Sprachen II:
XL vs. BPEL4WS

Technische Universität München

Lehr- und Forschungseinheit III
Prof. R. Bayer, Prof. D. Kossmann

Hauptseminar Informatik im Sommersemester 2003

Web-Services

Bearbeiter: Sergei Chevtsov

Betreuer: Andreas Grünhagen

Vortragsdatum: 26.06.2003

Inhaltsverzeichnis

1. Einführung

1.1. Was sind Web Services?

Nicht-technisch gesprochen verbirgt sich hinter dem Begriff Web-Service eine Reihe internationaler, internet-basierter Standards, die es von einander unabhängigen Applikationen erlauben, Dienste auszutauschen, ohne dass Menschen in die Kommunikation eingreifen oder Schnittstellen anpassen müssen. Das gilt für Anwendungen innerhalb ein und desselben Unternehmens genauso wie für Applikationen unterschiedlicher Firmen. Ein entscheidender Aspekt dabei ist, dass diese neuen Standards und Protokolle den Leistungsaustausch unabhängig von verwendeten Rechnern, Endgeräten, Betriebssystemen, Softwareprodukten und Programmiersprachen erlauben sollen. Darüber hinaus beschreiben die Standards auch einen Weg, über den Anwendungen quasi automatisch die jeweils richtige Applikation für bestimmte Leistungsbedarfe auffinden und den Leistungsaustausch vornehmen können. In der Endvision wird das Internet damit zu einem naht- und grenzenlosen Meer an Applikationen, die „real-time“ untereinander Daten und Leistungen austauschen.

Web-Services werden in zwei Kontexten intensiv diskutiert. Einerseits im Bereich der „***Enterprise Application Integration***“. Hier liegt das Ziel in der Reduktion der Integrationskosten durch eine vergleichsweise einfache und flexible Kopplung unternehmensinterner Applikationen. Der Reiz der Web-Services besteht dabei insbesondere in der Plattformunabhängigkeit der Standardisierung. Leiden doch so gut wie alle großen IT- Abteilungen unter der Heterogenität ihrer gewachsenen Legacy-Systeme.
Der ursprüngliche Verwendungszweck der Web-Services aber ist die effiziente Gestaltung von Wertschöpfungsketten im ***E-Business***. Ob im „Supply Chain Management“ oder beim Aufbau elektronischer Marktplätze, die Vernetzung über Unternehmensgrenzen hinweg zwecks Schaffung unternehmensübergreifender Applikationen ist das anfängliche Ziel der Gestaltung der Standards rund um Web-Services.

Beiden Anwendungsoptionen liegt die Vorstellung der uneingeschränkten Interoperabilität von Applikationen zugrunde. Viele Gremien und ungezählte Standards haben in der Vergangenheit dieses Ziel verfolgt. Web-Services führen insofern eine Tradition fort. Letzter großer Hoffnungsträger im Zeichen der Interoperabilität war die 1989 gegründete „Object Management Group“ mit ihrer *Common Object Request Broker Architecture*, bekannt unter dem Akronym CORBA. Davor war es die *Distributed Computing Environment* der “Open Group”.
Architekturen und Erfahrungen dieser Standardisierungsbemühungen haben durchaus die Gestaltung von Web-Services beeinflusst und sind zum Teil auch Grundlage der neuen Standards.

Im Gegensatz zu den Vorläufern folgen Web-Services zwei neuen grundsätzlichen Maximen: dem Konzept der explizit losen Kopplung und einem vollständig dezentralen Ansatz. Beides fördert die Flexibilität der Kooperation der Applikationen untereinander und reduziert damit den Aufwand zur gegenseitigen Anpassung.

Falls Interesse an technisch detaillierter Darstellung rundum den Begriff „Web Service“ besteht, verweise ich auf vorherige Ausarbeitungen.

1.2. Probleme bei der Implementierung von Web Services

Die Nutzung der klassischen Programmiersprachen wie Java oder Visual Basic, sowie eines SQL- basierten relationalen Datenbankmanagementsystems (z.B. DB2) als Grundlage für Web Services bringt große Nachteile mit sich.
Das Hauptmanko dieser Werkzeuge ist, dass durch die Mischung der durchaus unterschiedlichen Paradigmen eine hohe Anzahl an irrelevanten, aber teuren und fehlerbehafteten Datentransformationen erforderlich wird.

Mit diesen Technologien aufgebaute XML- basierte Webanwendungen werden folgende Schwierigkeiten haben:

a) **XML - Java „mismatch“:** XML Daten müssen erst zu Java Objekten (oder zu internen Repräsentationen einer anderen ähnlichen Sprache) umgewandelt werden, bevor ein Java- Programm sie verarbeiten kann. Genauso müssen die Java Objekte zurück zu XML Daten am Ende der Verarbeitung konvertiert werden.

b) **Java- Datenbank „mismatch“:** Java Objekte müssen durch eine JDBC-ähnliche Schnittstelle hin- und hergereicht werden, um auf relationale Datenbanken zugreifen zu können. Diese berüchtigte „*database impedance mismatch*“ löste die Entwicklung objekt- orientierter Datenbanken aus.

Sprachen- und Datenbankentwickler setzten sich verstärkt dafür ein, ihre Produkte mit XML- Erweiterungen anzureichern und diese Arbeiten, mit denen sich der Programmierer momentan von Hand beschäftigen muss, automatisch durchführen zu lassen.
Tatsächlich gibt es bedeutende Anstrengungen seitens Java- und Datenbankentwicklern sowie Komplettlösungsanbietern in diese Richtungen. Doch empfindet eine nicht kleine Anzahl an Computertheoretikern intuitiv, dass die Typensysteme von XML, Java und relationalen Datenbanken einfach zu unterschiedlich sind- und damit inkompatibel, um auf effizientem Weg riesige skalierbare Anwendungen herzustellen, die diese drei Paradigmen umspannen,

Zusätzlich zu diesen doppelten „*mismatches*“ begegnet den Programmierern ein anderes Problem, das in drastischer Weise auf sowohl Produktivität als auch Performance der Web Services Einfluss nimmt. Es tritt häufig das Problem auf, dass in eine imperative Programmiersprache (wie z.B. Java) sehr unterschiedliche Semantiken beigemischt werden. So werden oft low-level Protokolle und Leistungsverbesserungen mit Datenvalidierung und echter Applikationslogik vermischt.
In dem Fall, wo jene komplizierten semantischen Aktionen stark überlagert werden, ist eine Weiterentwicklung der Anwendung fast unmöglich.
Man stelle sich nur vor, was eine Änderung eines Datenmodells kostet, wenn die Daten drei Paradigmen umspannen, Dateninstanzen auf drei verschiedenen

Schichten präsent sind und potentiell in verschieden Orten einer Plattform im Cache, oder auch dauerhaft, gespeichert werden.

Ein weiterer Nachteil von Java ist, dass die Sprache sich nicht immer dafür eignet, mit Programmausfall- bzw. der Dienstqualitäts- („quality of service“) Anforderungen in einem netzwerk- zentriertem Umfeld umzugehen.

Letztendlich sind viele, in den Web Services gebrauchte Funktionen- wie Logging, Datenbankzugriff, Sicherheit, Transaktionsunterstützung etc.- nicht im *Java Development Ki*t („JDK“) enthalten. Zwar bietet J2EE (s. z.B. die nächste Ausarbeitung) diese Funktionalitäten an, aber damit verkompliziert sich auch das Programmiermodell- ein Grund, warum J2EE nicht so beliebt ist.

Das sind die wichtigsten Gründe, warum man sehr stark an der Entwicklung einer speziell für Web Services geeigneten Programmiersprache forscht. Natürlich spielen auch Ehrgeiz und Prestige große Rollen.

2 Technische Grundlagen

Eine wichtige Voraussetzung für eine Programmiersprache für Web Services ist die Kompatibilität mit den existierenden Standards in diesem Bereich. Im folgenden Abschnitt wird im einzelnen der gegenwärtigen Stand der Entwicklung beschrieben, wobei bei größerem Interesse die in meinem Literaturverzeichnis erwähnten Quellen oder *vorhergegangene Ausarbeitungen* hinzugezogen werden sollten.

2.1. Das abstrakte Datenmodell von XML

Der Zweck eines abstrakten Datenmodells ist schon in den 70er Jahren in vielen theoretischen Arbeiten festgehalten worden. Die Abstraktion ermöglicht den Programmen eine Unabhängigkeit zwischen den logischen und den physischen Daten. Somit brauchen die Programmierer, soweit sie das Modell anwenden dürfen, sich nur um abstrakte Repräsentation der Daten kümmern; die Realität der physischen Datenrepräsentation (z.B. „byte order“) braucht nicht beachtet zu werden. Außer dem offensichtlich reduzierten Aufwand für die Entwickler ergibt sich ein weiterer, vielleicht der wichtigste Vorteil: Bei Veränderungen der physischen Datenrepräsentation(z.B. Wechseln der Plattform) brauchen die Anwendungen nicht neu geschrieben werden. Dieses Konzept wurde in den letzten 30 Jahren in der relationalen Datenbankwelt erfolgreich angewendet.

Auch bei der Entwicklung des XML Standards wurde auf das wichtige Datenbankerbe nicht verzichtet. Das sogenannte „XML abstract data model“ für XML-Anwendungen dient dem gleichen Zweck wie das relationale Datenmodell für relationale Datenbanken. Das Modell wurde erfolgreich bei den Beschreibungen der Semantik von XSLT und XQuery eingesetzt.
Das Datenmodell von XML beschreibt die in der XML- Anwendung vorhandenen Entitäten und die Beziehung zwischen diesen. Die Entitäten beschreiben die Daten (Knoten, Werte, Sequenzen), die durch sehr allgemeine mathematische Strukturen (wie „geordneter Baum von Knoten“) modelliert werden. Die (inneren) Knoten können

eindeutig identifiziert werden und sind vom bestimmten, sogenannten *komplexen* Typ wie *document*, *element*, *attribute, comment, namespace* etc.
In den Blättern der Bäume werden Werte gespeichert, die von einem sogenannten *Basis*typ sein können (*integer, decimal, string* etc.). Einen wichtigen Begriff des XML Datenmodells stellt die Sequenz dar. Eine Sequenz wird stets als flach vorausgesetzt, d.h. z.B. dass die Sequenzen von Sequenzen automatisch flach gedrückt werden.
Schließlich setzt das Datenmodell verbindlich die topologische Reihenfolge der Knoten im Baum fest (entspricht dem „*depth-first- traversing*“), die von einem Programm angefragt werden kann.

