

# Modellgetriebene Ablaufspezifikation

Jan Wittmer

Seminar Spezifikations- und Selektionsmethoden für Daten und Dienste  
Wintersemester 2005/2006

Universität Karlsruhe (TH)  
IPD, Lehrstuhl Prof. Böhm

Die Model Driven Architecture (MDA) ist ein viel versprechender Ansatz zur mehrstufigen Modelltransformation, der sich als Ziele Langlebigkeit durch plattform-unabhängige Spezifikation von Geschäftslogik, sowie Qualitätsverbesserung durch Entwicklung auf einem höheren Abstraktionsniveau gesetzt hat. In dieser Arbeit soll untersucht werden, wie eine modellgetriebene Verschaltung von Web-Services entlang von Geschäftsprozessen mit Konzepten der Model Driven Architecture umgesetzt werden kann. Hierfür werden nach einer kurzen Einleitung in das Thema zunächst theoretische Grundlagen der Modelltransformation und verwendeter Technologien beschrieben. Insbesondere wird auf die Erweiterung der Unified Modeling Language (UML) durch den Einsatz von Stereotypen, sowie auf die Business Process Execution Language (BPEL) als eine Prozessspezifikationsprache um Web-Service-basierte Geschäftsprozesse zu definieren eingegangen. Um die Praxisfähigkeit der vorgestellten Konzepte unter Beweis zu stellen, wird ein Ansatz der Firma IBM vorgestellt und im Hinblick auf die angestrebten Ziele kritisch bewertet.

## 1 Motivation

Unternehmen sind heute zunehmend abhängig von ihren IT Anwendungen, da die Verknüpfung von Geschäftsprozessen und unterstützenden Anwendungssystemen immer stärker wird. Das oberste Ziel bei der Entwicklung bzw. dem Einsatz von IT Anwendungen sollte es daher sein, die Geschäftsprozesse des Anwenders möglichst gut zu unterstützen. Eine weitere Beobachtung ist, dass sich in Zeiten von Globalisierung und ständigem Kostendruck die Geschäftsprozesse der Unternehmen immer schneller ändern um veränderten Marktbedingungen zu entsprechen bzw. entgegenzutreten. Die schnelle und flexible Anpassung der Prozesse kann einen wesentlichen Wettbewerbsvorteil gegenüber Konkurrenten darstellen und letztlich über den Gesamterfolg eines Unternehmens entscheiden. IT Anwendungen, die nicht in der Lage sind, ebenso flexibel an geänderte Anforderungen des Geschäftsbereichs angepasst zu werden, stellen damit ein großes Hindernis für die Weiterentwicklung von Unternehmen und zudem ein hohes Kostenrisiko dar.

Aus diesen Beobachtungen lassen sich Anforderungen an eine IT Architektur ableiten, die die Entwicklung von Anwendungen, die die genannten Bedürfnisse der Wirtschaft erfüllen, ermöglicht und wesentlich unterstützt:

- Die Geschäftslogik einer Anwendung sollte unabhängig von der verwendeten Technologie spezifiziert werden können um eine Entkopplung zu erreichen. Die Geschäftslogik sollte nicht von Änderungen und Weiterentwicklungen der Technologie beeinflusst werden.
- Die Anwendung sollte sich schnell und flexibel an geänderte Geschäftsprozesse anpassen lassen, insbesondere sollte kein Eingriff in die verwendete Technologie notwendig sein.
- Geschäftsprozesse sollten in einer intuitiven Weise auf einer höheren semantischen Ebene in Form von Modellen spezifiziert werden statt als Programmcode einer Programmiersprache. Man spricht hierbei auch vom „Programmieren im Großen“, dem Konfigurieren und Verschalten von Diensten entlang der Geschäftsprozesse, im Gegensatz zum „Programmieren im Kleinen“, das dem gewohnten Programmieren beispielsweise mit Java entspricht. Dadurch ergäbe sich auch die Möglichkeit, dass nicht Programmierer, sondern die Prozessverantwortlichen selbst die Spezifikation von Abläufen durchführen könnten, was zu erheblichen Kostenreduzierungen und Qualitätsverbesserungen führen könnte.

Ein möglicher Ansatz, der die aufgestellten Anforderungen erfüllen könnte, ist die Model Driven Architecture (MDA). Dabei werden Modelle in mehreren Schritten auf ausführbaren Programmcode abgebildet. Das Konzept der Model Driven Architecture wird im folgenden Kapitel ausführlich erläutert.

