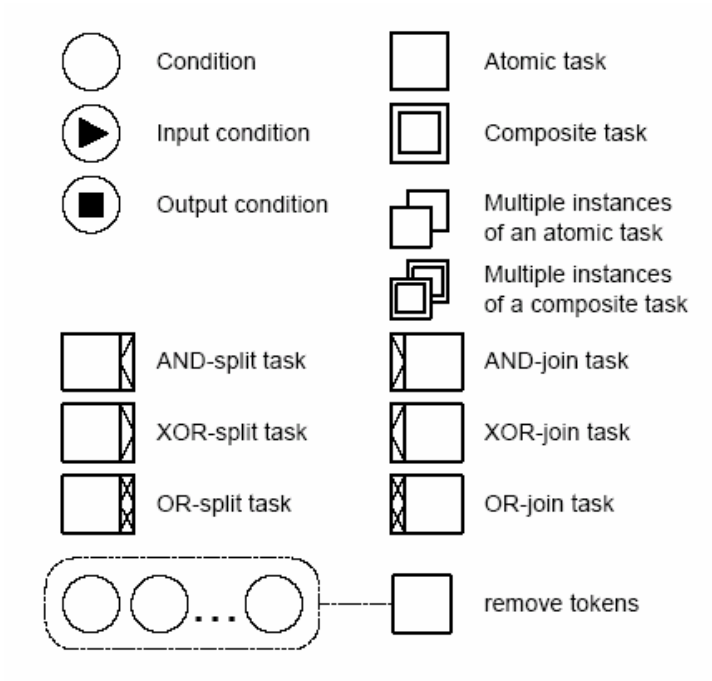


Abbildung 10 Symbole in YAWL

In YAWL stehen dem Anwender sechs neuartige Aktivitäten zur Verfügung. Sie verallgemeinern die zwei Petrinetz - Konzepte:

- Parallelisierung und
- Synchronisation.

Das Beispiel in Abb.10 veranschaulicht die Einsatzmöglichkeit einer OR - Verzweigung (OR - Split Task), s.d. mindestens ein ausgehender Zweig (flight, hotel oder car) gewählt werden muss. Dieses Szenario stellt das Workflow Pattern „Multi Merge“ dar. Die XML-Darstellung ist in [I] für den Task „register“ aufgeführt.

C Datenmodell

YAWL stützt sein Datenmodell auf die schon bestehenden Standards XPath und XQuery. Es ist in YAWL möglich, Variablen ähnlich dem vorgestellten **Blackboard - Ansatz** zu deklarieren und ihnen Werte zuzuweisen, um z.B. Bedingungen zu prüfen, neue Instanzen zu kreieren etc.. Das Element „localVariable“ erlaubt es einem Task auf seiner Hierarchieebene eine typisierte lokale Variable zuzuweisen[J]. Eine beispielhafte Variablendeklaration für das Szenario aus Abb.10 ist in [K] dargestellt. Es ist natürlich möglich, die lokalen Variablen eines zusammengesetzten Tasks einem untergeordneten Task als Eingabeparameter zu übergeben und Resultate in Form von Rückgabeparametern wieder geliefert zu bekommen. Diese Vorgehensweise lässt sich am durch den **explizite Datenflussansatz** klassifizieren**.** Am Beispiel der Dekomposition des Tasks „Register“ wird diese Parameterübergabe in [L] kurz skizziert.

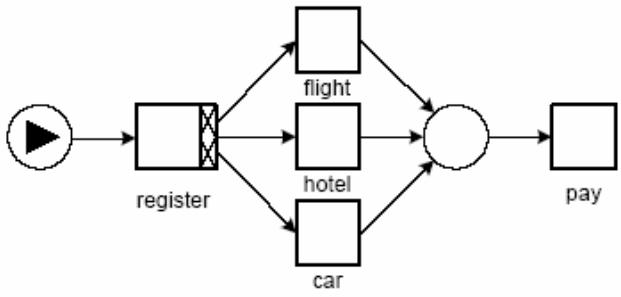


Abbildung 11 Beispiel einer YAWL-Prozessdefinition

D Dienstauswahl

Beim Anlegen einer neuen Workflowspezifikation, werden von der YAWL - Engine alle vorkommenden Dekompositionen (siehe decomposesTo – Element in [I]) von atomaren Tasks mit mindestens einem YAWL - Service registriert. Im Beispiel der Abb.10 würden die Tasks „flight“, „hotel“ und „car“ in einem YAWL - Worklist Handler registriert werden, während der Task „pay“ im YAWL Web Service Broker registriert wird. Das Anlegen einer Workflowspezifikation gilt nur dann als erfolgreich, wenn alle Task - Dekompositionen registriert werden können. Es folgt somit eine Einordnung der Dienstauswahl in ein eher sta-

tisch orientiertes Binden von Adressen zur Instanzbildung. In einer Task - Dekomposition [L] befinden sich minimal die Informationen über die Ein- und Ausgabeparametertypen. Zusätzlich können Angaben gemacht werden, in welchem der YAWL Services sich der Task registrieren soll:

- **YAWL - Worklist Handler**
  Die Task - Dekomposition muss eine bestimmte Menge von Rollen spezifizieren, welche die Instanz des Tasks sehen bzw. aus der Worklist nehmen und bearbeiten dürfen.
- **YAWL - Web Service Handler**
  Entsprechend des Komponenten Models von YAWL muss ein aufrufbarer, externer Dienst mit einer in WSDL beschriebenen Schnittstelle versehen sein. Die Dekomposition enthält Informationen über den WSDL – portType, um ein Binden an die Dienstschnittstelle zu ermöglichen.
  Analog zum Beispiel der Aktivität „pay“ in Abb.10 welche auf den externen Dienst „Payment Service“ referenziert, erfolgt die Koordination in Pfeilrichtung zwischen dem YAWL Service Interface I und der Operation des Payment Services P mit folgender exemplarisch skizzierter Protokollschnittstelle:
  - `I.createTaskInstance()` → `P.InitiateOnlinePayment()`
  - `I.taskStarted()` ← `P.OnlinePaymentInitiated()`
  - `I.taskCompleted()` ← `P.OnlinePaymentCompleted()`
  - `I.cancelTask()` → `P.CancelPayment()`

  Der letztendliche Aufruf erfolgt über das in WSDL spezifizierte Web Service Invocation Framework beschrieben in [http://ws.apache.org/wsif].
- **YAWL - Interoperability Broker**
  Die Task Dekomposition muss folgende Informationen bereitstellen:
  - Identifikator der Ziel YAWL - Engine B
  - Der Name des Prozesses, wenn der Task in der Engine B instanziiert wird.
  - Eine Abbildung der Eingabeparameter der Taskinstanz der Engine A zu denen der Prozessinstanz der Engine B
  - Eine Abbildung der Ausgabeparameter der Prozessinstanz der Engine B zur Engine A.