2.2. XML Schema und XML- Typen Schema

Das Datenmodell beschreibt nur die Grundbestandteile eines geordneten Baums. Das XML Schema dagegen kümmert sich um strukturelle und inhaltliche Informationen der geordneten Bäume. So definiert das XML Schema die sogenannten *einfachen Typen* (mit dem dazugehörigen Wertebereich und den Grundoperationen), die vom XML Datenmodell unterstützt werden (dort dürfen die Knoten vom einfachen Typ ausschließlich Blätter sein). Weiterhin ermöglicht es die Definition der *komplexen Typen* durch den Nutzer und stellt grundlegende Unterstützung für nutzerspezifische Integritätsbedingungen (z.B. lexikalische Einschränkungen) bereit.

Der Kern des XML Schemas ist das XML- Typen Schema. Durch die Definition eines Typensystems werden drei Funktionalitäten erreicht:

a) **Typüberprüfung:** Bei einem gegebenen Ausdruck und dem gegebenen Typ der Eingabedaten ist es möglich statisch festzustellen, ob der Ausdruck Fehler bei allen (oder nur einigen) gültigen Instanzen des Eingabetyps liefert.
b) **Typbestimmung:** Bei gleichen Voraussetzungen wie oben ist es für das System möglich, den Ergebnistyp eines Ausdrucks zu bestimmen, ohne Berechnungen mit konkreten Instanzen durchzuführen.
c) **Testen der Typannahme:** Wie oben ist es fürs System möglich zu entscheiden, ob das Ergebnis des Ausdrucks eine gültige Instanz eines vordefinierten, erwarteten Ausgabedatentyps ist.

2.3. XML Ausdrücke, XQuery und XPath

Ein komplementärer Standard behandelt XML- Ausdrücke und XML Anfragen.
XQuery ist eine funktionale Sprache. Wie bei allen funktionalen Sprachen bestehen die XQuery- Ausdrücke aus Anwendungen der Funktionen der ersten und zweiten Ordnungen auf Variablen und Konstanten. Funktionen der ersten Ordnung sind logische, arithmetische oder *string*- Manipulationen; Mengenoperationen wie Vereinigung oder Durchschnitt etc.
Funktionen der zweiten Ordnung sind sort, map etc.

Besonders wichtig sind in XQuery die sogenannten FLWR- Ausdrücke der zweiten Ordnung. Das sind XML- Ausdrücke, die nach dem SQL- „SELECT WHERE FROM“-

Muster aufgebaut sind. Wie SQL- Anfragen besitzen die FLWR- Ausdrücke einen speziellen Konstrukt zur Definition von Variablen und deren Wertebereichen (der FOR- Konstrukt in XQuery entspricht dem FROM- Konstrukt in SQL); einen speziellen Konstrukt, der gemäß Prädikaten die Werte der Variablen herausfiltert (in beiden Sprachen der WHERE- Konstrukt) und einen speziellen Konstrukt, der den Aufbau des Ergebnisses beschreibt (der RETURN- Konstrukt in XQuery entspricht dem SELECT- Konstrukt in SQL). Weitere spezielle Ausrücke, sogenannte Pfadausdrücke, werden für die Navigation in einem XML- Baum gebraucht: Deren Syntax und Semantik werden im *XPath* Standard definiert.

XML- Anfragen sind deklarative, seiteneffektfreie Programme, die auf XML Daten angewendet werden. Eine Anfrage besteht aus zwei Teilen.
Der erste Teil enthält die Definitionen von Funktionen und die Deklarationen von lokalen Typen, von Funktionen und XML Schema Importen. Im Hauptteil wird ein Ausdruck definiert, der ausgewertet und als Ergebnis nach dem Ausführen des Programms zurückgeliefert werden soll.

Leider kann die Logik der komplexen Web Services nicht nur mit Hilfe von deklarativen Programmen beschrieben werden. Auch die ausschließliche Verwendung der seiteneffektfreien XML Anfragen würde die Funktionalität stark einschränken.

2.4. XML Protokoll (SOAP)

Ein wichtiger Standard im Gebiet der Web Services ist das XML Protokoll. Der anfängliche Blickpunkt bei der Standardisierung des XML Protokolls (SOAP) richtet sich auf die Erschaffung eines einfachen, XML- basierten Nachrichtenprotokolls, das allgegenwärtig verwendet und leicht mit Hilfe von Skriptsprachen, XML und anderen Werkzeugen programmiert werden kann.
Das Ziel dabei ist die Errichtung eines durch die Webeigenschaften festgelegten Schichtensystems. Dieses Schichtensystem soll direkt auf die Bedürfnisse der Anwendungen eingehen, indem es mit einer einfachen Schnittstelle (z.B. getStockValue oder setCardNumber) ausgestattet ist und inkrementell erweitert werden kann. Der Zweck solch eines Schichtensystems ist der Wunsch, für komplexere Anwendungsschnittstellen Sicherheit, Skalierbarkeit und Robustheit zu garantieren.

Das XML Protokoll besteht aus den Komponenten a) bis d):

a) Ein Umschlag zum Einkapseln der XML- Daten
b) Eine Vereinbarung zum Inhalt des Umschlages, wenn die Daten von RPC (*Remote Procedure Call*)- Anwendungen verwendet werden
c) Ein auf Datentypen des XML Schemas basierender Mechanismus zur Serialisierung derjenigen Daten, die Instanzen eines *nicht*-syntaktischen Datenmodells sind (Objektgraphen etc.)
d) Ein Mechanismus zum Benutzen des HTTP- Transportes im Kontext des XML Protokolls

2.5. Web Services Definition Language (WSDL)

Schließlich ist eine der wichtigsten Voraussetzungen zum Entwickeln der Web Services die Fähigkeit deren Schnittstelle zu beschreiben, über welche die Anwendungen (Web Services- Agenten oder andere Web Services) miteinander kommunizieren. Der Standard hier heißt WSDL.

WSDL besteht aus folgenden Komponenten:

a) **Nachricht**: eine Definition der zum Austausch verwendeten Datentypen und -strukturen
b) Beschreibung der Art und Weise des **Nachrichtenaustauschs:** die Beschreibung der Folgen von Operationen, die vom Web Service unterstützt werden
c) **Protokolleinbindung:** Ein Mechanismus zum Einbinden eines vom Web Service benutzen Protokolls- unabhängig von dessen Nachrichten und der Art und Weise des Nachrichtenaustauschs.

Die ganzen oben beschriebenen Standarte (das abstrakte Datenmodell von XML, das XML Schema und Typenschema, die XML Ausdrücke, das Nachrichtenprotokoll der Web Services und die abstrakte Schnittstelle der Web Services) bilden eine exzellente Grundlage zum Erstellen einer Web Service Infrastruktur.

Was im Moment fehlt ist ein Werkzeug zur tatsächlichen Programmierung der Web Services. Da dafür, wie am Anfang beschrieben, die klassischen Programmiersprachen oder -Plattformen wenig geeignet sind, wird nun anhand von zwei maßgeschneiderten Beispielen die Möglichkeit der konkreten Umsetzung der Web Service Ideen präsentiert: XL von der TU München und BPEL4WS, von IBM (mit Einflüssen von Microsoft).

3. XL

3.1. Programmiermodell

In diesem Abschnitt wird beschrieben, wie Web Services mit Hilfe von XL definiert werden können. Hier finden sich zuerst die grundlegenden Designprinzipien der vorgeschlagenen Programmierschnittstelle. Dann wird erläutert, was man sich unter Web Services aus der Sicht von XL vorstellen kann und wie Web Services miteinander über sogenannte *conversations*[1] kommunizieren. Zum Schluss wird die Syntax der wichtigsten Programmkonstrukte vorgestellt.

3.1.1. Designprinzipien

[1] Dieser Begriff wird später erläutert, im Moment stellen Sie sich darunter einfach irgendeinen Kommunikationsmechanismus vor

Eine Liste vom XL- Entwicklerteam mit den 17 Forderungen an eine Programmiersprache, welche die Definition und Komposition von Web Services unterstützt:

1. **Kompatibilität mit den W3C Standards.** XL muss vollkommen kompatibel zu den XML W3C Standarts sein (u.a. XML Schema, XQuery, XPath, XSLT oder SOAP).
2. **Geschäftsprozesse, Web *conversations*.** Die Sprache muss Konstrukte zum Implementieren der Geschäftsprozesse bereitstellen, und sie muss- allgemeiner gesehen- *conversations* zwischen zwei oder mehreren Web Services unterstützen.
3. **Komposition des Service.** XL soll es ermöglichen, einen high-level Service durch eine Komposition von mehreren Grundservices zu erstellen. Es sollte außerdem möglich sein, neue Services aus den *nicht in XL* geschriebenen Services zu erstellen. So sollte z.B. ein in Java geschriebener Service auf transparente und nahtlose Art integriert werden können. Dabei sollten die Services, die an einer *conversation* teilnehmen, lose gekoppelt sein, so dass Veränderungen in der Implementierung eines Service keine Auswirkung auf andere Services haben.
4. **Nachrichten-basiertes Programmieren.** Ein in XL implementierter Web Service soll mit anderen Web Services über Nachrichten kommunizieren. Services werden über Nachgerichten aufgerufen, und Ergebnisse werden auch über Nachrichten zurückgegeben.
5. **Ortsunabhängiger Aufruf**. Die Web Services sollen eindeutig durch eine URI identifiziert werden. Der Aufruf eines Web Service sollte die entsprechende URI nutzen und vom tatsächlichen Speicherort bzw. dem Ort der Ausführung des Codes unabhängig sein.
6. **Plattformunabhängigkeit, Code- Mobilität.** Die Sprache muss plattformunabhängig sein. Es muss möglich sein, Programme auf jedem Rechner, der einen Interpreter für die Sprache besitzt, laufen zu lassen- unabhängig vom benutzen Betriebs- oder Datenbanksystem.
7. **XML- basiertes Datenmodell und Typsystem.** Die Daten sollen vom standardisierten XML- Datenmodell und Typsystem modelliert werden. Keine anderen Datenmodelle und Typsysteme sollen erlaubt sein.
8. **Optional starke Typisierung.** Die Typen für Datenkomponenten (z.B. Variablen) sind optional. Wenn jedoch eine Variable von einem bestimmten Typ ist, muss starke Typisierung erzwungen werden. Spezielle Konstrukte müssen bereitgestellt werden, so dass der Programmierer Eigenschaften der Komponenten dynamisch erzwingen kann, wenn statisch kein spezieller Typ mit einer Komponente assoziiert wird.
9. **Unabhängigkeit zwischen logischer und physischer Datenrepräsentationen.** Den Programmierern soll nur die logische Struktur der XML Daten bewusst sein; sie brauchen keine Kenntnisse über die physikalische Repräsentation der Daten (z.B. DOM Bäume, SAX Ereignisse, XML Dateien etc.)
10. **"High-level"- Programmierung.** XL muss *high level* sein und überall, wo es möglich ist, deklarative Konstrukte verwenden. Die Sprache muss außerdem die *high-level* Ausnahmebehandlung und andere spezielle Konstrukte unterstützen, um eine einfache Implementierung der komplexeren Services wie Logging, zeitabhängige Aktionen etc. zu ermöglichen.