## 2 Model Driven Architecture (MDA)

Die Model Driven Architecture (MDA) ist ein Ansatz zur mehrstufigen Modelltransformation um Geschäftslogik in Form von plattform-unabhängigen Modellen auf einer hohen Abstraktionsebene auf ausführbaren Programmcode einer spezifischen Plattform abzubilden. Dafür werden verschiedene Technologien wie die Unified Modeling Language (UML) und die Business Process Execution Language (BPEL) eingesetzt, die im Laufe dieses Kapitels genauer beschrieben werden. Die angesprochene mehrstufige Transformation spiegelt sich in den verschiedenen Abstraktionsebenen der Modelle wider, die jeweils einen bestimmten Aspekt spezifizieren und dadurch eine saubere Trennung der Verantwortlichkeiten und damit einhergehend eine Verminderung der Komplexität von einzelnen Modellen bewirken.

Es wird zwischen folgenden drei Ebenen unterschieden:

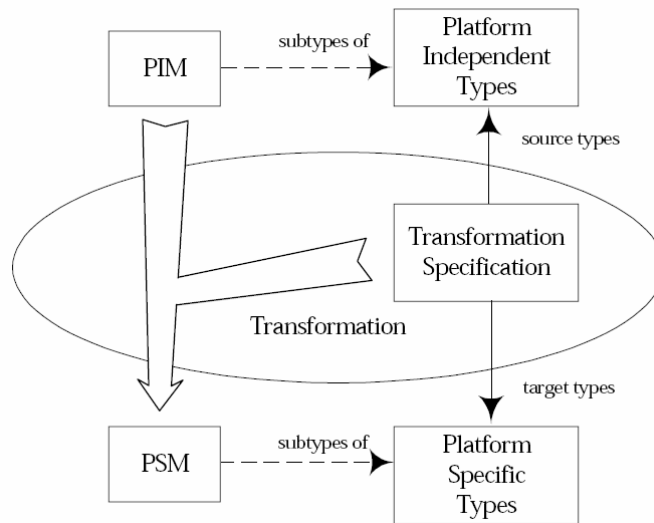
- **Platform Independent Model (PIM)**  
Plattform-unabhängige Spezifikation der Geschäftslogik in Form von Modellen auf einem hohen Abstraktionsgrad
- **Platform Specific Model (PSM)**  
Erweiterte Modelle, die Metainformationen zur Ausführung auf einer bestimmten Plattform enthalten
- **Ausführbare Ablaufspezifikation**  
Durch eine Prozess-Engine direkt ausführbarer Programmcode, der Abläufe eines Geschäftsprozesses sowie dabei aufzurufende Dienste definiert

Mit dem Model Driven Architecture Ansatz werden verschiedene Ziele verbunden. Zum einen erhofft man sich eine Langlebigkeit und Investitionssicherheit durch die plattform-unabhängige Spezifikation von Geschäftslogik mit der eine klare Schnittstelle geschaffen ist um entweder Geschäftsprozesse unabhängig von der Technologie anpassen zu können oder um die einmal definierte Geschäftslogik vor Weiterentwicklungen der Technologie zu schützen. Durch diese gewünschte Entkopplung würde man insbesondere langfristig deutlich an Flexibilität gewinnen. Ein weiteres Ziel das mit dem Einsatz einer Model Driven Architecture verbunden wird ist eine Qualitätsverbesserung von entwickelten Anwendungen aufgrund der Entwicklung auf einem höheren Abstraktionsniveau. Diese ermöglicht nicht nur eine intuitivere Spezifikation von Prozessabläufen und der damit verbundenen höheren Produktivität, sondern auch eine geringere Fehleranfälligkeit bedingt durch die größtenteils automatische Generierung von Programmcode im Gegensatz zum manuellen Programmieren.

Im Folgenden werden die erforderlichen Transformationen der Modelle auf den verschiedenen Abstraktionsebenen vorgestellt sowie benötigte Technologien eingeführt.

### Modell Transformation

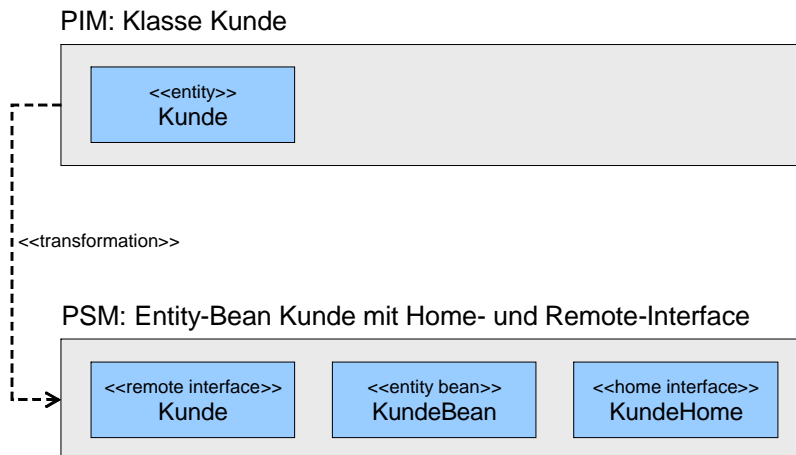
In diesem Abschnitt wird die Transformation von plattform-unabhängigen Modellen (PIM) in plattform-spezifische Modelle (PSM) beschrieben. Die hierbei verwendeten Konzepte werden zunächst theoretisch und anschließend anhand eines konkreten Beispiels der Transformation in die J2EE Plattform dargestellt.