E Exception Handling und Transaktionen

Die Fehlerbehandlung in YAWL erfolgt durch die „Remove Tokens“ - Aktivität. Wird diese Aktivität zur Laufzeit eines Prozesses gewählt, so werden alle existierenden Tokens des Prozesses aus dem Bereich des Petrinetzes entfernt, wodurch die aktuelle Ausführung abgebrochen wird. Alle weiteren Aktivitäten, die eine „Remove Tokens“ – Aktivität danach einleitet, (siehe Abb.11 – ausgehende Kante von cancel_activity) können individuell definiert werden.

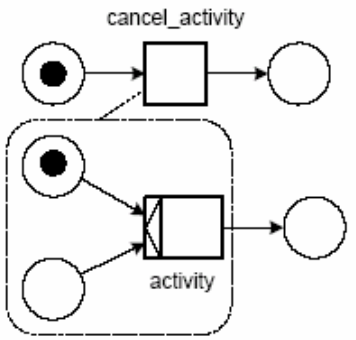


Abbildung 12 Fehlerbehandlung in YAWL

Transaktionale Verarbeitung von Prozessen muss in YAWL per Hand vom Anwender durch den Einsatz von „Composite Tasks“ und „Remove Tokens“ – Aktivitäten realisiert werden und wird momentan nicht von der YAWL - Engine unterstützt. Des Weiteren existieren keine vom System zur Verfügung gestellte Kompensationsmechanismen.

4.4 RosettaNet

Das RosettaNet Framework[21] entsprang einer Initiative zwischen dem Elektronikgroßhändler Ingram Micro und dem Hardwarehersteller 3Com. Es sollte eine EDI (Electronic Data Interchange) Lösung zum elektronischen Abwickeln von Supply – Chain - Prozessen über das Internet entwickelt werden. Das Interesse anderer Unternehmen an dieser Initiative war jedoch so groß, dass 1998 aus der ehemaligen Einzellösung in Zusammenarbeit von 40 Unternehmen ein eigenes Konzept unter der Schirmherrschaft der gemeinnützigen Organisation RosettaNet entsprang. Momentan beteiligen sich circa 400 Unternehmen am Konzept RosettaNet.

A Der Stein von Rosetta

RosettaNet wurde und wird auch heute noch durch den Stein von Rosetta inspiriert, der 1799 in Ägypten entdeckt wurde. Dieser Stein von Rosetta stammt aus dem Jahre 196 v.Chr. Er enthält einen eingemeißelten Gesetzestext in drei Sprachen (siehe Abb.12 Hieroglyphen (oben), Demotisch (in der Mitte) und Griechisch (unten)).Durch diese Dreisprachigkeit wurde es erstmals möglich, ägyptische Hieroglyphen zu entschlüsseln.
Hieraus ergab sich die leitende Direktive von RosettaNet. Man möchte Sprachbarrieren zwischen Organisationen beseitigen, um eine bessere Kommunikation zu ermöglichen.

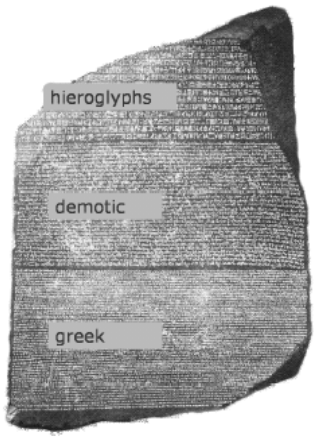


Abbildung 13 Stein von Rosetta

B Standards

RosettaNet steht für eine offene Standardisierung zwischenbetrieblicher Prozesse, um das Supply - Chain Management für die beteiligten Geschäftpartner zu erleichtern. Da allerdings 1998 noch keine Web Service Spezifikationen (SOAP, WSDL) auf dem Markt existierten, musste RosettaNet auf eigenen Lösungen bauen.
Die RosettaNet Standardisierungsbemühungen gliedern sich in folgende Kerngebiete:

- **Geschäftsprozesse**
  Die offensichtlichen Bestrebungen sind auf die Koordination der verteilten Geschäftsprozesse zwischen den teilnehmenden Geschäftspartnern ausgerichtet. Das Konzept der „RosettaNet Partner Interfaces Processes“ (PIP) definiert entsprechende Protokolle. In jedem PIP werden Rollen, Geschäftsdokumente, Geschäftsvokabeln und die Orchestrierung der Nachrichten definiert.
- **Datenformate**
  Jedes Unternehmen arbeitet mit unterschiedlichen Begriffen bzw. Vokabeln in ihren Geschäftstätigkeiten. Um eine eindeutige Kommunikation mehrerer Unternehmen zu ermöglichen, müssen diese Begriffsbarrieren überwunden werden, indem alle aufkommenden Begriffe und Definitionen global spezifiziert werden. RosettaNet bietet hier zwei Arten von Verzeichnissen an:

  - **RosettaNet Business Dictionary (RNBD)**
    Hier werden die Begriffe bestimmt, die in den grundlegenden Aktivitäten der Geschäftsprozesse benutzt werden. So könnte hier z.B. geklärt werden, dass das Zeichen für die europäische Währung EUR lautet. Ein € - Zeichen wäre demnach ungültig.

- **Technical Dictionary (RNTD)**
  Hier werden gebräuchliche Ausrücke definiert, um Produkte und Dienste zu beschreiben.