11. **Imperatives Programmieren.** Die Sprache muss die imperativen Standardkonstrukte wie Sequentialisierung und Iteration bereitstellen.
12. **Transaktionen.** Die Sprache muss Konstrukte bereitstellen, die den Programmierern ermöglichen, Reihenfolgen von Aktionen, die nach dem ACID- Paradigma ablaufen sollen, zu spezifizieren.
13. **Universale Namensgebung für jede Komponente.** Jede Komponenten (Programm, *conversation*, Nachricht, Transaktion, Ausnahme) muss eine URI haben, um die Herkunft der Daten feststellen zu können bzw. Informationsverfolgung zu ermöglichen.
14. **Authentifizierung, Autorisierung und Sicherheit.** Es muss möglich sein, eine uneingeschränkte Zugriffskontrolle und damit eine Nutzungseinschränkung des Service zu implementieren.
15. **Automatische Optimierung.** Der Sprachdesign soll automatische Laufzeitoptimierungen ermöglichen und fördern bzw. soll von *low-level* (fest kodierten) Optimierungen abhalten oder sie gar nicht zulassen.
16. **Protokolltransparenz.** Zugriffe auf eine Datenbank und Aufrufe von entfernten („*remote*“) Web Services müssen transparent sein. Die benutzten Protokolle (z.B. JDBC oder http) müssen in der Implementierung der Sprache versteckt werden.
17. **Eleganz und Einfachheit.** Die Semantik der Sprache soll sauber und intuitiv sein.

3.2. Web Services in XL

Aus der Perspektive des XL- Entwicklerteams ist ein Web Service jeder eigenständige Software-Bestandteil, der von einer einzigartigen URI identifiziert wird, SOAP Nachrichten versteht und dessen Handlungen von WSDL beschrieben werden können.

Eine solche Beschreibung eines Web Service ist sehr allgemein und lässt auf kein besonderes Programmiermodell schließen.
Mit anderen Worten könnte ein Web Service auch in Java oder jeder anderen Sprache implementiert werden. Auch Web Services, die in verschiedenen Sprachen geschrieben wurden, könnten im Prinzip unter Benutzung von SOAP und unabhängig von der Weise, wie sie definiert wurden, kommunizieren.

Jedes XL Programm wird den beschriebenen Forderungen von seiner Natur aus gerecht. Es nimmt vom Anwendungsprogrammierer außerdem die Last, sich um Standards wie URI, SOAP, oder WSDL kümmern zu müssen.

Technisch gesprochen versteht XL einen Web Service als eine allgemeinere XQuery-Entität („*entity*“). Folgende vier optionale Teile können bei der Implementierung festgelegt werden:

- **Web Service Definitionen:** Wie in XQuery können lokale Funktionen und Typen im Rahmen einer Web Service Definition definiert werden. Desweiteren können Schemata und Namespaces importiert werden. Die Syntax und Semantik sind die gleichen wie bei XQuery.

- **Variablen- Deklarationen:** Dieser Teil deklariert die internen Daten (Zustände) des Web Services. In anderen Worten: Dieser Teil spezifiziert die *globalen Variablen* eines Web Services.
  XL unterstützt zwei Arten von globalen Variablen: die „*Web Service- Instanz*"- Variablen und „*conversation- Instanz*"- Variablen. Beide Arten von Variablen haben einen globalen Gültigkeitsbereich.
  „Web Service- Instanz"- Variablen haben eine einzige Instanz, die für den ganzen Web Service einmalig ist. Ein Beispiel für diese Art von Variablen ist die Kunden-Datenbank eines Online-Shops, auf die über alle *conversations* hinaus zugegriffen werden muss.
  „*conversation*- Instanz"-Variablen werden in den *conversations*, an denen der Web Service teilnimmt, unterschiedlich instanziiert. Diese Variablen repräsentieren den Kontext einer *conversation*. Ihre Instanzen werden jedes Mal, wenn ein Web Service eine neue *conversation* mit anderen Web Services startet, gebildet. In einem Online- Shop würde jeder Verkaufsprozess von einer *conversation* repräsentiert werden. Eine Variable in diesem Zusammenhang könnte den spezifischen Kunden oder die gültigen Zahlungsbedingungen speichern.
  Zusätzlich unterstützt XL die lokalen Variablen, die in Operationen benutzt werden können; diese Variablen werden jedoch nicht global deklariert. Solche (traditionell) lokalen Variablen werden für jeden Block- /Aktionsaufruf instanziiert und zerstört („destroyed"), wenn die Ausführung von dem Block/der Aktion beendet ist.
  Werte aller Variablen in XL sind XML Werte, d.h. Instanzen des XML-Datenmodells. Mit anderen Worten kann der Wert einer XL Variable ein XML-Dokument oder der Inhalt einer SOAP Nachricht sein. Der Typ eines XL-Wertes darf (muss aber nicht) mittels XML Schema eingeschränkt werden. Jedoch ist es auch möglich, mit *nicht-typisierten* XML Daten oder mit XML Daten, für die der Typ a priori nicht bekannt ist, zu arbeiten.

- **Deklarative Web Service Konstrukte:** Dieser Teil enthält *Invarianten* (Integritätseinschränkungen) und andere deklarative *high-level* Anweisungen, die das Laufzeit-Verhalten von Web Service überwachen, wie z.B. deklarative Fehlerbehandlungsmechanismen, deklarative Zugriffskontrolle, Default-Aktionen, periodische Aktionen und deklarative „*conversation*"- Schemata. Invarianten werden typischerweise Werte der globalen Variablen einschränken- sie sind insofern ähnlich den Integritäts- *constraints* in den Datenbanken. Invarianten können aber auch benutzt werden, um bestimmte *Policies* zu implementieren; z.B. was die Sicherheit angeht. So kann eine Invariante ausdrücken, dass es einem Web Service nicht erlaubt ist, im Rahmen ein und der gleichen c*onversation* mit zwei unterschiedlichen Banken zu interagieren.

- **Operationsspezifikationen:** Dieser Teil beschreibt die möglichen *Aktionen* (oder Operationen), die vom Web Service unterstützt werden. Die Operationen eines Web Service entsprechen in objektorientierter Terminologie den Methoden einer Klasse. Die Anweisungen, die benutzt werden können, um den Rumpf der Operationen zu definieren sind weiter unten zusammengefasst.

3.3 Conversations

Wie früher erwähnt kommunizieren die Web Services mit den Mitteln der Nachrichten. Eine *conversation* wird definiert als eine Menge von gegenseitig in Beziehungen stehenden Nachrichten, die zum Erreichen eines Ziels (z.B. Informationsaustausch) zwischen zwei oder mehreren Teilnehmern ausgetauscht werden. *Conversations* werden einmalig von URIs identifiziert. Sie werden implementiert, indem im Umschlag einer jeden ausgetauschten Nachricht die *conversation* URI mittransportiert wird.

Ein typisches Beispiel für eine *conversation* ist eine Kundensitzung in einem Online-Geschäft. Zuerst logt sich der Kunde ins System ein (die erste Nachricht vom Kunden ans System plus die Antwort vom System). Danach kauft der Kunde etwas (die zweite Nachricht und Antwort), und schließlich bestimmt der Kunde eine Zahlungsmethode (die dritte Nachricht und Antwort). Natürlich kann die dritte Nachricht nur im Zusammenhang mit den ersten beiden Nachricht verstanden werden.
Ein anderes Beispiel für *conversations* ist eine Online-Auktion. Die Auktions-Webseite informiert ihre Kunden über ein neues Produkt, das verkauft wird; die Kunden antworten durchs Senden der Gebote; die Auktions- Webseite informiert wiederum die Kunden über neue Gebote und den Stand der Auktion.
Wieder kann ein Gebot nur im Zusammenhang einer ganzen Auktion verstanden werden, und die im Rahmen einer speziellen Auktion ausgetauschten Nachrichten stehen in wechselseitiger Beziehung.
Das Auktions-Beispiel zeigt, dass mehr als zwei Web Services in eine *conversation* verwickelt werden können, und dass Interaktions- Schemata ziemlich komplex sein können. Offensichtlich ist es nicht allen Teilnehmern einer *conversation* gestattet, allen Nachrichten zuzuhören, die im Rahmen der *conversation* ausgetauscht werden; so können z.B. private Auktionen wunderbar als *conversation* implementiert werden..

XL unterstützt die Definition von Web Services, die an *conversations* teilnehmen, auf zwei Arten.