**Abbildung 1: Modell Transformation**

Abbildung 1 zeigt die Transformation eines plattform-unabhängigen Modells in ein plattform-spezifisches Modell [3]. Die Elemente des plattform-unabhängigen Modells bestehen hierbei aus plattform-unabhängigen Datentypen, welche generische Eigenschaften und Verhaltensweisen beschreiben. Diese Elemente sollen auf spezielle, auf einer bestimmten Plattform ausführbare Elemente abgebildet werden, die aus plattform-spezifischen Datentypen bestehen. Die für diese Abbildung notwendigen Informationen befinden sich in der Transformationsspezifikation, die in den Transformationsprozess mit einbezogen wird. Das Ergebnis der Transformation ist ein Modell, das funktional dem plattform-unabhängigen Modell entspricht, allerdings zusätzliche Metainformationen über die Ausführung des Modells auf einer bestimmten Plattform enthält.

An einem Beispiel der Transformation eines plattform-unabhängigen Modells zu einem spezifischen Modell der J2EE Plattform soll das Beschriebene nochmals veranschaulicht werden: (Abbildung 2)



**Abbildung 2: Modell Transformation (J2EE)**

In diesem Beispiel wird ein UML Modell durch zusätzliche Metainformationen erweitert. Hierfür gibt es in UML das Konzept UML Profile. Grundsätzlich kann ein beliebiges UML Modell durch den Einsatz so genannter Stereotypen mit zusätzlicher Semantik versehen werden. Beispielsweise könnte eine Klasse Kunde mit dem Stereotyp `<<entity>>` erweitert werden um darzustellen, dass es sich bei dieser Klasse um ein Datenobjekt, eine Entität, handelt. Der Stereotyp ist in diesem Beispiel sehr generisch, da er nichts darüber aussagt, wie ein Datenobjekt letztlich realisiert wird. Stereotypen können allerdings durchaus auch plattform-spezifische Informationen enthalten, welche die tatsächliche Umsetzung eines Modell-Elements auf einer bestimmten Plattform näher beschreiben. Ein UML Profil (UML Profile) ist eine Sammlung von Stereotypen zur Beschreibung eines bestimmten Anwendungsgebiets. In dem dargestellten Beispiel in Abbildung 2 wird ein UML Profil verwendet, welches spezifische Stereotypen der J2EE Plattform enthält. Hierbei wird eine zunächst plattform-unabhängig spezifizierte Klasse Kunde durch eine Modell Transformation abgebildet auf verschiedene Klassen, die im J2EE Umfeld notwendig sind um eine Entität als EJB (Enterprise Java Bean) Entity Bean zu definieren. Dies sind im Einzelnen das Remote Interface Kunde, das Home Interface KundeHome, sowie die eigentliche Entity Bean KundeBean. Das entstehende Modell besitzt also die gleichen funktionalen Eigenschaften wie das ursprüngliche Modell, enthält darüber hinaus allerdings weitere Metainformationen zur Umsetzung des Modells auf der J2EE Plattform. Weiterhin existiert eine Vielzahl von UML Profilen für diverse andere Plattformen, unter anderem ein Profil zur Spezifikation von BPEL-Konstrukten (siehe nächster Abschnitt).

In diesem Abschnitt wurde die Modell Transformation von plattform-unabhängigen Modellen zu plattform-spezifischen Modellen erläutert. Dies entspricht der Abbildung der obersten beiden Ebenen in der Model Driven Architecture. Der nächste logische Schritt ist nun die Transformation von plattform-spezifischen Modellen in eine ausführbare Prozessspezifikationsprache. Mit der Business Process Execution Language (BPEL) wird nun eine der existierenden Sprachen vorgestellt, die sich zunehmend als Standard zu etablieren scheint.