- **Nachrichtendienste**
  Um einen definierten PIP auszuführen, stellt RosettaNet dem Anwender das „RosettaNet Implementation Framework“ (RNIF) als Middleware zur Verfügung. Hier wird die Unterstützung für zu verschickende Nachrichten spezifiziert. RosettaNet unterscheidet zwischen folgenden Nachrichtentypen:

  - **Business Action Nachrichten** – Sie enthalten Nutzdaten wie Geschäftsdokumente (z.B. Kaufaufträge oder Rechnungen) und werden in den PIP spezifiziert
  - **Business Signal Nachrichten** –Sie quittieren automatisch den Erhalt einer Business Action Nachricht positiv bzw. negativ.

  Da der Ursprung von RosettaNet vor den derzeitigen Web Service Standards liegt, verwendet RosettaNet in der ursprünglichen Form nicht SOAP zum Nachrichtentransport. Beide Nachrichtentypen werden vor dem Versenden in eine Art Umschlag verpackt, der im RNIF näher spezifiziert ist. Insgesamt werden dem RNIF folgende Aufgaben zugeordnet: Verpacken von Nachrichten, Transport von Nachrichten, Sicherheit von Nachrichten, Routen von Nachrichten

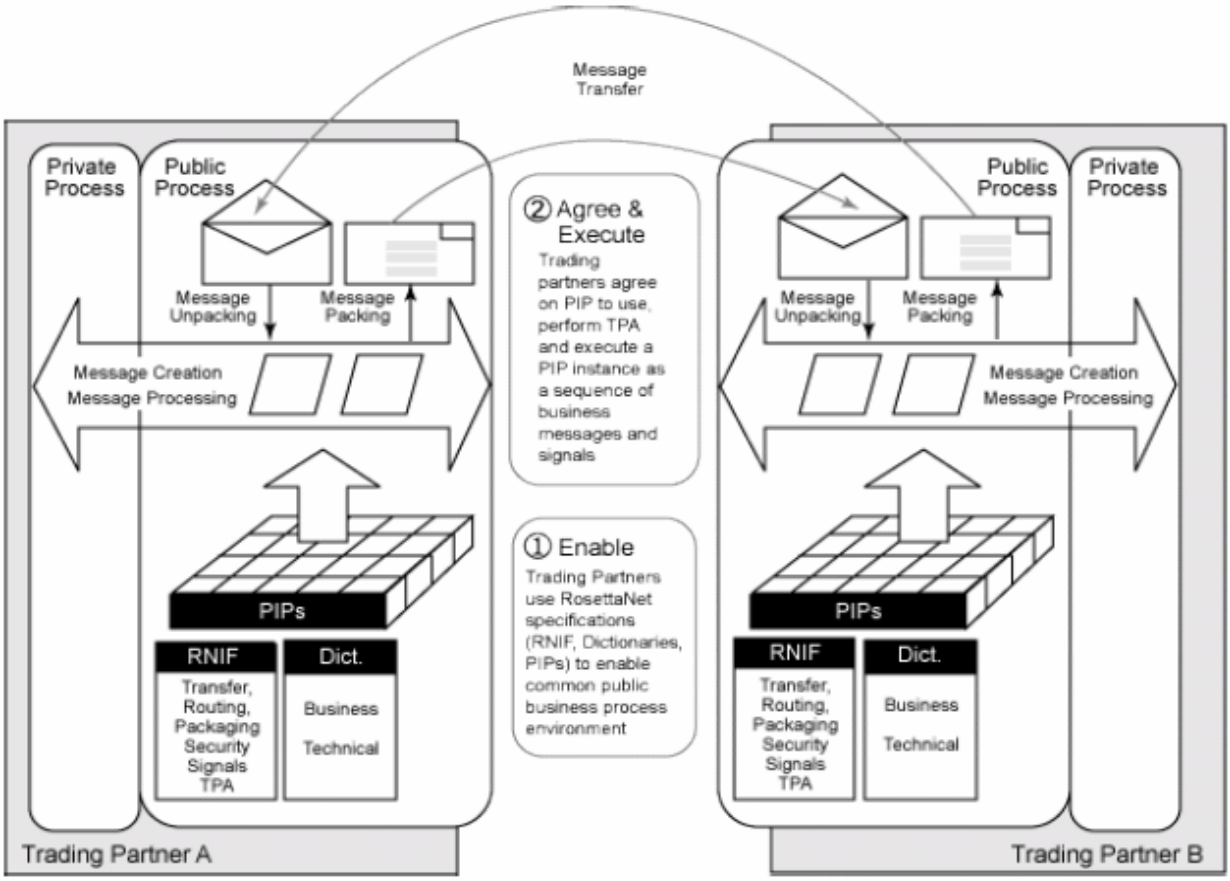


Abbildung 14 RosettaNet Standards zur Kommunikation

Die Abbildung 13 zeigt graphisch wie sich die Kommunikation in RosettaNet aufgrund definierter Kerngebiete zwischen den öffentlichen Prozessen zweier Geschäftspartner aufbaut. Im folgenden Abschnitt wird nun das Konzept der PIP näher betrachtet.

C Orchestrierung durch PIP's

Die RosettaNet PIP-Spezifikation beschreibt, wie die Zusammenarbeit der Geschäftspartner in den öffentlichen Prozessen koordiniert wird. Um eine Kommunikation zu ermöglichen, enthält jeder PIP ein Technical Dictionary, welches die auszutauschenden Komponenten innerhalb des Prozesses erklärt und ein auf den einzelnen PIP bezogenes Business Dictionary, welches die Geschäftsmodalitäten klärt. In RosettaNet wählt man einen Top-Down-Ansatz in vier Schritten, um ein jeweiliges PIP aus einem schon bestehenden Geschäftsprozess zu kreieren:

(1) Im ersten Schritt wird ein Geschäftsmodell erstellt, in dem beschrieben wird, wie die geschäftliche Tätigkeit zwischen den Geschäftspartnern momentan behandelt wird.
(2) Im zweiten Schritt wird ein Re – Engineering des erstellten Modells durchgeführt, um die netzbasierte Kommunikation zwischen den Geschäftspartnern einzubinden.
(3) Im dritten Schritt wird ein sogenanntes PIP Blueprint Document generiert, in dem gemäß der Definition eines Geschäftsprozesses beschrieben wird, auf welche Art und Weise die Rollen, welche die teilnehmenden Geschäftspartner in dem Geschäftsprozess einnehmen, in ihrem Zusammenspiel die Zielsetzung des Prozesses erreichen. Die teilnehmenden RosettaNet-Mitglieder stimmen im Anschluss daran ab, ob der Blueprint anerkannt wird.
(4) Im letzten Schritt wird ein PIP Protokoll erstellt, in dem beschrieben wird, wie die Kommunikation im Netzwerk zu realisieren ist.