Zuerst befreit XL den Programmierer von manueller Verwaltung der Kontext-Daten, die mit einer *conversation* assoziiert werden, indem es „*conversation*- Instanz“-Variablen bereitstellt. Bleiben wir bei unserem Beispiel: Für die Auktions-Website ist der Kontext einer speziellen Auktion die Produktspezifikation; der Kunde, der das Produkt verkaufen will; das Höchstgebot; der Name des Kunden, der das Höchstgebot abgab, und die Liste von Kunden, die an einer speziellen Auktion teilnehmen und über den Stand der Auktion informiert werden wollen. Offensichtlich können auf einer Auktions- Webseite mehrere Auktionen gleichzeitig stattfinden, so dass viele Instanzen dieser Variablen gespeichert werden müssen: eine Instanz jeder Variablen für jede Auktion.
Wenn die Auktions- Webseite eine Nachricht bekommt, die zur Auktion X gehört und die eine Ausführung der Operation closeAuction beinhaltet, dann wird von XL *automatisch* die richtige Instanz z.B. der Variable highest_Bid geladen und kann in der Definition der closeAuction-Operation verwendet werden.
Für einen Kunden könnte der Kontext einer Auktion einfach das Höchstgebot und die Information, ob dieses Gebot von dem Kunden selbst abgegeben wurde, sein; diese Information könnte, sagen wir mal, in der Operation react gebraucht werden, die in einem Web Service des Kunden definiert wurde, u.s.w.

Der zweite Weg, auf dem XL die Implementierung der *conversations* erleichtert, ist die deklarative Definition der *conversation*- Muster. Es würde viel Arbeit ausmachen und viele Fehler fabrizieren , wenn die Programmierer manuell die *conversations* und ihre URIs handhaben würden. Glücklicherweise gibt es in der Praxis nur eine Handvoll von verschiedenen Mustern, welche die Teilnahme der Web Services an *conversations* beschreiben. XL besitzt eine Menge von vordefinierten Mustern und gestattet den Programmierern, sie zu deklarieren; es werden dann automatische Schritte durchgeführt, um sie zu implementieren.
Wählt zum Beispiel der Implementierer ein „Mandatory conversation“ („verbindliche Unterhaltung“)- Muster, dann muss jede eingehende Nachricht eine *conversation*-URI enthalten (d.h. Teil einer *conversation* sein); sonst wird ein Fehler ausgegeben. Auch fordert das „mandatory“ Muster, dass die *conversation*- URI einer ausgehenden Nachricht (z.B. elektronische Zahlung) die gleiche ist wie die conversation- URI der entsprechenden eingegangenen Nachricht (z.B. ein Kaufauftrag , der den Zahlungsvorgang ausgelöst hatte).
Mit anderen Worten: Die eingehenden und ausgehenden Nachrichten sind Teil der gleichen *conversation*.

3.4 XL Anweisungen

Der Rumpf einer XL Operation wird von Anweisungen beschrieben, die Erweiterungen der XQuery- Ausdrücke sind. Zusätzlich zu klassischen imperativen Anweisungen wie Variablenzuweisungen, bedingten Ausdrücke, Schleifen, Fehlerbehandlungen und Return- Anweisungen, unterstützt XL einige XML spezifische Update- Anweisungen und einige Web Services- spezifische Anweisungen (z.B. Web Service- Aufruf, Logging, „sleep“). Schließlich unterstützt XL zusätzlich zu dem klassischen iterativen Anweisungskombinator (Sequenzierung) andere Anweisungskombinatoren aus den Workflow- und Datenfluß-Theorien (z.B. Parallelität, Auswahl).
Im Folgenden werden kurz die wichtigsten Anweisungen erläutert.

(In den Ausdrücken einer XL Anweisung sind alle globalen Variablen gültig, die in dem Variablendeklarationsteil vom Web Service deklariert wurden, sowie auch alle lokalen Variablen, die in der aktuellen Operation definiert wurden.
Dann sind für jede Operation noch zwei weitere lokale Variablen *impliziert* definiert: *$input*- Variable, die automatisch an den Rumpf der SOAP- Nachricht gebunden wird, welche die aktuelle Operationsausführung auslöste, und *$output*- Variable, deren Wert implizit den Rumpf der Antwort-Nachricht bildet.)

3.4.1. Variablenzuweisung

Die einfachste Anweisung ist die Zuweisung einer (globalen oder lokalen) Variable. Die Syntax ist wie folgt:

Let [type] variable := expression

Jeder durch den *W3C Query*- Vorschlag genehmigte Ausdruck kann auf der rechten Seite der Zuweisung verwendet werden.
Lokale Variablen brauchen nicht deklariert sein, bevor sie verwendet werden.

Der Typ oder XML Schema einer lokalen Variable kann optional im Rahmen der ersten Zuweisung zu dieser Variable gesetzt werden. (Globale Variablen *müssen* deklariert werden; der Typ einer globalen Variable kann optional gesetzt werden , wenn die Variable deklariert wird).
Wie in Java ist der Gültigkeitsbereich einer lokalen Variable der Block, in dem die Variable definiert ist.

3.4.2. Update Anweisungen

Leider stellt XQuery noch keine Ausdrücke bereit, um XML Daten zu manipulieren. Don Chamberlain u.a. entwickelten einen Arbeitsentwurf, um XQuery in diesem Hinblick zu erweitern und, sobald eine Empfehlung durch das W3C gegeben wird, wird XL die entsprechende Syntax und Semantik dieser Ausdrücke adoptieren.

Mittlerweile können folgenden Anweisungen benutzt werden, um XML Daten zu manipulieren:

- **Insert**, um neue Knoten in die XML Hierarchie einzufügen (z.B. ein zusätzliches Kreditkarten-Element)
  - insert <creditcard> . . .</creditcard>
    into $client
- **Delete**, um Knoten aus der XML Hierarchie zu löschen (z.B. Visa Kreditkarte)
  - delete $client / creditcard [type="Visa"]
- **Replace**, um Elemente zu ersetzen (z.B. die Telefonnummer)
  - replace $client/telephone with
    <telephone>(007)906090</telephone>
- **Rename**, um bestimmte Knoten (Elemente oder Attribute) umzubenennen:
  - rename $client/name as „fullname"
- **Move**, um XML Knoten an einen anderen Ort im XML Baum zu verschieben, dabei bleiben die interne Struktur und die Knoten- Identifikatoren erhalten
  - move $client/Telephone
    after $client/city

3.4.3. Serviceaufruf- Anweisungen

Wahrscheinlich sind die relevantesten atomaren Anweisungen in XL jene, die für den Aufruf anderer Web Services zuständig sind; d.h. jene, die Nachrichten zu einem anderen Web Service verschicken. Oft wird der andere Web Service in XL geschrieben sein, aber es können Nachrichten zu jedem Service verschickt werden, der eine URI hat und auf SOAP Nachrichten reagiert.
Web Services werden unabhängig von der Art ihrer Implementierung aufgerufen. XL bietet zwei Methoden an, einen Web Service als Teil eines XL Programms aufzurufen: synchrone und asynchrone Methode.

Die Syntax eines **synchronen Aufrufes** ist wie folgt:

<expression> → <uri>[::<operation>]
[→ <variable>]

Die Semantik ist direkt verständlich. Eine Nachricht mit dem Wert von „expression" wird zu dem durch „uri" identifizierten Web Service geschickt.

Falls eine spezielle Operation des Web Service aufgerufen werden soll, dann kann der Name der Operation spezifiziert werden; sonst wird vom Empfänger erwartet, die Nachricht ohne die Spezifikation der Operation zu verstehen. In einem synchronen Aufruf wird die Ausführung verzögert, bis der aufgerufene Web Service seine Ausführung beendet und das Ergebnis oder einen Fehler zurückgibt- wobei beides in eine SOAP- Nachricht eingebettet wird.
Falls eine Variable im Teil des Aufrufs auftritt, dann wird der Rumpf der durch den aufgerufenen Service zurückgegebenen Nachricht in diese Variable kopiert (entspricht *return* in Java).
Die Nachricht wird genau einmal und auf effizienteste Art verschickt.

Als Beispiel betrachten wir den folgenden synchronen Service- Aufruf , der den Online- Broker bittet, 1000 SAP- Aktien für maximal 140 € zu kaufen; das Ergebnis wird in der *$receipt* Variable gespeichert:

```
<order>     <stock>SAP</stock>
            <limit>140</limit>
            <currency>Euro</currency>
            <amount>1000</amount>
</order>    → http://www.broker.com :: buy
            → $receipt
```

Die Syntax eines **asynchronen** Aufrufes ist ähnlich der des synchronen:

```
<expression> ==> <uri> [::<operation>]
            [==> <operation>]
```

Hinsichtlich der Semantik wird in diesem Fall die Ausführung nicht blockiert: Das Programm wird sofort mit der Ausführung der nächsten Anweisung fortsetzen, nachdem die Nachricht an den aufgerufenen Service verschickt wurde.

Falls die Ausgabe (Antwort oder Fehlermeldung) verarbeitet werden muss, dann kann der Name der Operation, die das asynchrone Ergebnis verarbeiten wird, als Teil des Aufrufs angegeben werden; diese Operation muss zum Web Service gehören, von dem der asynchrone Aufruf stammte.
Wieder wird die Nachricht genau einmal und am effizientesten verschickt.
.
Gegenwärtig stellt XL keine Möglichkeit zur Verfügung, den Umschlag der SOAP-Nachricht explizit zu bestimmen. Solche Konstrukte könnten nützlich sein, um bestimmte Arten der *conversations*, *Quality Of Service*-Garantien, und/oder verteilte Transaktionen und sichere Nachrichten zu implementieren.
Es wird geplant, XL zu erweitern, sobald SOAP und bevorstehende XML Protokoll-Empfehlung sich stabilisiert haben.

3.4.4. Bedingte Anweisungen und Iterationen

Wie die meisten anderen Programmiersprachen stellt XL die IF-THEN-ELSE Anweisung, die WHILE- Schleife und die DO WHILE- Schleife bereit. Die Semantik ist direkt verständlich und die gleiche wie in anderen imperativen Programmiersprachen.