### **Business Process Execution Language (BPEL)**

Die Business Process Execution Language wurde im Jahr 2003 von den Firmen IBM, BEA und Microsoft entwickelt und ist damit eine sehr junge Sprache, wenngleich viele Erfahrungen der Beteiligten mit bereits vorhanden Prozessspezifikationsprachen wie beispielsweise XLANG von Microsoft in die Entwicklung eingeflossen sind. Mittlerweise ist BPEL auch von der Standardisierungsorganisation Organization for the Advancement of Structured Information Standards (OASIS) standardisiert. Die Sprache existiert momentan in der aktuellen Version 1.1 unter dem Namen BPEL4WS (BPEL for Web Services), eine neue Version 2.0 unter dem Namen WS-BPEL ist in Arbeit.

BPEL ist eine Prozessspezifikationsprache, welche eine Notation und eine Semantik bereitstellt, um Geschäftsprozesse auf Basis von Web-Services auf einer hohen Abstraktionsebene zu beschreiben. Die Notation erfolgt dabei in der eXtensible Markup Language (XML), die durch die Vielzahl an existierenden Hilfsmitteln eine komfortable Verarbeitung der Spezifikation ermöglicht.

Die Ausführung von Prozessen, die mittels BPEL spezifiziert werden, erfolgt durch eine so genannte BPEL-Engine, welche eine Ausführungsumgebung (Runtime Environment) für BPEL Dateien bereitstellt. Eine bereits existierende BPEL-Engine ist BPWS4J (Business Process Execution Language for Web Services Java Run Time) von der Firma IBM [9]. Bis vor kurzer Zeit gab es großen Mangel im Bereich von BPEL Implementierungen. Diese Situation hat sich jedoch innerhalb des letzten Jahres stark verändert, sodass heute eine breite Auswahl von BPEL-Engines verschiedener Hersteller zur Verfügung steht.

Anhand eines Beispiels soll nun der grundsätzliche Aufbau eines BPEL Dokumentes veranschaulicht werden: (Abbildung 3)

```

<?xml version='1.0' encoding="UTF-8"?>
<process name="Beispiel_Person_in_DB_speichern" ...>

  <variables>
    <message name="PersonMessage">
      <part name="firstName" type="xsd:string"/>
      <part name="lastName" type="xsd:string"/>
    </message>
    <variable name="person" messageType="PersonMessage"/>
  </variables>

  <sequence>

    <receive partnerLink="inputPartner" portType="test"
      operation="setPerson" variable="person"/>

    <invoke partnerLink="dbPartner" portType="test"
      operation="savePerson" inputVariable="person"/>

  </sequence>
</process>

```

**Abbildung 3: BPEL Beispiel**

Das Beispiel unterteilt sich zunächst in zwei Bereiche: `<variables>` und `<sequence>`. Im `<variables>`-Element werden Nachrichtenformate, Datentypen und Variablen definiert. Der Typ einer Variablen muss dafür zunächst mittels XML-Schema Definitionen der einzelnen Bestandteile in einem `<message>`-Element festgelegt werden. In dem Beispiel wird eine Variable „person“ definiert, die aus Vor- und Nachname besteht. Der eigentliche Prozessablauf wird im Element `<sequence>` spezifiziert. In dem Beispiel werden zunächst Daten empfangen, d.h. der BPEL Prozess wird aus einer anderen Anwendung heraus aufgerufen. Hierfür existiert das Element `<receive>`, wobei das Attribut „variable“ festlegt, in welche Variable die übermittelten Daten gespeichert werden sollen. Die Attribute „portType“ und „operation“ entsprechen den gleich lautenden Elementen einer WSDL (Web Service Description Language) Datei und bestimmen die Schnittstellen eines Web-Services. Das Attribut „partnerLink“ entspricht der Rolle der aufrufenden Anwendung in der Interaktion des Prozesses mit diesem Partner. Mit Hilfe des `<invoke>`-Elements wird nun im nächsten Schritt der Aufruf eines externen Web-Services angestoßen. Die Attribute „partnerLink“, „portType“ und „operation“ entsprechen denen des `<receive>`-Elements, das Attribut „inputVariable“ spezifiziert die Eingabedaten mit denen der aufgerufene Web-Service parametrisiert wird. In dem Beispiel aus Abbildung 3 wird also der Prozess zunächst selbst aufgerufen, genauer gesagt die Operation `setPerson` und als Parameter wird eine Instanz des Typs `person` übermittelt. Der Prozess stellt sich somit nach Außen selbst als Web-Service dar. Anschließend wird die Operation `savePerson` eines anderen Web-Service aufgerufen mit den zuvor empfangenen Daten als Parameter.