Ein vollständiger PIP wird in ein RosettaNet **PIP Specification Package** gepackt. Ein PIP Specification Package ist ein im ZIP – Format erstelltes Archiv welches drei Dokumenttypen

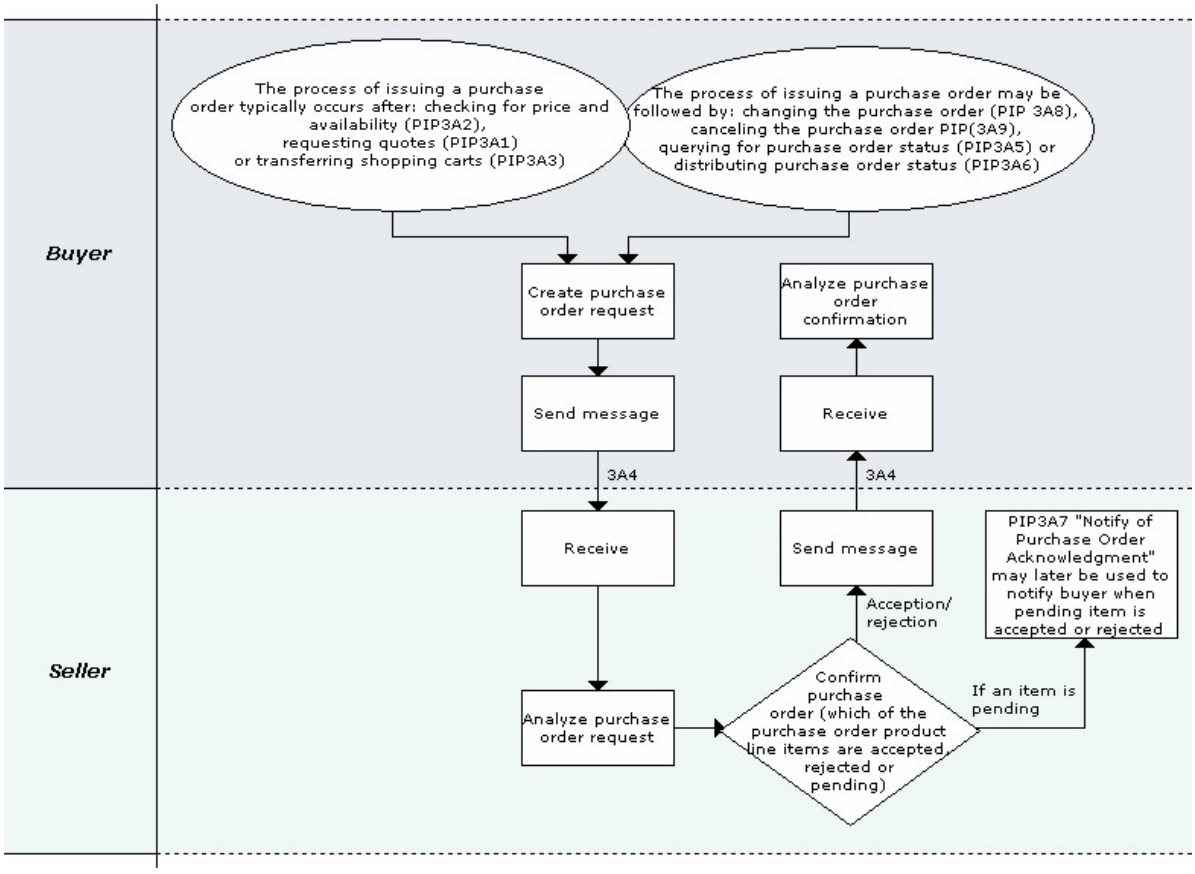


Abbildung 15 UML Diagramm PIP3A4

enthält. Die PIP Spezifikation an sich ist ein MS-Word-Dokument. Hilfen und Richtlinien für definierte Nachrichtentypen liegen in HTML Format vor. Die Nachrichtenstruktur und Inhalte sind im XML-DTD-Format angelegt. Im Folgenden wird der Aufbau der PIP-Spezifikation anhand des PIP PIP3A4: Request Purchase Order erklärt (sieh Abb.14).
Eine PIP-Spezifikation besteht aus drei Hauptpunkten:

- **Business Operational View (BOV)**
  In dieser Sicht (auch Action Layer genannt) wird der eigentliche Geschäftsprozess mit seiner konkreten Zielsetzung definiert. In einem Flussdiagramm (Abb.15) wird außerdem der Kontrollfluss mit seinen Zuständen zwischen den involvierten Geschäftspartnern (hier Buyer und Seller) geklärt. Gemäß der Blueprint-Dokumentation werden hier außerdem Rollen, Rollentypen und Interaktionen zwischen Rollen festgelegt. Es sind drei definierte Rollentypen in RosettaNet vorgesehen:

  - **Organizational** – Rolle wird von einer Organisation eingenommen
  - **Employee** – Rolle wird von einem Angestellten der Organisation angenommen
  - **Functional** – Rolle kann von Organisation oder von einem Angestellten angenommen werden.

  Zusätzlich dazu werden die benötigten Geschäftsdokumente und ein auf den entsprechenden PIP zugeschnittenes Business - Dictionary angelegt.