Als Beispiele steht unten ein IF- THEN- ELSE- Konstrukt:

If (<booleanExpression>)
Then
    <statement>
Endif else
    <statement>
endelse

Zusätzlich zu diesen allgemeinbekannten Kontroll- Anweisungen unterstützt XL eine FOR- LET- WHERE- DO Schleife, mit der folgenden Syntax:

For <variable> in <expression>
Let <variable> in <expression>
Where <booleanExpression>
Do <statement>

Die FOR- LET- WHERE- DO Schleife entspricht den FLWR Ausdrücken in XQuery, aber sie führt Anweisungen aus (*mit potentiellen Seiteneffekten*), anstatt die Ausdrücke seiteneffektfrei auszuwerten, z.B. kann eine Zählervariable im DO- Teil erhöht und ausgegeben werden.

3.4.5. Ausnahmebehandlungen

In XL implementierte Web Services signalisieren Fehler durchs Werfen von Ausnahmen, genau wie bei Java oder C++.
Die Syntax der XL Anweisung, die eine Ausnahme wirft, ist wie folgt:

throw <expression>

Hier kann „expression“ jede Art von XQuery- Ausdrücken sein.
Falls die Ausnahme nicht lokal behandelt wird, wird die Ausführung der Operation beendet, und der Wert des Ausdruckes (statt des Wertes der *$output* Variable) wird als eine SOAP- Nachricht zum Aufrufer des Services zurückgeliefert.
Genau wie Variablen und jeder andere Ausdruck können die Ausnahmen optional auf starke Art typisiert werden.

XL adoptiert auch die Java Syntax fürs Fangen von Ausnahmen. TRY zeigt an, dass eine Anweisung (oder eine Folge von Anweisungen) Ausnahmen bewirken kann; CATCH wird verwendet, um Kode zu schreiben, der auf Ausnahmen reagiert.
Die Syntax ist wie folgt:

Try <statement>
endtry
Catch <variable> do
    <statement>
endcatch

Die Variable in der CATCH- Anweisung ist an den Wert der Daten der Ausnahme gebunden, welche beim Ausführen der Anweisung(en) innerhalb des TRY- Blocks geworfen wurde.
Wie in Java wird eine gefangene Ausnahme die Ausführung der entsprechenden Anweisung auslösen.

3.5 XL Kombinatoren

Offensichtlich kann der Rumpf eines XL- Programms mehr als eine atomare Anweisung enthalten. Es gibt mehrere Arten, Anweisungen zu kombinieren. Im folgenden stellen "statement1" und "statement2" beliebige atomare Anweisungen, oder Kombinationen von den Anweisungen, dar.

Die häufigste Methode, Aussagen zu kombinieren, ist die Benutzung des ";" Symbols, wie in C++ oder Java. Es bedeutet, dass "statement1" vor dem „statement 2“ ausgeführt wird:
.
<statement1> ; <statement2>

Desweiteren stellt XL eine Menge zusätzlicher Kombinatoren bereit:

- **? Einfache Fehlerbehandlung**. Falls "statement1" scheitert, führen „statement2“ aus.
- **| Nichtdeterministische Auswahl.** Führen entweder "statement1" oder "statement2" aus, aber nicht beide.
- **|| Parallelität**. Führen "statement1" und "statement2" parallel aus.
- **& Abhängigkeiten**. Je nach existierenden Abhängigkeiten zwischen den Anweisungen wird die Ausführungsreihenfolge vom System selber gewählt.

3.5.1. Block

Wie in C++ und Java benutzt XL eine Syntax, um einen Block von Anweisungen zu Identifizieren.
Der Rumpf eines XL Programms kann beispielsweise als ein Block von Anweisungen geformt sein. Der Gültigkeitsbereich einer Variableninstanz ist der Block von Anweisungen, in dem die Variable zum ersten Mal instanziiert wird.

Begin
    <statement>
End

3.6. XL Architektur

Zur Ausführung von Web Services soll eine XL Plattform bereitgestellt werden, die mit folgenden vier Zielen entwickelt wird:

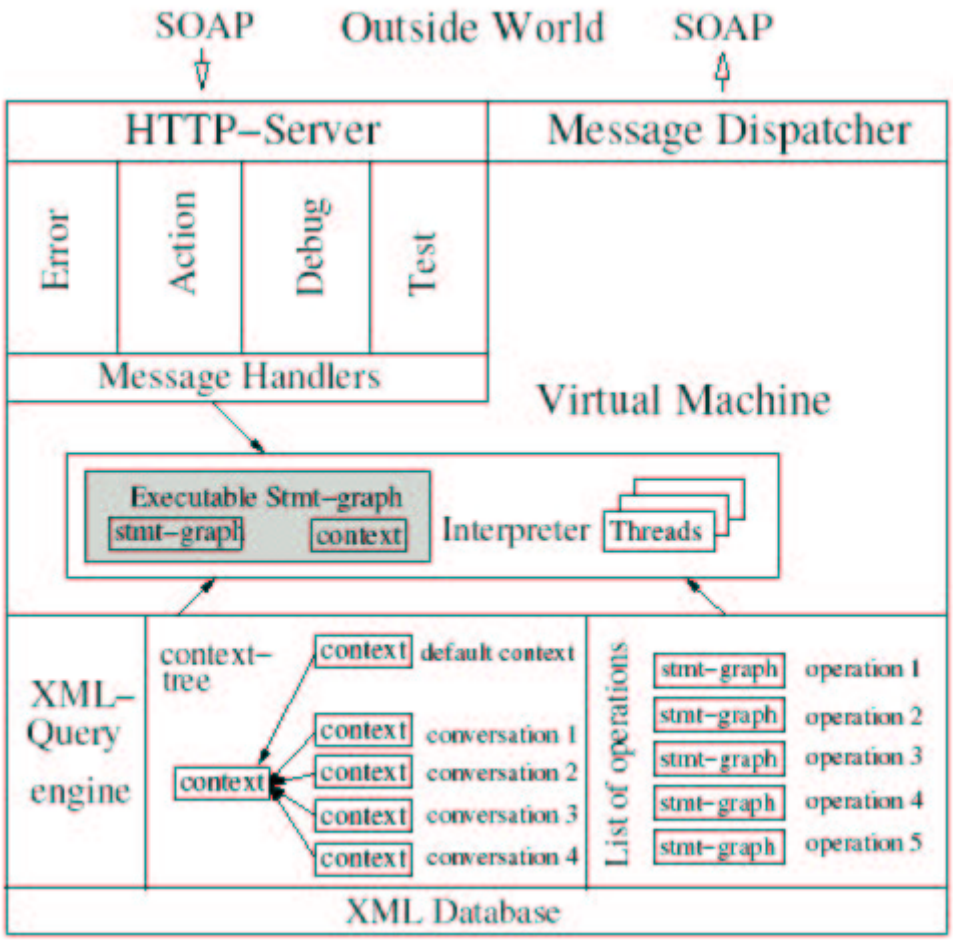
a) hohe Performance

b) hohe Zuverlässigkeit
c) hohes Sicherheitsniveau und
d) sehr geringe oder gar keine Administrationskosten (d.h. automatische Optimierung).

Die XL- Entwickler finden, dass das deklarative *high-level* Programmier-Modell von XL das Erreichen dieser Ziele ermöglicht, während diese Absichten in einer Umgebung, die vollständig auf einer imperativen *low-level* Programmiersprache wie Java oder C# basiert, sehr schwer realisiert werden können.

Die Implementierung der XL Plattform wird durch Kombination der Techniken aus verschiedenen Informatik- Bereichen- vor allem: Datenbanksysteme, Compilerbau, verteilte Systeme, Netzwerke, und Datenfluss-Verarbeitung- bewerkstelligt. Allerdings soll hier angemerkt werden, dass im Moment im vorhandenen Prototyp nicht alle diese Techniken integriert sind. Jedoch wird hart daran gearbeitet, die fehlenden Techniken dem System hinzuzufügen.

3.6.1. Übersicht der Architektur



Die XL Plattform besteht aus zwei Hauptkomponenten:
(a) XL Compiler und
(b) XL *virtual machine*.

Die Rolle des XL Compilers besteht in der Übersetzung der Text- Repräsentation eines XL Programms in den sogenannten **Anweisungsgraph** („statement graph“).
Zum Erstellen eines Anweisungsgraphs wird eine **Kern- „Algebra“**- eine Menge von einfachen Grundanweisungen- verwendet, wobei diese „Algebra“ die *ganze* XL Semantik unterstützt.

Die Kompilierung besteht aus zwei Schritten. Zuerst wird die textuelle Repräsentation eines XL- Programms vom XL Parser in einen naiven (also nicht-optimalen) Anweisungsgraph übersetzt. Dann wandelt der XL- *Optimierer* diesen naiven Anweisungsgraph in einen äquivalenten, aber effizienteren Anweisungsgraph um.
Die Äquivalenz von Anweisungsgraphen wird hinsichtlich der Gleichwertigkeit der von ihnen zurückgegebenen Ergebnisse und auch hinsichtlich der Gleichwertigkeit von ihren Wirkungen auf den Kontext einer „*conversation*“, in der sie ausgeführt werden (z.B. Variablen- Updates, gesendete Nachrichten), definiert.

Der Anweisungsgraph, der vom Optimierer erstellt wird, wird von der XL *Virtual Machine* jedes Mal, wenn eine auslösende Nachricht eingeht, interpretiert. Dieser Prozess kann wie folgt beschrieben werden.

Zuerst wird jede eingehende Nachricht von einem *„message handler“* verarbeitet. Die XL- „*Message handlers*“ werden entsprechend verschiedenen Interaktionsmodi mit der XL *Virtual Machine* („normale Aktion“, „Fehler“ u. ggf. andere) verwendet.
Als nächstens, wenn der „*message handler*“ für „normale Aktionen“ eine SOAP Nachricht an die *Virtual Machine* weitergibt, lädt die *Virtual Machine* (falls noch nicht im Cache) den Anweisungsgraph der aufgerufenen Operation aus der Datenbank und führt die Operation aus. Die Ausführung findet in einem bestimmten Kontext, der alle für die Ausführung nötigen Informationen beinhaltet, statt.

Die Daten, die als Ergebnis der Ausführung des Anweisungsgraphs produziert wurden, werden an den Aufrufer in einer SOAP- Nachricht integriert verschickt.