### 3 Modellgetriebene Web-Service Orchestrierung

Unter modellgetriebener Web-Service Orchestrierung versteht man die Komposition und Verschaltung von Web-Services entlang von spezifizierten Geschäftsprozessen auf der Basis von Modellen. Hierfür existieren grundsätzlich zwei unterschiedliche Vorgehensweisen: Ein Top Down Ansatz und ein Bottom Up Ansatz.

Bei einem Top Down Ansatz liegt der Schwerpunkt der Entwicklung auf der Modellierungsumgebung. Ausgangspunkt sind plattform-unabhängige Modelle, die mit zusätzlichen Metainformationen erweitert werden um einerseits einen hohen Abstraktionsgrad zu erhalten und andererseits genügend Detailinformationen zur Spezifikation von ausführbaren Prozessen ausdrücken zu können. Aus diesem Grund sind die entstehenden Modelle auch von vergleichsweise hoher Komplexität, wohlgermt zugunsten einer wesentlich einfacheren Verarbeitung in den darunterliegenden Schichten der Architektur (siehe Architektur des Top Down Ansatzes). Die eigentlichen Transformationen der Modelle erfolgen mehrstufig und entsprechen den Konzepten der Model Driven Architecture.

Bei einem Bottom Up Ansatz liegt der Schwerpunkt im Bereich der Integrierten Entwicklungsumgebung (IDE), von der aus die Komposition und Verschaltung gesteuert wird. Ausgangspunkt sind hierbei bereits vorhandene Web-Services, die mittels der Web Service Description Language (WSDL) beschrieben wurden. Eine Praxislösung, die den Bottom Up Ansatz verfolgt ist ein Vorschlag der Firma Microsoft, der neben Microsoft Produkten wie MS Visio und MS BizTalk Server den Einsatz der proprietären Prozessspezifikationssprache XLANG empfiehlt. Die Orchestrierung kann hierbei durch einfaches „zusammenklicken“ der Web-Services in einer sehr einfach gehaltenen Modellierungssyntax durchgeführt werden. Das dahinter liegende Modell bietet lediglich simple Kontrollstrukturen und Ablaufbeschreibungen und ist in der Ausdrucksmächtigkeit nicht mit Modellen des Top Down Ansatzes vergleichbar.

Der Bottom Up Ansatz wurde an dieser Stelle der Vollständigkeit halber erwähnt, im Folgenden wird allerdings lediglich der Top Down Ansatz vertieft, da nur bei diesem Ansatz versucht wird, die Konzepte der Model Driven Architecture umzusetzen.

### Architektur des Top Down Ansatzes

In diesem Abschnitt wird die dem Top Down Ansatz zugrunde liegende Architektur und die dabei unterschiedenen Entwicklungsebenen genauer beschrieben.

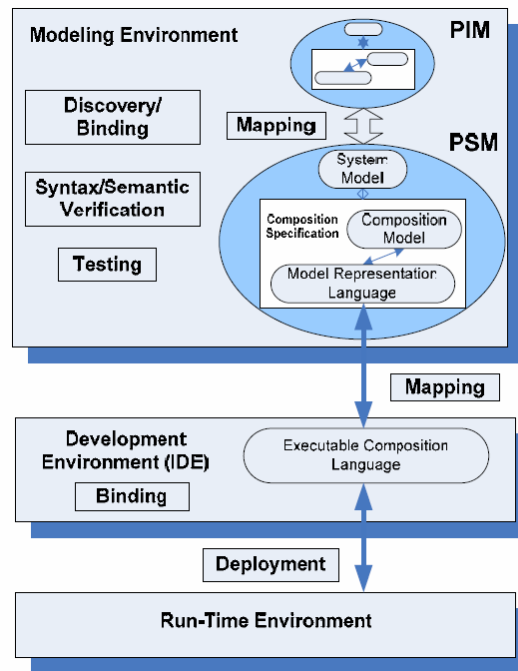


Abbildung 4: Architektur des Top Down Ansatzes

Die Architektur besteht aus drei Abstraktionsebenen: Modellierungsumgebung, Integrierte Entwicklungsumgebung und Ausführungsumgebung. Der Schwerpunkt in dieser Architektur liegt eindeutig auf der Modellierungsumgebung, welche die meisten Aufgaben zu erfüllen hat. Zum einen wird dort das plattform-unabhängige Modell erstellt und auf ein plattform-spezifisches Modell abgebildet, zum anderen werden auf dieser Ebene einige Zusatzfunktionen angeboten, die die Entwicklung erleichtern sollen. Beispielsweise existieren Mechanismen um vorhandene Web-Services zu suchen und zu referenzieren (Discovery/Binding). Außerdem existieren Mechanismen um Syntax und Semantik der erstellten Modelle auf ihre Gültigkeit zu überprüfen (Syntax/Semantic Verification). Schließlich kann der erstellte Entwurf noch getestet werden (Testing).