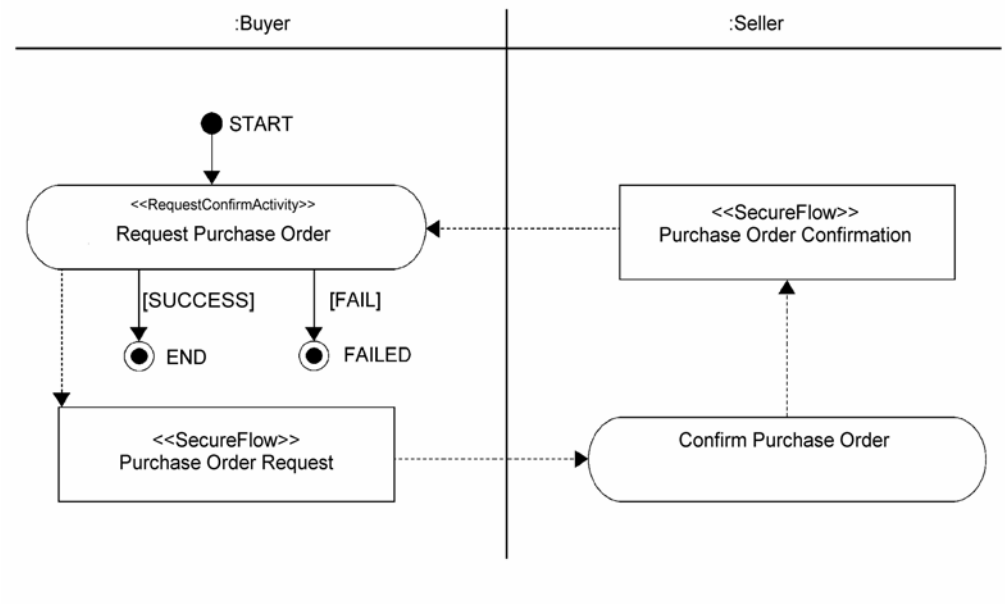


Abbildung 16 Flussdiagramm Request Purchase Order

- **Functional Service View (FSV)**
  Die Functional Service View (auch Transaction Layer genannt) ist abgeleitet von der Business Operational View und definiert die Koordinationsprotokolle zwischen den teilnehmenden Komponenten. So definiert das **Network Component Model** (siehe

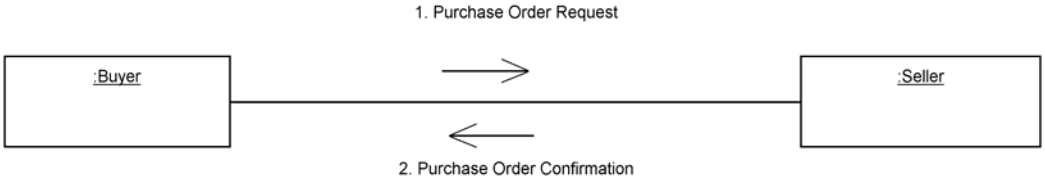


Abbildung 17 Network Component Collaboration

Abb.16) die Interaktion der Komponenten durch entsprechende Business-Action- bzw. Business-Signal-Nachrichten. Geschäftsaktivitäten werden in ihrem Ablauf in der „**Business Transaction Dialog Specification**"(siehe Abb.17) näher beschrieben.

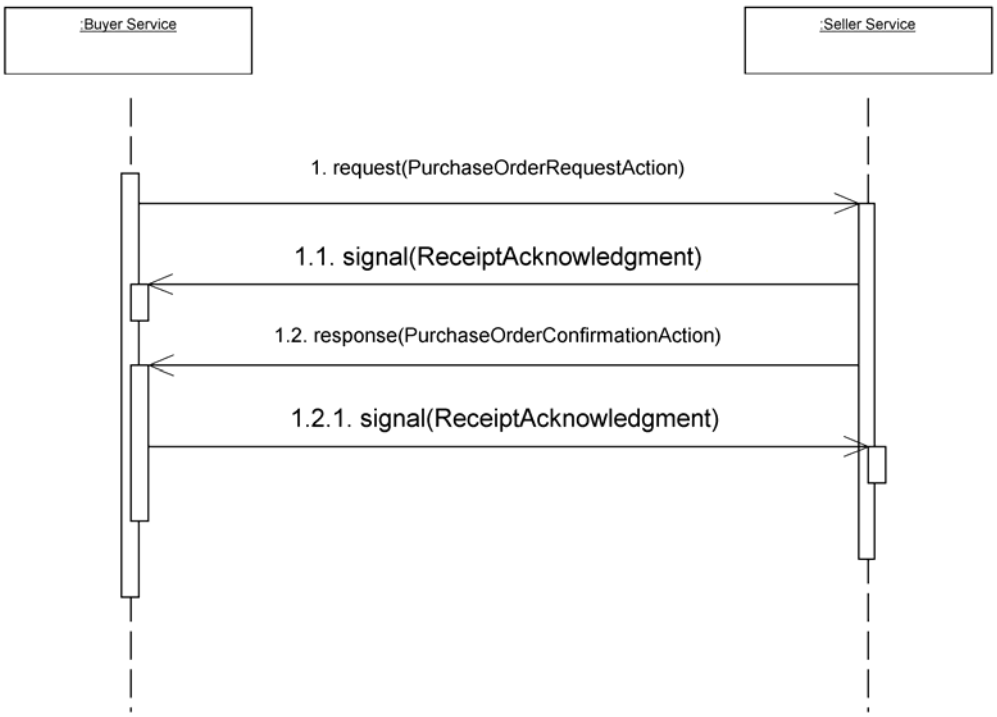


Abbildung 18 Request Purchase Order Dialog

- **Implementation Framework View (IFV)**
  Die Implementation Framework View (auch Service Layer genannt) spezifiziert Nachrichtenformate und Anforderungen an die Kommunikation. Diese Anforderungen schreiben vor, ob ein sicheres Protokoll wie SSL oder digitale Signaturen erforderlich sind. Die Richtlinien für die Nachrichtenformate sind in den Data Type Definitionen und den Message Guidelines zu finden.