Drei wichtige Merkmale der XL *Virtual Machine* sollen im weiteren herausgestellt werden.
Erstens, um Anweisungen parallel auszuführen, ist die *Virtual Machine* „*multi-threaded*“.
Zweitens wurde die *Virtual Machine* so entworfen, dass sie fähig ist, aus den zwischen den Anweisungen vermittelten Daten *Ströme* („streams“) herzustellen. *Pipelining* ist ein sehr wichtiges Merkmal des XL- Entwurfs.
Drittens, um Skalierbarkeit und hohe Zuverlässigkeit zu erreichen, ist die XL *Virtual Machine* so entworfen worden, dass sie die Migration von Prozessen von einer Maschine zu einer anderen Maschine im Cluster unterstützt (es wird erwartet, dass die XL- Plattform auf einem Cluster von Servern installiert wird).

Diejenigen Anweisungen und Aktionen, die Werte von globalen Variablen verändern, müssen mit einer *XML Datenbank* interagieren, welche die entsprechenden Werte speichert. Desweiteren speichert die XML Datenbank alle Kontext- Informationen und eine XML- Repräsentation von dem Anweisungsgraph für jede Operation.

Die richtige *interne XML Daten- Repräsentation* ist für gute Performance in der *Virtual Machine* von höchster Bedeutung. Die XL Programme, wie oben bereits erwähnt, manipulieren ausschließlich XML Daten (die Werte der Variablen sind XML-Dokumente); die Programme selber (bzw. ihre Anweisungsgraphen) sowie ihre Kontexte, die ebenso in XML repräsentiert sind.

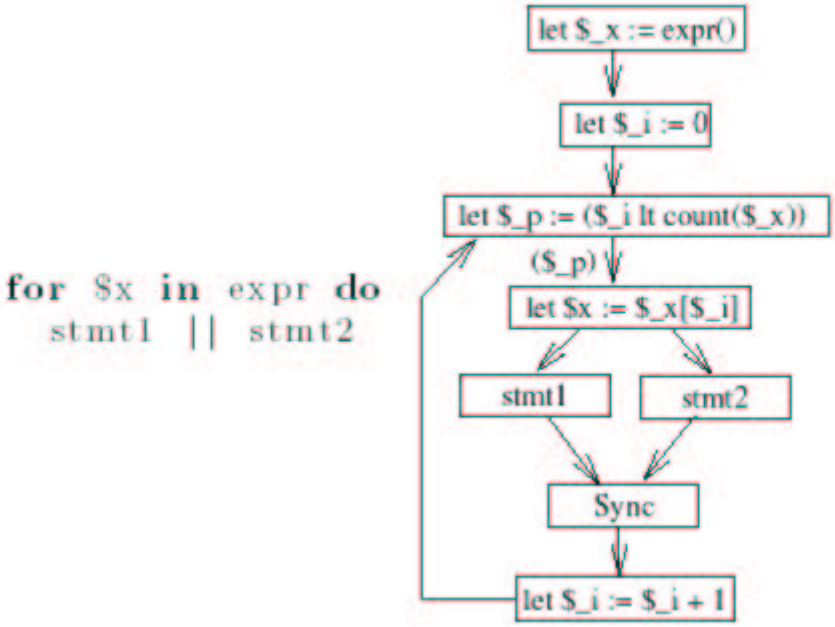
Innerhalb der XL- Plattform werden XML Daten als ein *Vektor von Zeichen* („vector of tokens“) repräsentiert; der XML Parser erzeugt eine Sequenz von Zeichen für jedes XML- Dokument auf ähnliche Weise wie ein SAX Parser Ereignisse erzeugt.
Die Repräsentierung der XML Daten als Vektoren von Zeichen ermöglicht eine sehr kompakte interne Darstellung, und sie gestattet die Verarbeitung der Daten auf einem *Strom- basierten* („stream-based“) Weg. In vielen Fällen braucht ein solcher Vektor nie materialisiert werden; stattdessen werden die Anweisungen als *Iteratoren* implementiert und verbrauchen die Zeichen in einem pull-basierten Paradigma.
XML Daten existiert während des ganzen Prozesses innerhalb der XL *Virtual Machine* in diesem internen Format und werden nur am Ende der Ausführung einer Operation zurück in einen XML- String serialisiert. Das Ergebnis/ die Antwort wird als SOAP- Nachricht zurückgeschickt.

3.6.2. XL Kern "Algebra"

Obwohl XL eine ziemlich mächtige Programmiersprache ist, werden alle Konstrukte von einer einfachen Algebra aus nur sechs Anweisungen unterstützt. Diese Einfachheit macht es leicht, XL zu optimieren bzw. hoch vertrauliche und sichere Plattformen für XL zu bauen. Die XL Kernalgebra entspricht in gewissem Maße der relationalen Algebra im Datenbankbereich. Genau wie dort können diese Kern-Anweisungen auf verschiedene Weise implementiert werden (z.B. pipelined und „nicht – pipelined“). Es liegt in der Verantwortung des XL- Optimierers, die beste Implementierung auszuführen.
Die selbsterklärenden **Grundanweisungen** sind: *Assignment, Send, Receive, Wait, Sync und Update.*

3.6.3 Anweisungsgraph

Jede Operation eines XL Webdienstes wird abstrakt durch einen Anweisungsgraph, ähnlich den Anweisungsgraphen in der Datenflussanalyse und Codeoptimierung, repräsentiert.



Die Knoten dieses Anweisungsgraphs sind genau die oben beschriebenen Kern-Anweisungen, und die Kanten im Graph kodieren die Reihenfolge, in der die Anweisungen ausgeführt werden sollen. Eine Kante von einer Anweisung S1 zur Anweisung S2 zeigt an, dass S1 vor S2 ausgeführt werden muss. Die im Graph nicht geordneten Anweisungen können parallel ausgeführt werden. Kanten können mit boolschen Variablen, welche die Bedingung der Ausführung einer Anweisung nach der Anderen kennzeichnen, beschriftet werden. So kodiert der obere Anweisungsgraph die if- Anweisungen und Schleifen.

Auch Knoten des Anweisungsgraphs können mit zusätzlichen Informationen (Kosten etc.) beschriftet werden.

3.6.4 Ausführungskontext

Jede XL Anweisung wird in einem Kontext kompiliert und ausgeführt;
der Kontext beinhaltet die Werte der Variablen und alle andere Informationen, die gebraucht werden, Ausdrücke auszuwerten. Die XL Anweisungen könnten z.B. XQuery- Ausdrücke auswerten; solche Ausdrücke brauchen auch einen Auswertungskontext, der Informationen wie importierte Schemata, Namespace Definitionen etc. beinhaltet.
Kontexte können dynamisch erzeugt werden und werden in einer Hierarchie organisiert. In einer solchen Hierarchie von Kontexten wird ein XQuery- Ausdruck in einem speziellen Kontext ausgeführt. Falls dieser Kontext eine spezielle im Ausdruck verwendete Definition nicht beinhaltet, dann wird der Elternkontext inspiziert.
Die Wurzel der Kontexthierarchie, der Basiskontext, beinhaltet die Definition aller eingebauten XQuery- Typen, Namespaces und Funktionen. Dieses Merkmal wird kommt in der XL *Virtual Machine* zur Verwertung, um Gültigkeitsbereiche von Variablen und *conversations* zu implementieren (d.h., um den *conversation*- Kontext zu organisieren).

3.6.5. Optimierung

Allgemein geht es bei der Optimierung darum, einen Anweisungsgraph in einen effizienter Anweisungsgraph umzuwandeln. Dazu werden bekannte Techniken aus den Gebieten Compilerbau und Datenbankentwicklung sowie XL- spezifische Techniken verwendet.
Die Liste der hier angegebenen Methoden ist nicht komplett; sie gibt aber einen Einstieg in die Möglichkeiten, die XL aufgrund seiner *high-level* Abstraktion einer Programmiersprache und der sehr einfachen Kernalgebra sowie der Verwendung der Anweisungsgraphen zur Implementierung der Sprache bieten kann.

- **Gemeinsame Unterausdrücke:** Identifiziere (Unter-)Ausdrücke in verschiedenen Anweisungen, die garantiert den gleichen Wert liefern.
- **Umordnung der Anweisungen:** Identifiziere Anweisungen, die parallel ausgeführt werden können. Desweiteren identifiziere Anweisungen, die auf den gleichen Daten operieren, um die temporäre Lokalität der Anwendung zu verbessern
- ***Multi- query* Optimierung:** In bestimmten Situationen könnte es besser sein, zwei Ausdrücke gleichzeitig, anstatt jeden Ausdruck individuell, auszuwerten. Solche Situationen können in XL entdeckt werden, weil der Compiler eine *globalere Sicht* auf die Anwendung hat, als traditionelle Datenbank-Compiler (d.h. das ganze Programm, und nicht nur eine isolierte Anfrage, wird analysiert).
- **Batched Updates:** Ersparen von Hin - und Rück- Nachrichten zur Datenbank
- **Identifikation lokaler Service- Aufrufe:** Beseitigung des hohen SOAP-Overheads; gemeinsame Optimierung des Aufrufers und des aufgerufenen Service zur besseren Performance.
- **Identifikation lokaler seiteneffektfreier Serviceaufrufe:** Solche Aufrufe könnten unter Umständen ganz vermieden werden; falls nicht, so können die Ergebnisse zumindest im Cache gespeichert werden.
- **IF- THEN- ELSE- Optimierungen:** Falls die Bedingungen der „nested“ *if-then-else*- Anweisungen disjunkt sind, kann die Reihenfolge der Auswertung der Bedingungen *getunt* werden (z.B. je nach Selektivität oder Kosten). Desweiteren können Materialisierung und Hashing verwendet werden, um bei der Suche nach dem richtigen Zweig der tief „nested“ *if- then- else-* oder *switch-* Anweisungen Kosten zu sparen.
  Das ist eine besonders wichtige Funktionalität, wenn man bedenkt, dass der Trend in Web Services zur Personalisierung führt.