In der Integrierten Entwicklungsumgebung erfolgt das Referenzieren von konkreten Web-Services (Binding). Außerdem werden die erstellten Modelle in eine ausführbare Prozessspezifikationsprache transformiert.

In der Ausführungsumgebung wird die erstellte Prozessspezifikation schließlich ausgeführt und benötigte referenzierte Web-Services automatisch aufgerufen und entsprechend den Anweisungen miteinander verschaltet.

## IBM Ansatz

Nachdem im vorigen Absatz eine allgemeine Architektur zur Umsetzung des Top Down Ansatzes vorgestellt wurde, wird nun anhand einer Praxis-Lösung der Firma IBM untersucht, inwieweit die Konzepte der Model Driven Architecture zum heutigen Stand von Technologie und Standardisierung zur modellgetriebenen Verschaltung von Web-Services eingesetzt werden können.

Während Abbildung 4 eine statische Sicht auf die Architektur gezeigt hat, wird im Folgenden der dynamische Ablauf der Transformation dargestellt. Abbildung 5 stellt die einzelnen Teilschritte und deren Abhängigkeiten dar, unterteilt in die drei Ebenen Modellierungsumgebung, Integrierte Entwicklungsumgebung und Ausführungsumgebung:

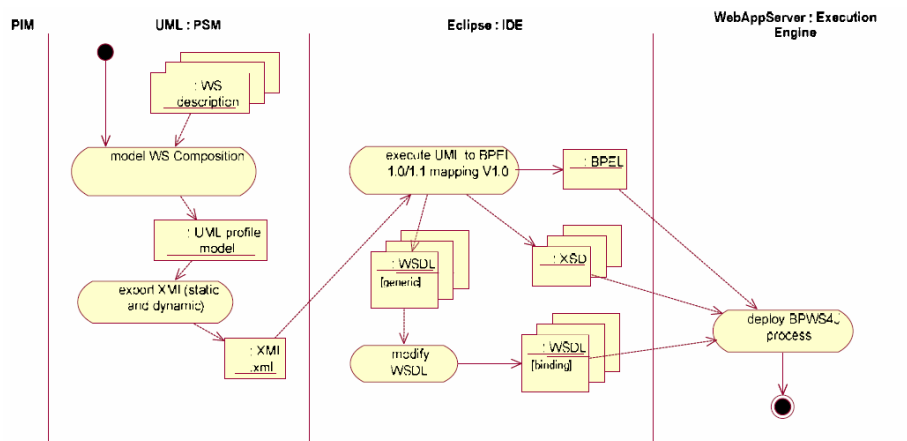


Abbildung 5: Ablauf des IBM Ansatzes

Ausgangspunkt der Transformation ist das plattform-spezifische Modell innerhalb der Modellierungsumgebung. Der von der Model Driven Architecture vorgegebene erste Schritt, die Abbildung von einem plattform-unabhängigen Modell in ein plattform-spezifisches Modell, wird bei diesem Ansatz vorausgesetzt beziehungsweise wird nicht berücksichtigt. Die Lösung von IBM leistet folglich im wesentlichen die modellgetriebene Entwicklung von plattform-spezifischen UML Modellen, welche unter Einsatz von einem UML Profil und darin enthaltenen Stereotypen erweitert werden und der anschließenden Transformation dieser Modelle in die ausführbare Prozessspezifikationsprache BPEL. Das hierbei verwendete UML Profil heißt „UML 1.4 Profile for Automated Business Processes with a mapping to BPEL 1.0“ [4] und ist spezifisch für eine BPEL-basierte Plattform. Bei der Modellierung werden sowohl Klassendiagramme für statische Konstrukte als auch Aktivitätsdiagramme für die Spezifikation des Prozessablaufs erstellt. Die UML Modelle enthalten durch Einsatz der Stereotypen bestimmte Informationen über aufzurufende Web-Services wie beispielsweise bereitgestellte Operationen, Schnittstellen und Nachrichtenformate, die

sich in den verschiedenen Diagrammtypen wieder finden lassen. So wird die statische Sicht auf Datentypen, Nachrichten, Protokolle und Rollen innerhalb der Klassendiagramme dargestellt, während die dynamische Sicht in den Aktivitätsdiagrammen sichtbar wird. Viele dieser Informationen über aufzurufende Web-Services befinden sich bereits in den WSDL Schnittstellenbeschreibungen der Web-Services, sofern diese bereits vorhanden sind. Unglücklicherweise ist in der hier vorgestellten Lösung von IBM kein Import von WSDL Dateien möglich, weshalb viele Informationen manuell in die UML Modelle übertragen werden müssen. Der letzte Schritt innerhalb der Modellierungsumgebung besteht aus dem Export der erstellten UML Modelle in eine Datei im XMI Format (XML Metadata Interchange), welches den Austausch beliebiger Metadaten zwischen unterschiedlichen Anwendungen ermöglicht und bei dieser Lösung genutzt wird, um die erstellten Modelle von der Modellierungsumgebung in die Integrierte Entwicklungsumgebung zu übertragen.