RosettaNet stellt auf [http://www.rosettanet.org] die aktuellen PIP Spezifikationen zur Verfügung. Sie sind in sieben Cluster unterteilt, welche die Kernprozesse geschäftlicher Unternehmungen wiederspiegeln.

CL	Titel	Beschreibung
0	RosettaNet Support	Unterstützt administrative Funktionalitäten
1	Partner Product and Service Review	Erlaubt es Information über Dienste und Produkte zu sammeln, zu pflegen und zu verteilen
2	Product Information	Ermöglicht Verteilung und Aktualisierung der Produktinformationen einschließlich Produktänderungen und technische Spezifikationen.
3	Order Management	Unterstützt das gesamte Bestellwesen
4	Inventory Management	Ermöglicht Warenbestandsmanagement
5	Marketing Information Management	Ermöglicht das Mitteilen von Marketinginformationen
6	Service and Support	Realisiert technischen Support und Service
7	Manufacturing	Unterstützt virtuelle Herstellungsprozesse

D PIP und BPEL4WS

[22] Es stellt sich einem die Frage, ob eine Konkurrenz zwischen den verschiedenen neu geschaffenen Spezifikationen und RosettaNet besteht? Die Antwort soll uns ein Ausblick auf den Web – Service – Technology - Stack (siehe Abb.18) liefern. Es wurde bereits erwähnt, dass das PIP - Konzept von RosettaNet lediglich öffentliche Geschäftsprozesse unterstützt, keine Sprachkonzepte zur Implementierung zur Verfügung stellt und seinen technologischen Unterbau aufgrund seiner Historie nicht auf aktuelle Standards wie WSDL und SOAP aufbaut. Allerdings beinhaltet und fördert die Philosophie von RosettaNet die Möglichkeit neue Standards für Geschäftsprozesse zu benutzen. Versucht man nun die beiden Konzepte PIP und BPEL4WS zu vereinigen, erhält man eine mächtige Kombination aus einer wohldefinierten Sprache (BPEL), um elektronische und vor allem verteilte Geschäftsprozesse zu implementieren, und gründlich erforschten, standardisierten und vor allem von der Industrie anerkannten Geschäftsablaufspezifikationen aus dem PIP – Konzept. Um diese Heirat jedoch zu ermöglichen bedarf es noch einiger Transformationen und Zwischenschritte, die im folgenden kurz erläutert werden:

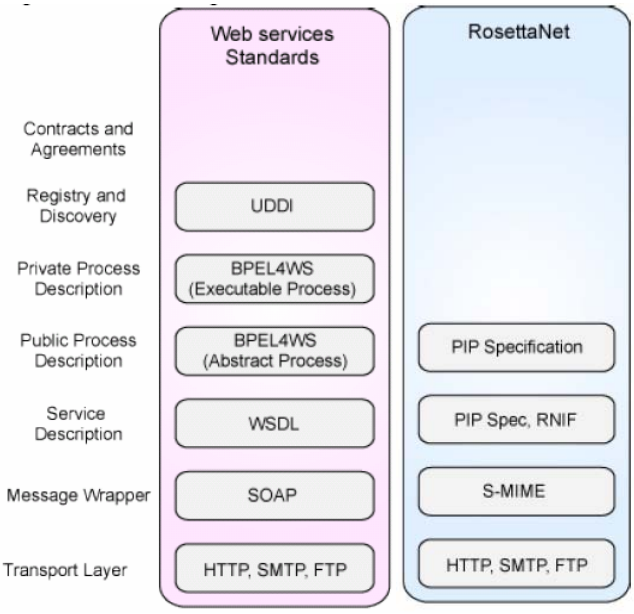


Abbildung 19 RosettaNet vs. BPEL4WS

- Die in den XML-DTDs definierten Nutzdaten (Geschäftsdokumente und Geschäftsbegriffe), welche in den Business-Action-Nachrichten verschickt werden, müssen als Typen in WSDL definiert werden.
- Die Definitionen der RosettaNet–Nachrichten, inklusive der Nachrichtennamen, müssen auf die Nachrichtendefinition von WSDL abgebildet werden.

- Die Geschäftsaktivitäten der RosettaNet PIP müssen in WSDL beschrieben werden.
- Das Rollenkonzept von RosettaNet PIPs muss in das Rollenkonzept von BPEL4WS transformiert werden.
- RosettaNet PIP Choreographien müssen mit „Business Protocols“ und „Executable Business Processes“ aus BPEL4WS implementiert werden.
- Die Fehlerbehandlung von dem RosettaNet RNIF muss der Fehlerbehandlung von BPEL4WS angepasst werden.

Folgende Konzepte können einerseits nicht direkt von BPEL4WS realisiert werden und erfordern somit individuelle Implementierungen oder werden andererseits in RosettaNet nicht zur Verfügung gestellt:

– In RosettaNet können Kontrollparameter (Anzahl der Wiederholungen, Zeitschranken, usw.) für den Nachrichtenaustausch definiert werden, die in BPEL4WS gegebenenfalls außerhalb der Prozessdefinitionen geklärt werden müssen.
– Das RosettaNet RNIF stellt für die Sicherheit und die Vertrauensbeziehung zwischen den Geschäftpartner eigene Nachrichtenumschläge zur Verfügung, während BPEL4WS hier auf weitere Standards (WS-Security) verweist.
\+ In BPEL4WS können mehr als zwei Geschäftspartner in einem Geschäftsprozess interagieren.
\+ In BPEL4WS existierende, strukturierte Kontrollflussaktivitäten (<while>, <switch>, <pick>, usw.) erlauben eine höhere und elegantere Flexibilität in der Orchestrierung von Geschäftsprozessen.