3.6.6. *Streaming* Ausführung

XL *Virtual Machine* verwendet den Prozess des *Pipelining* und kann somit als eine Datenflussmaschine charakterisiert werden.
Pipelining bedeutet, dass „Ergebnisse“ einer Anweisung an andere Anweisungen stückweise („token by token“) weitergegeben werden. Dazu werden die in Frage kommenden Anweisungen im Programm bestimmt (außer „event“ und „sync“ unterstützen nämlich alle Kern- Anweisungen das Pipelining) und als Iteratoren

implementiert. Im Anweisungsgraph kann man sich dann eine Kante als eine Menge von Pipelines (eine pro Variable) vorstellen.

Die Gründe für Pipelining in der XL- Plattform sind im wesentlichen die gleichen wie in kommerziellen Datenbanksystemen.

- Pipelining ermöglicht schnellere Antwortzeiten, d.h. das System liefert die ersten Antworten früher zurück.
- Pipelining reduziert die Hauptspeicheranforderungen und reduziert Kosten der Materialisierung von Zwischenergebnissen (z.B. im Hauptspeicher oder auf der Platte), bzw. des nicht-erfolgreichen Swappings.
- Pipelining ist eine Form der Parallelität und kann deshalb die Gesamtantwortzeit reduzieren.
- Pipelining ist insbesondere wichtig, falls es Staus im Netzwerk gibt: Ein Service kann die Verarbeitung der ersten Eingaben starten, während er auf den Rest der Eingabe wartet.
- Pipelining ist sehr hilfreich, um unnötige Arbeit zu meiden oder sogar sicherzustellen, dass bestimmte Operationen enden- auf ähnliche Weise wie „*lazy evaluation*“ es in den Programmiersprachen macht.

  Dazu folgendes Beispiel:

  let $a := (0, 1, 0, 1, 0, 1 ...) // z.B. unendliche Eingabe eines Sensors
  if (some $b in $ a satisfies $b = 0) then
  let $output := “Es gibt mindestens eine Null”;
  endif

  Ohne Pipelining würde das Programm nie über die erste (let) Anweisung kommen. Mit Pipeling erfolgt aber schon bald die Ausgabe „Es gibt mindestens eine Null“.

Natürlich ist Pipelining nicht immer das beste Ausführungsmodell, und manchmal bereitet Pipelining zusätzlich Kosten, z.B. für Verarbeitung individueller Zeichen („tokens“) oder für Synchronisierung, falls mehrere Anweisungen die Ergebnisse einer Anweisung verarbeiten. Insbesondere muss Pipelining mit großer Sorgfalt in Iterationen einer Schleife benutzt werden.

3.6.7 Prozess-Migration innerhalb eines Clusters

Um Skalierbarkeit und hohe Zuverlässigkeit zu erreichen, wird erwartet, dass die Installation der XL Plattform auf einem Cluster von Services stattfindet. Auf jedem Server soll eine eigene XL *Virtual Machine* laufen.
Für die Migration eines Prozesses (z.B. einer *conversation*) muss nur dessen Kontext auf einen anderen Server verlagert werden. Prozessmigration ist insbesondere wichtig in der Web Services- Umgebung, weil die Interaktion mit externen, autonomen Web Services (und Anwendern) Nachrichtenumlauf- und Wartezeiten von Tagen ergeben kann.
Für komplexe Operationen kann die *Virtual Machine* so konfiguriert werden, dass sie Teile einer Aktivität auf verschiedenen Maschinen parallel ausführen soll. Dadurch

kommen Pipelining und unabhängige Parallelität innerhalb einer Operation zum Vorteil.
Wieder finden die Entwickler, dass XL hier gegenüber Java hinsichtlich der Einfachheit Vorteile bietet, da XL's Programmiermodel sehr einfache und saubere Semantik (Kern- Algebra) besitzt.
Allerdings müsste am Beweis dieser Aussage noch weiter geforscht werden.

4. BPEL4WS

4.1. Einführung

4.1.1. Grober Überblick

Business Process Execution Language For Web Services definiert ein Modell und eine Grammatik für die Beschreibung des Verhaltens eines Geschäftsprozesses („business process") basierend auf Interaktionen zwischen dem Prozess und seinen Partnern. Die Interaktion mit jedem Partner geschieht durch die Web Service-Schnittstellen (verfasst in WSDL). Die Struktur der Beziehung am Schnittstellenniveau (z.B. welcher Service welche Rolle spielt) wird in einen sogenannten *Service Link* eingekapselt.
In BPEL4WS lassen sich für Prozesse die gewünschte Koordination der Service-Interaktionen zwischen den Partnern sowie der Prozesszustand bzw. die Logik, die für die Koordinierung notwendig sind, definieren. Außerdem führt BPEL4WS bietet Mechanismen für den Umgang mit Ausnahmen („business exceptions") und Verabreitungsfehlern an.
Schließlich kann man in BPEL4WS für einen Prozess festlegen, wie individuelle oder zusammengesetzte Aktivitäten innerhalb dieses Prozesses im Falle des Auftretens der Ausnahmen oder im Falle der Rücknahmen durch den Partner kompensiert werden müssen („compensation").

4.1.2. Einsatzmöglichkeiten von BPEL4WS

BPEL4WS unterscheidet folgende zwei Arten von Geschäftsprozessen:

- **Ausführbare Geschäft-Prozesse** („executable business processes") modellieren tatsächliches Verhalten der Teilnehmer einer Geschäftsbeziehung („business interaction"). Sie werden meist zur Integration der Unternehmensanwendungen geschrieben („enterprise application integration")
- **Geschäftsprotokolle** („business protocols") verwenden Prozessbeschreibungen, die öffentliche Bestandteile des Verhaltens beim Nachrichtenaustausch zwischen jeder der am Protokoll beteiligten Partei spezifizieren- ohne deren internes Verhalten zu enthüllen. Die Prozess-Beschreibungen für Geschäftsprotokolle heißen „**abstrakte Prozesse**" („abstract processes"). Sie werden für die Spezifikation in E- Business-Beriechen benötigt, beziehen sich nur auf WSDL- Schnittstellen tatsächlicher Dienste und können somit nicht ausgeführt werden.

Die Grundkonzepte von BPEL4WS können auf zwei Arten angewendet werden.

Ein abstrakter BPEL4WS Prozess kann für einen Partner *eine Rolle* bezüglich eines Geschäftsprotokolls definieren. Zum Beispiel sind in einem Protokoll der Vorratskette der Käufer und der Verkäufer zwei unterschiedliche Rollen, jede mit seinem eigenen abstrakten Prozess. Ihre Beziehung wird typischerweise als ein *Service Link* modelliert. In abstrakten Prozessen können alle Konzepte von BPEL4WS benutzt werden, außer einigen Ausnahmen bei der Datenbehandlung („data handling"). Dafür wird aber ein weiteres Konzept angeboten: die undurchsichtige („opaque") Variablenzuweisung- so kann tatsächlich erreicht werden, dass die Implementierungsdetails versteckt bleiben.

Im Gegensatz zum abstrakten Prozess wird im ausführbaren Geschäftsprozess nicht nur bestimmt, wie der Prozess zwischen den Geschäftspartnern interagiert, sondern es wird auch seine Logik und sein Wesen komplett implementiert.

Es ist aus der Sicht einer eigenen Implementierung möglich, dass eine Prozess-Definition nicht vollständig in BPEL4WS geschrieben ist, denn die Sprache definiert ein portables Ausführungsformat für diejenigen Geschäftsprozesse, die sich ausschließlich auf Web Service Ressourcen und XML Daten stützen.
Desweiteren kann die Interaktion der BPEL4WS- Prozesse mit ihren Partnern,oder die Ausführung ihrer Partner, ungeachtet der unterstützten Plattform oder des Programmiermodells der Host- Umgebung verlaufen.

Sogar da, wo eigene Implementierungen plattformabhängige Funktionalität nutzen- was wahrscheinlich in vielen, wenn nicht in den meisten Fällen realistisch ist- macht es BPEL4WS durch sein Modell möglich, öffentliche Aspekte der Geschäftsprotokolle als Prozesse zu ex- und importieren, um sie plattformunabhängig zu nutzen.

Dieses ist wohl der attraktivste Vorteil von BPEL4WS hinsichtlich dem Aufbruch des Potentials der Web Services. Es erlaubt nämlich, andere Werkzeugen und Technologien zu entwickeln, die stark die Automatisierung fördern und damit die Kosten der Herstellung von zwischenbetrieblichen automatisierten Geschäftsprozessen senken.

4.1.3. Einfließende Technologien

BPEL4WS verwendet als Basis verschiedene Standarte: WSDL, XML Schema, und XPath.

WSDL findet eine starke Verwendung bei BPEL4WS. So bildet die Interaktion zwischen den WSDL Services den Kern des BPEL4WS Prozessmodells. WSDL Nachrichten und XML Schema Typdefinitionen sind die Grundlage des verwendeten Datenmodells. Alle externen Partner oder Ressourcen werden als WSDL Services repräsentiert. Die BPEL4WS Prozesse selber werden auch als WSDL Services veröffentlicht.

Sehr wichtig ist auch der Standard Xpath, der die Grundlage für Unterstützung der Datenmanipulation bildet.

Die Entwicklers rühmen sich besonders damit, dass zukünftige Standarte relativ leicht in BPEL4WS übernommen werden können.