Der IBM Ansatz sieht als Integrierte Entwicklungsumgebung (IDE) die Anwendung Eclipse vor, eine sehr mächtige Java-basierte Open-Source-Entwicklungs-umgebung, die mittels Plugins sehr flexibel erweitert werden kann. Ein von IBM entwickeltes Plugin ist für die eigentliche Transformation des UML Modells in die Prozessspezifikationsprache BPEL zuständig. Dabei wird zunächst die in der Modellierungsumgebung erzeugte XMI Datei importiert um dann im nächsten Schritt den Transformationsprozess anzustoßen. Während der Transformation werden verschiedene Dateien generiert wie WSDL Dateien zur Beschreibung der Web-Service Schnittstellen, XML-Schema Dateien (XSD) zur Definition der verwendeten Datentypen und Nachrichtenformate sowie die eigentliche Prozessspezifikation in Form einer BPEL Datei. Die erzeugten WSDL Dateien sind generisch, d.h. es werden noch keine konkreten Web-Services referenziert, sondern lediglich das Format von ausgetauschten Nachrichten sowie die aufgerufenen Operationen und deren Schnittstellen definiert. Mithilfe eines WSDL Bearbeitungs-Tools innerhalb der Integrierten Entwicklungsumgebung werden dann Bindings zu konkreten Web-Services manuell festgelegt. Bei diesem Schritt existiert keine Unterstützung des Entwicklers durch beispielsweise Suchmöglichkeiten innerhalb eines Verzeichnisdienstes oder gar die automatische Suche und Referenzierung von geeigneten Web-Services. Dies bedeutet unter anderem einen Bruch in der automatisierbaren Abfolge des Transformationsprozesses, da die manuelle Referenzierung von Web-Services mit jeder weiteren Transformation nach einer Änderung des Ausgangsmodells erneut durchgeführt werden muss. Im letzten Schritt wird ein Paket aus den erzeugten Dateien sowie der angepassten WSDL Dateien auf dem Application-Server installiert und kann durch die Java-basierte BPEL-Engine von IBM (BPWS4J: Business Process Execution Language for Web Services Java Run Time) ausgeführt werden. Während der Ausführung werden benötigte Web-Services von der BPEL-Engine aufgerufen und entsprechend der Ablaufspezifikation miteinander verschaltet.

### **Bewertung des IBM Ansatzes**

Nachdem der IBM Ansatz beschrieben wurde soll er nun hinsichtlich der in der Einleitung erarbeiteten Anforderungen kritisch bewertet werden.

Die Vorteile des Ansatzes sind:

- Die Geschäftslogik wird auf einer hohen Abstraktionsebene in Form von Modellen spezifiziert. Dies entspricht der Anforderung nach einer intuitiveren Art der Prozessentwicklung.
- Der Ansatz ermöglicht eine flexible Anpassung von Prozessen, im Idealfall sind lediglich Änderungen auf der Modellebene notwendig.
- Die Ausführung des generierten BPEL-Codes ist nicht auf eine bestimmte Plattform beschränkt. BPEL und Web-Services sind zwar als Technologien festgelegt, die Wahl einer BPEL-Engine und der darunterliegenden Plattform steht jedoch frei.
- Die Verwendung von anerkannten Standards wie UML, XMI und WSDL erhöht das Vertrauen in die Robustheit der Lösung und erhöht zudem die Wahrscheinlichkeit auf Investitionssicherheit.

Wie man sieht sind wesentliche Aspekte der Model Driven Architecture bereits in der Lösung von IBM vorhanden. Allerdings gibt es ebenso wichtige Punkte, die nicht oder noch nicht im geforderten Umfang vorhanden sind.

Die Nachteile des Ansatzes sind:

- Es gibt keine Unterstützung für ein plattform-unabhängiges Modell. Der erste Transformationsschritt der Model Driven Architecture wird übersprungen und das plattform-spezifische Modell wird als Ausgangspunkt angesehen.
- Die Transformation ist nicht vollständig automatisierbar. Insbesondere bei der Referenzierung von konkreten Web-Services, dem Binding, muss die WSDL Datei manuell angepasst werden und verhindert somit eine komplette Automatisierung.
- Die Möglichkeiten zur Verifikation und zum Testen sind beschränkt bis nicht vorhanden. Tritt an beliebiger Stelle ein Fehler auf, wird die Transformation ohne entsprechende Fehlermeldung abgebrochen.

Insgesamt ist der Ansatz von IBM viel versprechend und es bleibt abzuwarten ob die noch vorhandenen Probleme in nächster Zeit gelöst werden.

## 4 Zusammenfassung und Ausblick

In dieser Arbeit wurde untersucht, wie eine modellgetriebene Verschaltung von Web-Services entlang von Geschäftsprozessen mit Konzepten der Model Driven Architecture (MDA) umgesetzt werden kann. Dabei wurden zunächst theoretische Grundlagen der Modelltransformation und verwendeter Technologien beschrieben und anschließend ein praxisorientierter Ansatz der Firma IBM vorgestellt und kritisch bewertet.

Es bestehen noch einige offene Fragen im Hinblick auf Standardisierung: Zum einen ist die Frage, ob sich BPEL als Prozessbeschreibungssprache durchsetzt, da durchaus qualitativ gleichwertige Alternativen wie beispielsweise WSCI/BPML existieren. Momentan sieht es jedoch nicht zuletzt durch die große Unterstützung durch namhafte Firmen wie IBM, BEA und Microsoft so aus, als würde sich BPEL als einzige Sprache in diesem Bereich durchsetzen. Zum anderen ist offen, inwieweit sich verwendete Technologien wie UML, XMI oder WSDL, die sich für sich betrachtet bewährt haben, auch im gemeinsamen Zusammenspiel etablieren. Beispielsweise gibt es auf Modellierungsseite einige diskutierte Alternativen zu UML wie eEPK (erweiterte Ereignisgesteuerte Prozessketten) oder BPMN (Business Process Modeling Notation). Diese Frage wird nicht zuletzt auch von Modellierungswerkzeug-Herstellern mitentschieden, da nur eine breite Verfügbarkeit in vorhandenen Werkzeugen die Etablierung einer Modellierungssprache ermöglicht.

Viel Potenzial steckt in Verfahren zum automatischen Auffinden und Referenzieren von Web-Services mit Hilfe von Verzeichnisdiensten wie UDDI (Universal Description, Discovery and Integration). So könnten Dienste anhand definierter Kriterien automatisch bewertet und ausgewählt werden. In diesem Bereich existieren allerdings noch viele ungelöste Probleme, beispielsweise reicht eine Schnittstellen-Definition auf rein syntaktischer Ebene oftmals nicht aus um eine automatische Integration von Diensten zu ermöglichen, d.h. es fehlt eine formale Spezifikation der Semantik von Operationen.

Eine Voraussetzung zum produktiven Arbeiten mit den vorgestellten Methoden ist die Möglichkeit der Verifikation der Modelle auf allen Ebenen und das Bereitstellen aussagekräftiger Fehlermeldungen. Dieser Bereich ist nach wie vor Gegenstand intensiver Forschungsarbeiten.

## Literatur

- [1] K. Pfadenhauer, S. Dustdar, B. Kittl: Comparison of Two Distinctive Model Driven Web Service Orchestration Proposals, IEEE CECW'05, 2005
- [2] K. Mantell: From UML to BPEL, IBM developerworks, 2003,  
<http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/>
- [3] J. Miller, J. Mukerji: MDA Guide Version 1.0.1, Object Management Group, 2003
- [4] J. Aamsden, T. Gardner, et. al.: Draft UML 1.4 Profile for Automated Business Processes with a mapping to BPEL 1.0, IBM, 2003
- [5] K. Pfadenhauer, S. Dustdar, B. Kittl: Challenges and Solutions for Model Driven Web Service Composition
- [6] T. Andrews, F. Curbera, et. al.: Business Process Execution Language for Web Services Version 1.1, BEA, IBM, Microsoft, SAP, Siebel, 2003
- [7] A. Dreiling, M. Rosemann, et. al.: Model-Driven Process Configuration of Enterprise Systems, 2004
- [8] D. Skogan, R. Gronmo, I. Solheim: Web Service Composition in UML, IEEE EDOC, Moterey, 2004
- [9] IBM Developerworks: BPWS  
<http://www.alphaworks.ibm.com/tech/bpws4j>
- [10] E. Breton, J. Bézivin: Model Driven Process Engineering, IEEE, 2001