Schlusswort

Diese Ausarbeitung beschrieb durch eine sehr allgemeine Einführung in das Workflow Management und der Dienstorientierung den Aufstieg zur nächsten Technologiestufe der Choreographie von Diensten. Es wurden allgemeine Charakteristika von Choreographiebeschreibungssprachen für Web Services genannt und erklärt. Um einen groben Einblick in die Menge bereits bestehender Spezifikationen zu geben, wurden die Charakteristika auf zwei unterschiedlichen Ansätze von Choreographiespezifikationen, BPEL4WS und YAWL, angewendet. Zusätzlich dazu wurde ein Einblick in das RosettaNet Framework mit seinem PIP - Konzept gegeben, um die Entstehungsgeschichte von konkreten Web-Service-Choreographien zu zeigen.
Abschließend lässt sich sagen, dass der Gedanke, verschiedene Dienste miteinander zu einem neu entstehenden Dienst zu verknüpfen, auf seit Jahren bereits vorhandene Entwicklungen im Workflow Management beruht, und somit keine wirkliche Innovation der heutigen Forschung im Bereich der Web Services darstellt. Allerdings lässt sich dieser Gedanke innovativ in der Standardisierung von Protokollen, Sprachkonzepten, Schnittstellen und Ablaufmustern anwenden. Hierdurch wird es nun möglich eine effektive Choreographie von Diensten basierend auf dem Web Service Technology Stack zu realisieren.
Ein Zukunftsausblick birgt die Hoffnung, dass Standardisierungsgremien wie OASIS und W3C eine effektive „Synchronisation“ der ihnen übergebenen Spezifikationen durchführen, um durch eine einheitliche Schnittstelle das Plug & Play Prinzip in der Choreographie von Diensten zu ermöglichen. Wünschenswert wären auch einfach zu bedienende, GUI - basierte Werkzeuge, mit denen zwischen den bestehenden Diensten eine Choreographie einfach gezeichnet werden kann, um die unübersichtliche Kodierung der XML - Sprachansätze zu vermeiden.

Anhang BPEL4WS

[A] Variablendeklaration:

```
<variables>
      <variable name="ncname" messageType="qname"?
       type=”qname”? element=”qname”?/>+
</variables>
```

[B] Wertzuweisungen:

```
<assign>
      <copy>
            <from variable="c1"/>
            <to variable="c2"/>
      </copy>
</assign>
```

[C] Anlegen eines portTypes:

```
<portType name="ServicePT">
      <operation name="orderRequest">
            <input message="orderRequestMsg"/>
      </operation>
</portType>
```

[D] Anlegen eines partnerLinkType:

```
<plnk:partnerLinkType name="order_digi_cam">
      <plnk:role name="warehouse">
            <plnk:portType name="ServicePT"/>
      </plnk:role>
      <plnk:role name="customer">
            <plnk:portType name="ServiceCustomerPT"/>
      </plnk:role>
</plnk:partnerLinkType>
```

[E] Anlegen eines konkreten partnerLinks:

```
<partnerLinks>
      <partnerLink name="warehouse_partner"
       partnerLinkType="order_digi_cam"
       partnerRole="warehouse"
       myRole="customer"/>
</partnerLinks>
```

[F] Allgemeiner Aufbau eines Scopes:

```
<scope>
      standard-elements
      <variables>?
            ...
      </variables>
      <correlationSets>?
            ...
      </correlationSets>
      <faultHandlers>?
            ...
      </faultHandlers>
      <compensationHandler>?
```

```
            ...
      </compensationHandler>
      <eventHandlers>?
            ...
      </eventHandlers>
      activity
</scope>
```

[G] Correlation Set

```
<correlationSets>
      <correlationSet name="OrderDigiCam" properties="orderID"/>
</correlationSets>
```

[H] Allgemeiner Aufbau eines Excepetion-Handler:

```
<faultHandlers>
      <catch faultName="x:foo">
            <empty/>
      </catch>
      <catch faultVariable="bar">
            <empty/>
      </catch>
      <catch faultName="x:foo" faultVariable="bar">
            <empty/>
      </catch>
      <catchAll>
            <empty/>
      </catchAll>
</faultHandlers>
```

z.B.

```
<faultHandlers>
      <catch faultName="novalidCreditCardNumber">
            <invoke partner="customer" portType="CustomerPT"
             operation="sendRejection" inputContainer="rejection"/>
      </catch>
</faultHandlers>
```

Anhang YAWL

[I] Allgemeiner Aufbau eines Tasks:

```
<task id="register">
      <name>Collect information from customer</name>
      <flowsInto>
            <nextElementRef id="flight"/>
            <predicate>/data/want_flight = 'true'</predicate>
             <isDefaultFlow/>
      </flowsInto>
      <flowsInto>
            <nextElementRef id="hotel"/>
            <predicate>/data/want_hotel = 'true'</predicate>
      </flowsInto>
      <flowsInto>
            <nextElementRef id="car"/>
            <predicate>/data/want_car = 'true'</predicate>
      </flowsInto>
      <join code="and"/>
      <split code="or"/>
      <startingMappings>
            <mapping>
                  <expression query="/data/customer"/>
                  <mapsTo>customer</mapsTo>
            </mapping>
      </startingMappings>
      <completedMappings>
            <mapping>
                  <expression query="/data/customer"/>
                  <mapsTo>customer</mapsTo>
            </mapping>
            <mapping>
                  <expression query="/data/want_flight"/>
                  <mapsTo>want_flight</mapsTo>
            </mapping>
            ...
      </completedMappings>
      <decomposesTo id="register"/>
</task>
```

[J] Lokale Variablen

```
<localVariable name="title">
      <type>xs:datatype</type>
      [<initialValue>value</initialValue>]
</localVariable>
```

[K] Variablendeklaration

```
<localVariable  name="customer">
<type>xs:string</type>
      <initialValue> Type name of customer </initialValue>
</localVariable>
<localVariable name= "payment_account_number">
      <type>xs:string</type>
</localVariable>
...
<localVariable name= "want_flight">
```

```
      <type>xs:boolean</type>
</localVariable>
<localVariable name= "want_hotel">
      <type>xs:boolean</type>
</localVariable>
<localVariable name= "want_car">
      <type>xs:boolean</type>
</localVariable>
<localVariable name="flightDetails">
      <type>xs:string</type>
</localVariable>
...
```

[L] Parameterübergabe

```
<decomposition id="register" xsi:type="Web ServiceBrokerFactsType">
      <inputParam name="customer">
            <type>xs:string</type>
      </inputParam>
      <outputExpression query="/data/customer"/>
      <outputExpression query="/data/start_date"/>
      ...
      <outputExpression query="/data/want_flight"/>
      ...
      <outputParam name="customer">
            <type>xs:string</type>
      </outputParam>
      <outputParam name="start_date">
            <type>xs:dateTime</type>
      </outputParam>
      ...
      <outputParam name="want_flight">
            <type>xs:boolean</type>
      </outputParam>
      ...
</decomposition>
```

Literaturverzeichnis

- Tobias Hauser, Ulrich M.Löwer; Web Services - Die Standards; Galileo Computing 1.Auflage 2004
- Setrag Khohafian; Web Services and Virtual Enterprises; Tect; http://www.Web Servicearchitect.com
- Workflow and Internet: Catalysts for Radical Change; http://www.wfmc.org; WfMC White Paper; 1998
- A. Martens; Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services. Dissertation, Institut für Informatik, Humboldt-Universität zu Berlin, 2003. Erschienen in WiKu: Stuttgart, Berlin & Paris.
- Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju; Web Services – Concepts, Architectures and Applications; 2004; Springer
- Frank Leymann; Web Services: Distributed Applications without Limits; BTW 2003
- Michael P.Papazoglou; Web Services and Business Transactions; ACM; 2003
- Business Process Execution Language for Web Services; Version 1.1; 2003; http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
- Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede; Pattern Based Analysis of BPEL4WS; Technical Report FIT-TR-2002-04, QUT
- Michael zur Muehlen, Jeffrey V. Nickerson, Keith D. Swenson; Developing Web Services Choreography Standards – The Case of REST vs. SOAP; erschienen in Decision Support Systems 36
- A.Kemper, A.Eickler; Datenbanksysteme; Oldenbourg 1999; 3.Auflage
- WS-Coordination - Specification; http://www-106.ibm.com/developerworks/Web Services/library/ws-coor/; BEA, IBM, Microsoft, 2002
- WS-Transaction – Specification; http://www-106.ibm.com/developerworks/Web Services/library/ws-transpec/; BEA, IBM, Microsoft, 2002
- W.M.P. van der Aalst; Design and implementation of the YAWL system; 2003
- W.M.P. van der Aalst; Yet Another Workflow Language; FIT-TR-2003-04; 2003
- PIP 3A4: Request Purchase Order; http://www.rosettanet.org/pipdirectory/
- RosettaNet based Web services: BPEL4WS and RosettaNet; Part 1; http://www-106.ibm.com/developerworks/webservices/library/ws-rose1/
- RosettaNet based Web services: BPEL4WS and RosettaNet; Part 2; http://www-106.ibm.com/developerworks/webservices/library/ws-rose2/
- RosettaNet based Web services: BPEL4WS and RosettaNet Part; 3; http://www-106.ibm.com/developerworks/webservices/library/ws-rose3/
- Simone Schlachter, Beatrice Huber; RosettaNet - Eine Einführung; Studienarbeit
- n.ethz.ch/student/jodaniel/37-310/ reports/rosettanet_Schlachter&Huber.pdf; ETH Zürich; 2003

Quellenverzeichnis

[1] Tobias Hauser, Ulrich M.Löwer; Web Services - Die Standards; Galileo Computing 1.Auflage 2004; S.162
[2] Setrag Khohafian; Web Services and Virtual Enterprises; Tect; http://www.Web Servicearchitect.com; S.3
[3] Setrag Khohafian; Web Services and Virtual Enterprises; Tect; http://www.Web Servicearchitect.com; S.3
[4] Workflow and Internet: Catalysts for Radical Change; http://www.wfmc.org; WfMC White Paper; 1998
[5] A. Martens; Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services. Dissertation, Institut für Informatik, Humboldt-Universität zu Berlin, 2003. Erschienen in WiKu: Stuttgart, Berlin & Paris. S. 16
[6] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju; Web Services – Concepts, Architectures and Applications; Springer; 2004; S. 141
[7] Setrag Khohafian;Web Services and Virtual Enterprises; Tect; http://www.Web Servicearchitect.com; S.5
[8] Axel Martens; Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services; Humboldt Universität zu Berlin. S. 33
[9] Frank Leymann; Web Services: Distributed Applications without Limits; BTW 2003; S.1
[10] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju;Web Services – Concepts, Architectures and Applications; Springer; 2004; S. 256f
[11] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju;Web Services – Concepts, Architectures and Applications; Springer; 2004; S. 261f
[12] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju;Web Services – Concepts, Architectures and Applications; Springer; 2004; S. 264ff
[13] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju;Web Services – Concepts, Architectures and Applications; Springer; 2004; S. 269f
[14] Michael P.Papazoglou;Web Services and Business Transactions; ACM; 2003; S.1.
[15] A.Kemper, A.Eickler; Datenbanksysteme; Oldenbourg; 3.Auflage;1999; S.245f
[16] Michael zur Muehlen, Jeffrey V. Nickerson, Keith D. Swenson; Developing Web Services Choreography Standards – The Case of REST vs. SOAP; Decision Support Systems 36; 2004; S.6
[17] Business Process Execution Language for Web Services Version 1.1; http://www-106.ibm.com/developerworks/library/ws-bpel/, IBM, 2003
[18] W.M.P. van der Aalst; Design and implementation of the YAWL system; The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia, June 2004
[19] W.M.P. van der Aalst; YAWL: Yet Another Workflow Language;S.14, Technical report, FIT-TR-2003-04; 2003
[20] W.M.P. van der Aalst; Workflow Patterns; http://tmitwww.tm.tue.nl/research/patterns/; Technical Report FIT-TR-2002-04, QUT
[21] Simone Schlachter, Beatrice Huber; RosettaNet - Eine Einführung; Studienarbeit n.ethz.ch/student/jodaniel/37-310/ reports/rosettanet_Schlachter&Huber.pdf; ETH Zürich; 2003
[22] RosettaNet based Web services: BPEL4WS and RosettaNet; Part 1-3; http://www-106.ibm.com/developerworks/webservices/library/ws-rose1|2|3