4.2. Prozess

Ein Prozess kann als eine aufeinander abgestimmte Menge von Web Service Interaktionen gesehen werden. Die Struktur eines Prozesses schau so aus:

```
<process ...>
        <partner> ... </partner>
        // Web Services, mit denen der Prozess interagiert
        <container>....</container>
        // Die vom Prozess verwendeten Daten
        <correlationSet></correlationSet>
        //zur Unterstützung asynchroner Interaktionen
        <faultHandler></faultHandler>
        <compensationHandler></compensationHandler>
        // Der bei der Rücknahme einer Aktion auszuführende Code
        mehrere Aktionen
        //das was der Prozess eigentlich macht
</process>
```

Auf einen Partner wird über einen Web Service- „*Kanal*“ („channel“) zugegriffen. Der Kanal wird durch den „Service Link Type“ mittels Rollen und bereits definierten portTypes (deren Operationen dem Rolleninhaber zur Verfügung stehen) dargestellt:

```
<partner name="Broker_Inc." serviceLinkType = "direct" partnerRole="…" myRole="…"/>
<serviceLinkType name="direct">
        <role name="seller">
                <portType name = "…"/>          //Element aus WSDL
        </role>
        <role name="buyer">
                <portType name="…"/>
                <portType name="…"/>
        </role>
</serviceLinkType>
```

4.3. Interaktionen

BPEL4WS kann viele Typen von Interaktionen modellieren: von einfachen, zustandslosen bis zustandsbehafteten, lang andauernden, asynchronen Interaktionen.
Speziell für den letzten Typ werden die sogenannten *„correlation sets“* verwendet. In den *correlation sets* werden Daten gespeichert, die den Zustand der Interaktion repräsentieren (ähnlich XL’s „*conversation*“ und deren Kontext). Durch *correlation sets* wird gewissermaßen ein gezielter Zugriff der eingehenden Nachrichten auf bestimmte Instanz des Prozesses ermöglicht. Jede *„correlation set“* wird

selbstverständlich nur ein Mal initialisiert, und ihre Werte ändern sich im Laufe der Interaktion nicht.
Beispiele für *correlation set*- Daten: „customerId“, „creditCardNumber“ etc.

<correlationSets>
<correlationSet name=“...“ properties=“aID bID"/>
//*Liste* von properties
</correlationSets>
<bpws:properties name=”…” type=”…”/>
// Globaler Name und XML Schema- Datentyp für jede Eigenschaft.
<bpws:propertyAlias propertyName=“...“ messageType=”…” part=”…” query=”…”/>

Jede *property* wird auf ein Feld im WSDL Nachrichtentyp abgebildet. Sie kann also als Teil von Nachrichten ausgetauscht werden.

<receive partner=“...“ operation=“...“ portType=”…” container=”…”>
<correlations>
<correlation set=”any_correlationSet_Name” initiation=”yes”/>
<correlation set=”doubleohseven”/>
</correlations>
</receive>

Eine Eingabe- (oder Ausgabe-) Operation stellt fest, welche *correlation sets* dieser angekommenen (oder verschickten) Nachricht entsprechen. Diese *correlation sets* werden verwendet, um die Beziehung der Nachricht zum passenden Zustand zu sichern. Wie oben erwähnt wird eine *correlation set* nur einmal mit dem Setzen von *initiation*- Attribut auf „yes“ initialisiert.

4.4. Das Datenmodell

Zusammen mit den empfangenen Nachrichten repräsentieren die *globalen* Variablen den Zustand der Prozessesinstanz. Diese Variablen werden *container* genannt.
Im Moment können in BPEL4WS keine lokalen *container* definiert werden, doch daran wird intensiv gearbeitet.

Ein Beispiel für einen Container:

<**containers**>
<**container name**="**orderDetails**">
<**wsdl:message name**="**orderDetails**">
<**part name**="**processDuration**" **type**="**xsd:duration**"/>
</**wsdl:message**>
</**container**>
….
</**containers**>

BPEL4WS- Variablen sind ausschließlich vom Typ der WSDL- Nachricht. Das erlaubt BPEL4WS jede Zustandsinstanz als Nachricht zu behandeln.

Am Anfang des Prozesses sind keine *containers* instanziiert. *Container*s können auf verschiedene Arten initialisiert werden, so z.B. durch explizite Zuweisungen oder Bindung an eine angehende Nachricht (s. nächster Abschnitt). Wird ein *container* verwendet, bevor er initialisiert wurde, wird eine *bpws:uninitializedContainer* Ausnahme geworfen.

4.5. Grundaktivitäten

<invoke partner="WebCom" portType="..." operation="getIp" inputContainer="…" outputContainer="…"/>

Der Prozess ruft eine Operation bei einem Partner auf. Die Operationsangabe ist im Gegensatz zu XL nicht optional.

<receive partner=".." portType="...." operation="…" container="…" [createInstance="…"]/>

Der Prozess erhält einen Aufruf von einem Partner. Es spezifiziert die Operation und den portType, deren Aufruf von der Aktivität erwartet wird.

<reply partner="..." portType="..." operation="…" container="…" />

Der Prozess sendet Antwortnachrichten an den aufgerufenen Partner

<assign>
<copy>
<from container="..." [part="…" /]> <to container="..." [part="…"/]>
</copy>
</assign>

Die Daten aus einem Container werden einem anderen zugewiesen.
Es gibt verschiedene Zuweisungsmöglichkeiten, einige sind nur den abstrakten Prozessen vorenthalten. Zur genaueren Informationen schauen Sie bitte in der Literaturangabe nach.

<throw faultName="...." faultContainer="..."/>

Der Prozess stellt einen Fehler fest und wechselt in Fehlerverarbeitungsmodus

<wait for="..." until="...">

<terminate/>

4.6. Strukturierungsaktivitäten

<sequence> //führe Aktivitäten sequentiell aus
<flow> //führe Aktivitäten parallel aus
<scope> //entspricht { und } in Java
<switch>
<while>
<pick>

Die „pick“- Aktivität erwartet das Auftreten einer Ereignismenge und führt dann diejenige Aktivität aus, die mit dem aufgetretenen Ereignis verbunden ist (enstpricht dem Java- *Listener*- Modell)
Das Auftreten der Ereignisse ist oft gegenseitig ausschließend („mutual exclusive“), z.B. wird ein Prozess entweder eine Akzeptanz- Nachricht oder eine Ablehnungsnachricht erhalten, aber nicht beide.
Falls mehr als eins der Ereignisse auftreten, dann hängt die Auswahl der auszuführenden Aktivität davon ab, welches Ereignis zuerst auftrat.

<link>

Link definiert eine Abhängigkeit zwischen einer Quell- und einer Zielaktivität

Natürlich können alle Aktivitäten ver“nested“ werten. Im Gegensatz zu XL wird hier die Strukturierung schon zur *Kompilierzeit* festgelegt (keine Optimierung der Aktivitätsgraphen)

4.7. Fehlerbehandlung und Kompensation

Für jeden (strukturierten) Programmbereich („scope“) können zwei Arten von “handlers“ definiert werden: die Fehlerbehandlungs- und die sogenannten Kompensationsroutinen („*compensation handler*“).

Ein „*compensation handler*“ wird verwendet, um die Ausführung eines bereis fertigen Programmbereiches („scope“) rückgängig zu machen. Ein spezieller *compensation handler* kann ausschließlich aus dem ihn direkt umgebenen Programmbereich aufgerufen werden.

Ein „*fault handler*“ definiert alternative Ausführungspfade, wenn ein Fehler innerhalb des „scope“ ausgelöst wird.

Beispiel:
<sequence ...>
    <invoke .../>
    <reply.../>
    <faultHandler…/>
</sequence>

Wenn ein Fehler sowohl bei <invoke> als auch bei <reply> auftritt wird der *fault handler* aufgerufen.

5. Zusammenfassung und Bewertung

Sowohl XL als auch BPEL4WS bauen auf gleichen Voraussetzungen im XML-Bereich auf und verwenden etablierte Standards bei der Implementierung von Web Services. Die Syntax ist in beiden Fällen weitgehend unterschiedlich, aber intuitiv verständlich und relativ leicht zu lernen. Die Semantik der meisten Anweisungen ist ähnlich, Unterschiede gibt es hier nur im Speziellen.

Bei der angebotenen Funktionalität steht es auch mehr oder weniger unentschieden. So kann XL zwar keine SOAP- Nachrichten direkt manipulieren, was BPEL4WS anbietet- doch kann man in BPEL4WS keine lokalen Variablen deklarieren, was wiederum in XL geht. Über die Handhabung des Variablentyps kann man gewiss auch streiten. Während BPEL4WS auf strengste Art nur einen Typ zulässt, brauchen in XL Variablen bei der Deklaration gar nicht typisiert werden. Beides hat sicherlich Vor- und Nachteile.

Was mir am wichtigsten erscheint, ist der Kontext in dem die Konkurrenten TU München bzw. IBM ihre Produkte anbieten.

Denn XL ist, so wie es von den Entwicklern vorgestellt wurde, nicht nur eine Sprache, sondern gleich ein ganzes System für Web Services. Das ist der entscheidende Vorteil.

BPEL4WS überlässt nämlich wichtige und komplexe Aspekte, wie Optimierung oder „*Quality Of Service*“- Zusicherungen, dem Programmierer. Es folgt hier dem traditionellen Ansatz, den man auch von purem C++ kennt.

XL als Plattform ähnelt dagegen mehr dem Java DK (wenn wir schon bei klassischen Sprachen sind)- hier wird es zum Beispiel gewünscht, dass komplexe Arbeiten dem Programmierer abgenommen werden, die Optimierung des Programms automatisch verläuft und damit hohe Qualität tatsächlich *garantiert* werden kann.

Gegen XL spricht vor allem, dass es immer noch im Prototyp- Stadium ist. Und da es sich bei den XL- Entwicklern um Leute von einer Universität handelt, die es mit den *time-to-market*- Anforderungen nie genau nehmen, ist zu erwarten, dass BPEL4WS seine Lücken früher füllt.

6. Literaturverzeichnis

Einführung

[1] “Web Services- Potenzial und Anwendung in der Finanz IT“, EXCELSIS Business Technology AG, www.excelsisnet.com

Technische Grundlagen

[2] “XL- A Plattform for Web Services“, TU München
http://xl.in.tum.de/publ/cidr2003.pdf

[3] “XL- An XML Programming Language for Web Service Specification and Composition”, TU München http://xl.in.tum.de/publ/www2002.pdf

XL

[2] “XL- A Plattform for Web Services“, TU München
http://xl.in.tum.de/publ/cidr2003.pdf

[3] “XL- An XML Programming Language for Web Service Specification and Composition”, TU München http://xl.in.tum.de/publ/www2002.pdf

BPEL4WS

[4] “BPEL4WS Version 1.0”, IBM developerWorks (u.a. Francisco Curbera, IBM; Yaron Goland, BEA Systems; Johannes Klein, Microsoft), ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf