

Kapitel 2

Grundlagen

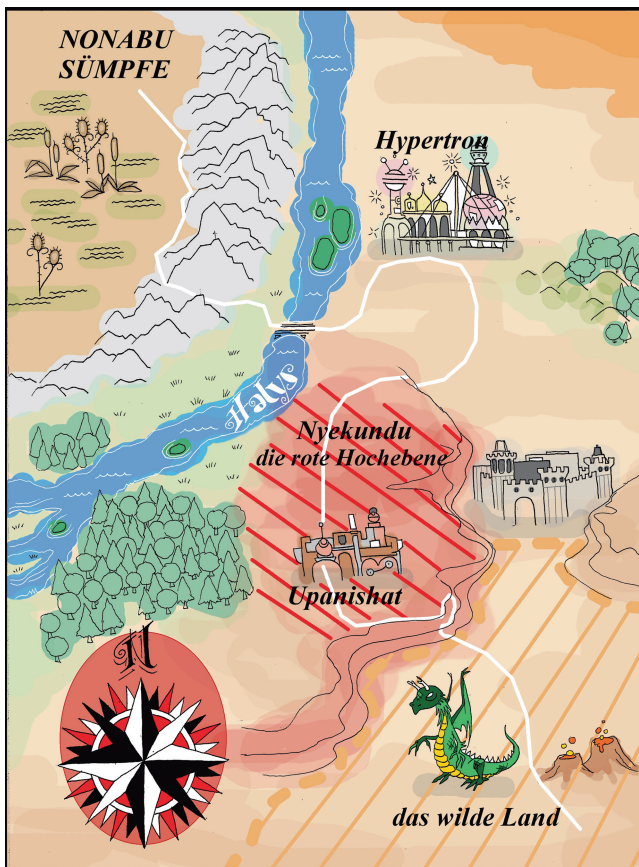


Abb. 2.1 Seans Reisekarte

unter Mitwirkung von Thomas Henninger

Sean war während der Überquerung von Babschinar schon früh auf seinen ersten Meister getroffen. Dieser hatte ihm gezeigt, wie er Aufgaben mit einem sehr begrenzten Satz von Befehlen effizient aber auch sehr mühsam an Maschinen übertragen konnte. Diese Fähigkeit war sehr beliebt bei den Ingenieuren, die daraus größere und komplexere Maschinen bauten.

Nach kurzer Zeit machte er sich weiter auf den Weg, von dem Gedanken beseelt, dass es doch noch weitere Möglichkeiten geben müsste. Es musste doch einfacher und schneller gehen, Programme zu erstellen. Und tatsächlich traf er auf weitere Meister und so lernte er die verschiedensten Sprachen kennen: streng formale, strukturierte, die mit strengen Reihenfolgen und Baumstrukturen agierten. Sprachen mit mächtigem Befehlssatz und welche mit sehr kleinem. Sprachen, die in realen Objekten „dachten“, Sprachen die auf Ereignisse reagierten und es gab sogar welche, die sich selbst erweitern konnten!

Sean lernte schnell und er hatte gute Lehrmeister. Er lernte auch, wie die verschiedenen Konzepte miteinander in Beziehung standen, wie sie miteinander verbunden werden konnten. Er stellte aber auch schnell fest, dass so mancher Meister in seiner Welt fest eingeschlossen war. Nicht mit jedem konnte er über die Vorzüge oder Nachteile einer Sprache sprechen. Auch hatte er es fast immer mit sehr begrenzten Aufgaben zu tun und das große Ganze hatten seine Meister oft nicht im Blick. Manchmal war das auch nicht erwünscht!

Aber Sean wollte mehr. Er wollte verstehen, wie alles miteinander zusammenhing. Er wollte sehen, was all die Menschen, die das Wissen seiner Meister verwendeten damit bauten und warum sie das taten!

Sein letzter Meister seufzte als Sean wieder einmal eine Diskussion mit ihm begann. „Weißt du Sean, du bist ein guter Schüler. Aber deine Neugierde und dein Streben nach Wissen geht weit über das hinaus, was ich dir beibringen kann! Ich lasse dich ungern ziehen, aber in Babschinar gibt es nicht das, nach dem du strebst. Zieh weiter mein Junge, möge das Glück mit dir sein und möge Tagos, der Weise Dich beschützen!“

Und so stattete ihn sein Meister großzügig mit allerhand Proviant aus und verabschiedete ihn mit einem Festessen. Sean machte sich auf den Weg, um am Fuße der Ostseite von Babschinar eine Furt durch den großen Grenzfluss Halys zu finden.

MDA & MDSD

MDA steht für Model Driven Architecture und ist ein Standard der OMG.¹ Die OMG ist ein unabhängiges Standardisierungsgremium. Standards wie CORBA, MOF, OCL, UML, SysML, XMI und nicht zuletzt die BPMN werden von der OMG verwaltet.

¹Object Management Group <http://www.omg.org/> (04.11.2010).

MDSD steht für Model Driven Software Development. Das lässt sich am ehesten mit „modellgetriebene Software Entwicklung“ übersetzen. Es gibt weitere synonyme Begriffe, wie zum Beispiel MDD für Model Driven Development oder MDE für Model Driven Engineering.²

Nun, was haben die beiden auf den ersten Blick sehr ähnlichen Begriffe miteinander zu tun? Sie teilen zunächst das Grundsätzliche: Software wird in plattformunabhängigen, formalen Modellen entwickelt und automatisiert generiert.

Die Unterschiede liegen im Detail.

Model Driven Architecture (MDA) der OMG

Die Model Driven Architecture [MDA] ist der Entwurf eines Standards der OMG für die modellbasierte Softwareentwicklung. Er umfasst alle Schritte der Softwareentwicklung von Analyse und Design über Implementierung und Test bis zur Integration und Wartung. Die MDA wurde mit dem Ziel entwickelt, die Plattformunabhängigkeit, Interoperabilität, und Wiederverwendbarkeit einer Vorgehensweise und der Zwischenprodukte des modellgetriebenen Entwicklungsansatzes zu gewährleisten.

Eine sehr kurze Geschichte der MDA

Im Jahr 2000 wurde innerhalb der OMG ein erstes Dokument mit dem Titel „Model Driven Architecture“ als Grundlage für die weitere Entwicklung zum Standard begutachtet. Vom sogenannten Drafting Team des OMG Architecture Board wurde die weitere formale Ausarbeitung 2001 im Dokument „Model Driven Architecture – A Technical Perspective“ als Whitepaper fixiert. Dieses Dokument wurde von den Mitgliedern als Basis-Architektur akzeptiert und von verschiedenen Task-Forces verwendet, um darauf aufbauend Standards für verschiedene Middleware Plattformen zu definieren. 2003 wurde „MDA Guide Version 1.0.1“ von der OMG veröffentlicht, welche als offizielle gültige Version gilt.

Ziele der MDA

Das Hauptziel der MDA ist die Trennung der geforderten Funktionalität eines zu entwickelnden Systems von der zur Implementierung dieses Systems herangezogenen Technik. Dies ermöglicht eine Abstrahierung der Systembeschreibung durch die Aufteilung in verschiedene Ebenen, die sich immer näher zur Plattform³ hin

²Es gibt noch weitere Ansätze, wie zum Beispiel das Feature Driven Development [FDD] oder Domain Driven Design [DD1] [DD2].

³Der Begriff Plattform ist hier weit gefasst. Eine Plattform kann sich aus den verschiedensten Komponenten zusammensetzen, wie zum Beispiel Implementierungssprachen, Frameworks, Applikationsserver, Rule-Engines und vieles andere mehr.

entwickeln. Die Modelle auf höherer Abstraktionsebene sind stabiler, da sie unabhängig von der konkreten Plattform weiter verwendbar bleiben, auch wenn die Plattform gewechselt wird.

Durch Modelltransformationen wird von einer abstrakten fachlichen Ebene über mehrere Stufen und durch schrittweise Anreicherung von Informationen in den jeweils folgenden Ebenen letztendlich plattformspezifischer Quellcode erzeugt. Das bedeutet, dass beim Wechsel einer Plattform nur die Transformationen der jeweiligen Ebenen ausgetauscht werden müssen, ohne dass man gezwungen wäre, die für die darüber gelegenen Ebenen entwickelten Transformationen anzupassen.

Dadurch erhofft man sich eine kürzere Entwicklungszeit mit gleichzeitig signifikant höherer Softwarequalität als beim durchgängig manuellen Ansatz der Softwareentwicklung. Außerdem soll die Wiederverwendbarkeit erhöht werden.

Vorgehen und Ebenen der MDA

Als Grundlage für die Modellierung in der MDA dient die MOF⁴ beziehungsweise die UML⁵ [UML]. Die Herangehensweise während der Entwicklung durchläuft mehrere Phasen. Zunächst erfolgt eine formale Beschreibung des Systems. Dann folgt die Spezifikation der Details der Plattform. Schließlich wird die formale Beschreibung durch eine Transformation in die für die Plattform notwendige Spezifikation überführt. Eine Spezifikation kann beispielsweise ein anderes Modell oder Quellcode sein.

Die MDA untergliedert den Entwicklungsprozess in vier Ebenen. Jede der Ebenen verfeinert die vorausgehende und führt somit zu einer Konkretisierung des Softwaresystems. Die Spezifikationen der einzelnen Schichten werden ebenfalls durch Transformationen ineinander umgewandelt. Tabelle 2.1 gibt eine Übersicht über die vier Ebenen der MDA.

Tabelle 2.1 MDA Ebenen

| Ebene | Gegenstandsbereich |
|-------------------------------------|--|
| Computation Independent Model (CIM) | Umgebung, in die das System eingebettet ist, Domäne, Geschäftsmodell |
| Plattform Independent Model (PIM) | Generelle Funktionsweise des Systems ohne Plattformdetails |
| Plattform Specific Model (PSM) | PIM erweitert um Plattformdetails |
| Implementation Specific Model (ISM) | Quelltext |

⁴Meta Object Facility [MOF].

⁵Die ist zwar nicht zwingend vorgeschrieben, aber faktisch gibt es nur Beispiele, welche die UML benutzen. „Use of UML, although common, is not a requirement; MOF is the mandatory modeling foundation for MDA.“. Aber eben auch: „Models used with MDA can be expressed using the UML language“.

Obwohl im Standard vorgesehen, sind automatische Transformationen heute zwar grundsätzlich spezifiziert und mit QVT⁶ existiert sogar ein eigener Standard der OMG. In der Breite durchgesetzt hat sich aber noch keiner der Ansätze. Und so wird meist vom Hersteller eines Modellierungswerkzeugs eine proprietäre Lösung für Modelltransformationen angeboten, beziehungsweise mitgeliefert. Durch Transformationen lassen sich die Strukturen der übergeordneten Ebene übertragen. Die ergänzenden Feinheiten der Plattform sind jedoch noch nicht vollständig automatisiert generierbar. Es sind daher manuelle Verfeinerungen notwendig.

Grundlage für die Transformationen ist das sogenannte Mapping. Das Mapping beschreibt die Abbildungsregeln, das heißt konkreter: welche Elemente des Quellmodells werden auf welche Elemente des Zielmodells abgebildet. Zum Beispiel könnte eine UML-Action, die einen Schritt in einem Geschäftsprozess repräsentiert in der nächsten Ebene auf einen Anwendungsfall abgebildet werden. Der Anwendungsfall wird dann weiter ausformuliert, zum Beispiel unter der Verwendung eines Aktivitätsdiagramms.

Die Transformationsregeln werden meistens auf Ebene der Metamodelle, zum Beispiel UML-Profile oder auch seltener auf MetaMeta-Ebene, zum Beispiel der MOF, beschrieben. Dabei kann noch zwischen uni- und bidirektionalen Transformationen unterschieden werden. Ein Forward-Engineering beschränkt sich dabei meist auf unidirektionale Transformationen, vom abstrakten Modell schrittweise zum plattformspezifischen Code. Round Trip-Engineering nutzt, zumindest auf der Designebene, bidirektionale Transformationen.

Gehen wir noch etwas genauer auf die verschiedenen Ebenen der MDA ein.

Computation Independent Model (CIM)

Das CIM enthält die Anforderungen an das zu implementierende System.⁷ Oft wird es auch als „Domänen Modell“ oder „Business Modell“ bezeichnet. Es ist völlig unabhängig davon, wie das System implementiert wird und enthält keine implementierungsspezifischen Informationen.

Ziel ist es, die Umgebung des Systems und den Kontext der Nutzung zu beschreiben. Damit hilft es klarzustellen, was ein System tun soll. Darüber hinaus ist es auch eine Quelle für die Standardisierung von Begriffen und der Bildung von Glossaren, die dann auch system- und projektübergreifend genutzt werden können.

Die im CIM modellierten Anforderungen sollten außerdem eine navigierbare Beziehung (Trace) zu den durch Transformation entstandenen Inhalten der darunterliegenden Ebenen im plattformunabhängigen Modell (PIM) und plattformspezifischen Modell (PSM) enthalten.

⁶Query View Transformation [QVT].

⁷Ein System kann in diesem Sinne auch ein Subsystem sein oder sich aus solchen zusammensetzen.

Ein CIM kann aus mehreren Modellen bestehen.⁸ Jedes der Modelle kann dabei eine ganz spezifische Sicht einnehmen. Zum Beispiel einerseits den Informationsfluss, die Geschäftsprozesse im Kontext des Systems und ergänzend dazu die organisatorische Sicht.

Plattform Independent Model (PIM)

Das plattformunabhängige Modell beschreibt das System, aber nicht die technischen Details der zur Implementierung herangezogenen Technik. Es enthält verschiedene Sichten auf das zu implementierende System, zum Beispiel auf die Datenstrukturen und deren Verwendung, Abläufe auf dem System und ergänzend Regeln und Berechnungen, die das System zu beachten oder auszuführen hat. Dabei werden Informationen des CIM durch eine Modelltransformation übernommen. Das PIM kann dabei anfänglich auch vollständig aus dem CIM erzeugt werden.

Die Sicht auf die Daten und die Darstellung der Daten kann dabei unterschiedlich zur Darstellung im CIM und zum nachfolgenden PSM sein. Aus den Informationen des PIM lassen sich über Modelltransformationen plattformspezifische Modelle erzeugen, die dann weiter angereichert werden. Auch hier gilt, dass die entstandenen Elemente mit ihren Vorgängern im Quellmodell verknüpft sein sollten.

Plattform Specific Model (PSM)

Das plattformspezifische Modell enthält die Beschreibung des Systems unter Einbeziehung der verwendeten Plattform. Die Sicht auf die Daten erlaubt meist die Überführung in eine Datenbank und der für konsistente Zugriffe notwendigen Persistenzschicht. Das PSM bildet dazu oft auch die Architektur des zu erzeugenden Systems ab und hält sich an deren Regeln. Das PSM ist damit nicht nur ein „Werkzeug“ für den Entwickler, sondern vor allem auch für den Architekten.

Je detaillierter die Modellierung im PSM ist, desto näher rückt das PSM an die Ausführbarkeit, beziehungsweise ist dann eine andere Sicht auf den Quellcode des implementierten Systems. Daraus folgt, dass der Grad der Detaillierung auch den Grad der Ausführbarkeit des aus dem Modell erzeugten Codes bestimmt. Dazu verwendet es die UML oder eine Kombination aus UML und OCL⁹ und wird in einem MOF-basierenden Repository gespeichert. 100% Ausführbarkeit ist dabei aber nicht immer das Ziel!

Das Modell kann neben der reinen statischen und dynamischen Sicht auf das System auch weitergehende Informationen, wie zum Beispiel Deployment- und Konfigurationsinformationen enthalten.

⁸Es wird nicht immer streng zwischen den Begriffen „Modell“ und „Diagramm“ unterschieden. Für MDSOA gilt, dass innerhalb einer Phase die Sichten mit unterschiedlichen Diagrammen dargestellt werden. Die Phase selbst wird als eigenständiges Modell begriffen.

⁹Object Constraint Language [OCL].

Implementation Specific Model (ISM)

Das implementierungsspezifische Modell ist der Code des Systems. Es ist auf der Plattform ausführbar. Das ISM kann, bei entsprechender Modellierung im PSM, bis 100% aus dem PSM erzeugt werden. Meist liegt der Anteil aber unter 100%.

Transformationen

Grundsätzlich geht die MDA davon aus, dass von Ebene zu Ebene mindestens eine Modelltransformation durchgeführt wird. Die Abbildungsregeln für die Transformation werden im Mapping festgelegt. Es werden also Inhalte eines Modells aus der vorhergehenden Ebene in die nächste Ebene übernommen und dabei transformiert. Die MDA erlaubt aber auch, dass zwischen den einzelnen Ebenen mehrere Mappings verwendet werden. So kann ausgehend vom PIM evolutionär über mehrere Mappings und Transformationen in unterschiedliche Modelle die Ebene PSM erreicht werden.

Es ist auch erlaubt, zusätzlich Informationen für die Transformation zu nutzen oder auch weitere Modelle hinzuzuziehen (Merge).

Damit eignet sich die MDA sehr gut für das agile, iterative Vorgehen, da die Modelle in jeder Iteration wiederverwendet werden können und den Zielen der jeweils aktuellen Iteration entsprechend angepasst und erweitert werden.

Vor der Ausführung einer Transformation wird ein Modell „markiert“. Was ist darunter zu verstehen?

Markieren des Modells

Durch Markieren einzelner Elemente in Modell, werden sie in Bezug zu einem bestimmten Mapping gesetzt. Denkt man sich pro Mapping auch eine Transformation, kann ein Element mehrfach markiert werden. Damit entstehen aus dem so markierten Element im Zielmodell für jedes Mapping eines oder mehrere Elemente. Dies kann auch interaktiv während der Ausführung einer Transaktion vorgenommen werden. Das heißt, der Anwender wird gefragt, welche Regeln (Mapping) auf die gewählten Elemente angewendet werden sollen.

In der Praxis wird das Markieren aus Effizienzgründen oft in den Transformationen hinterlegt. Das bedeutet, dass für eine gewählte Menge von Elementen im Zielmodell innerhalb einer Transformation tatsächlich mehrere Transformationen durchlaufen werden. Aus einem Geschäftsobjekt werden zum Beispiel eine Klasse und gleichzeitig auch ein Glossareintrag erzeugt. Die Schritte werden zwar nacheinander durchgeführt, für den Anwender geschieht dies aber transparent. Aus seiner Sicht wird eine Transformation gestartet und ausgeführt.

Ablauf innerhalb der MDA

Abbildung 2.2 stellt den Ablauf exemplarisch dar. Das Quellmodell (Modell Ebene₀) ist eine Instanz seines Meta-Modells (Meta-Modell Ebene₀). Das Zielmodell (Modell Ebene₁) ist wiederum eine Instanz seines Meta-Modells (Meta-Modell Ebene₁).

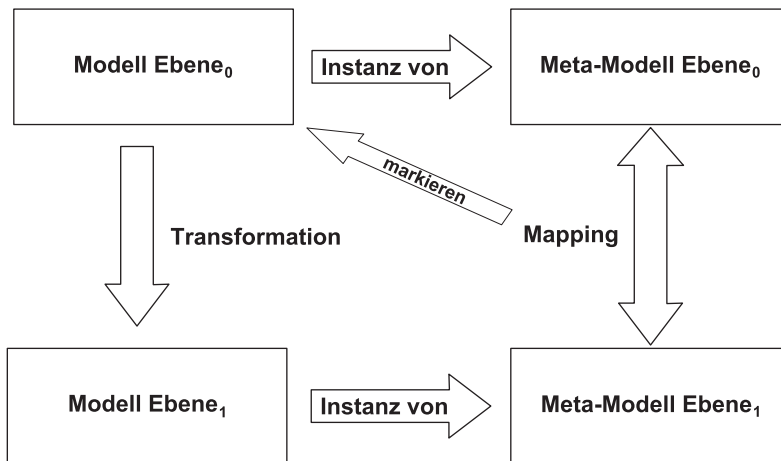


Abb. 2.2 Vorgehen in der MDA

Das Mapping wird auf Basis der Meta-Modelle definiert. Es beschreibt also, welche Typen aus Meta-Modell Ebene₀ auf welche Typen in Meta-Modell Ebene₁ abgebildet werden. Die Transformation wird auf Basis der im Mapping definierten Abbildungsregeln implementiert.

Die zu transformierenden Elemente im Modell Ebene₀ werden für die gewünschten Mappings markiert und danach wird die Transformation ausgeführt.

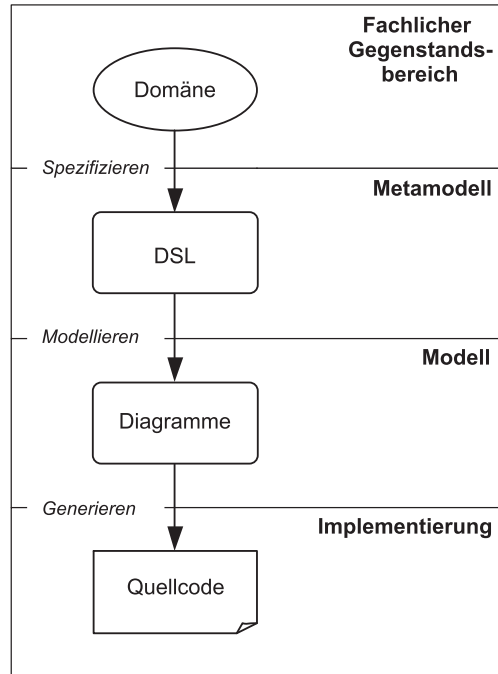
Modelbasierte Softwareentwicklung (MDSD)

Die Modellbasierte Softwareentwicklung (MDSE, engl. Model Driven Software Development, MDSD) hat die gleichen Ziele wie die MDA, ist aber grundsätzlich nicht auf MOF/UML eingeschränkt. Unter einem Modell ist auch bei der MDSD die abstrahierte Abbildung eines Bereichs der Wirklichkeit in graphischen Notationen oder formalen Sprachen zu verstehen. Und auch hier findet eine Trennung der Fachlichkeit von der technischen Umsetzung statt („Separation of Concerns“).

Der fachliche Gegenstandsbereich wird auch mit dem Begriff „Domäne“ bezeichnet. Es handelt sich dabei um ein begrenztes Interessen- und Wissensgebiet technischer oder fachlicher Natur. Dies können beispielsweise die spezifischen Datenobjekte (Geschäftsobjekte) und die fachlichen Abläufe einer Abteilung sein. Die Vorgehensweise bei der modellbasierten Entwicklung illustriert Abb. 2.3.

Die Vorgehensweise bei der modellbasierten Entwicklung startet damit, in einer formalisierten Beschreibung einer Domäne festzulegen, welche Elemente und Beziehungen in einem Modell vorkommen können. Diese Beschreibung bildet das Metamodell für die Modellierung. Es umfasst die abstrakte Syntax und die statische Semantik (Constraints) dessen, was in einem Modell darstellbar ist. Als „Constraints“ werden Bedingungen bezeichnet, welche die Validität eines Modells bestimmen. Das gilt so auch für die MDA, da diese, wie gesagt, eine spezielle Ausprägung des MDSD ist.

Abb. 2.3 Vorgehensweise Modellbasierte Softwareentwicklung



Das Metamodell definiert somit eine formale Sprache, die spezifisch für die Anforderungen des fachlichen Gegenstandsbereiches ausgelegt ist. Diese Art von Sprache wird folglich als domänenspezifische Sprache¹⁰ bezeichnet [STA07]. In der Praxis kann dies zum Beispiel, wie in der MDA, durch Anpassung des UML-Metamodells erzielt werden. Es gibt für MDSD aber keinen definierten Standard, der Einsatz der UML ist aber nicht unüblich. Die Definition eines Metamodells ist bei Verwendung standardisierter Modellierungssprachen optional. Implizit liegt für die gewählte Modellierungssprache ein Metamodell schon vor.

Auf die Spezifikation des Metamodells folgt mit der Modellierung die Abbildung des fachlichen Gegenstandsbereiches und der fachlichen Anforderungen in einem Modell. Die fachlichen Modelle werden im Folgenden um die Details der Plattform erweitert. Man spricht auch von Verfeinerung. Aus dem angereicherten Modell wird im weiteren Verlauf lauffähige Software generiert. Als Generierung wird die automatisierte Erzeugung von Software oder Quellcode bezeichnet.

Das hört sich zunächst sehr ähnlich zur MDA an. Wo genau liegt nun der Unterschied? Nun, in der Wahl der Mittel ist die MDSD „freier“ als MDA, die sich auf eine MOF basierende Sprachen beschränkt. Man kann auch sagen, dass die MDA eine spezielle Ausprägung von MDSD ist, aber nicht umgekehrt.

Um eine DSL zu entwickeln, lässt die MDSD alle Mittel zu. Damit kann zum Beispiel in der Versicherungswelt in einem Modell der „KFZ-Versicherungsvertrag“ mit einem eigenen Symbol dargestellt werden. Auch

¹⁰Domain Specific Language, DSL.

können im Metamodell direkt entsprechende Eigenschaften festgelegt werden. Ein „KFZ-Schadensfall“ kann ebenfalls sein eigenes „Aussehen“ haben.

Es werden also für die Domäne der „KFZ-Versicherung“ spezifische Elemente, deren erlaubten Beziehungen, sowie deren Symbolik definiert. Ein Fachexperte der Domäne „KFZ-Versicherung“ tut sich in der Regel leichter damit, in und mit seiner Begriffswelt Modelle zu erstellen. Abbildung 2.4 zeigt ein einfaches Beispiel einer graphischen DSL für die genannte Domäne.

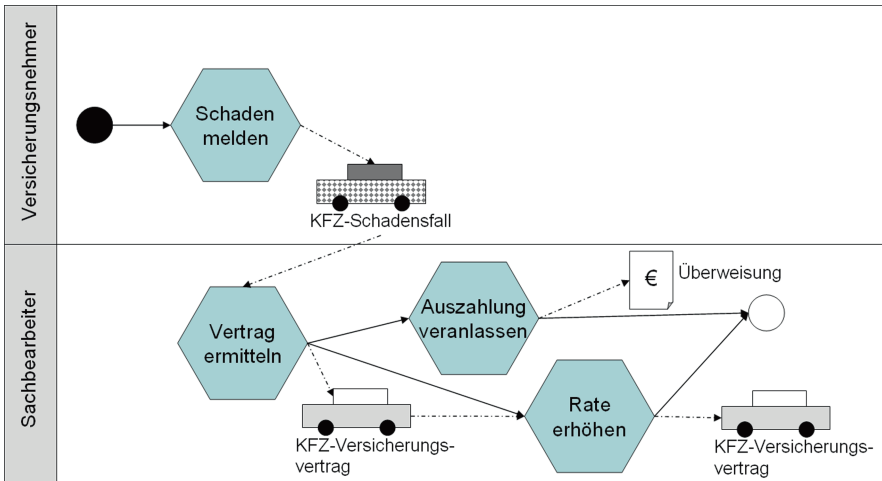


Abb. 2.4 Beispiel einer grafischen DSL für die Domäne KFZ-Versicherung

Wenn Metamodelle nicht auf Standards wie MOF oder der Erweiterung von UML-Profilen aufbauen,¹¹ ist die Kehrseite allerdings, dass sie schwerer Wiederverwendbar sind. Das muss aber nicht unbedingt als Nachteil zum tragen kommen, solange man sich in der Domäne und innerhalb des Unternehmens, in dem die DSL eingesetzt wird, befindet. Möchte man sich mit Partnern oder über die Grenze der Domäne austauschen, stößt man ohne standardisiertes Metamodell schnell an Grenzen. Ein Austausch der Metamodelle über Toolgrenzen hinweg kann sich dann durchaus als schwierig und aufwendig herausstellen.

Aus diesem Grund gibt es oft eine Mischung aus standardisierten Notationen, wie zum Beispiel der UML und BPMN, und anderen Sprachen oder Darstellungsarten. Feature-Modelle, Baumstrukturen, XML-Dialekte, Entscheidungstabellen oder die direkte Nutzung einer Programmiersprache wie Ruby oder Java zur Entwicklung einer DSL können in der MDSD genutzt und gemischt werden. Bei der MDSD steht

¹¹EMF ist ein weiterer Quasi-Standard der weit verbreitet ist und auf dessen Basis sich auch graphische Editoren zur Modellierung der mit EMF beschriebene DSL erstellen lassen. Ausführliche Informationen enthält neben [EMF] auch [EMF2].

die Erreichung des Ziels durch effiziente Modellierungen mit einer DSL,¹² Wiederverwendbarkeit und hohe Codequalität im Vordergrund und vor der Verwendung von Standards.

Die MDSD lässt so für die Entwicklung von Produktlinien, die 100% lauffähigen Code erzeugen und die für Produktvarianten genutzt werden sollen, die notwendigen Freiheiten bei der Wahl der Notationen. Es muss eben nicht ausschließlich die UML verwendet werden. Das liegt nicht darin begründet, dass sich mit der graphischen Notation der UML das Ziel nicht erreichen ließe, sondern vielmehr daran, dass für bestimmte Beschreibungen alternative Notationen, wie zum Beispiel Featuremodelle, effizienter sind. Abbildung 2.5 zeigt ein einfaches Featuremodell.

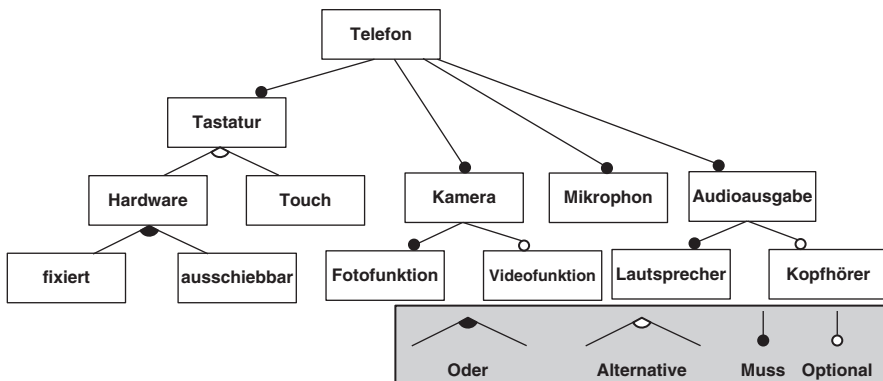


Abb. 2.5 Einfaches Feature-Modell für ein mobiles Telefon

Produktstrassen lassen sich also oft besser mit einer Mischung verschiedener Standards, Notationen, Tools und Metamodellen bauen. So manches lässt sich über XML oder in einer Programmiersprache schneller beschreiben als in einer graphischen Notation. Man darf dabei aber nicht die Komplexität der Gesamtheit aller eingesetzten Tools und Sprachen außer Acht lassen. Es ist immer sinnvoll sich selbst eine gewisse Sparsamkeit aufzuerlegen, was deren Verwendung angeht. Brüche in einer Toolkette bedeuten auch immer Aufwand an den Schnittstellen und eventuell notwendige Transformationen.

Klar dürfte auch sein, dass eine graphische Notation von einem Fachexperten meist einfacher anzuwenden ist. DSLs, die zum Beispiel direkt in XML oder einer Programmiersprache hinterlegt sind, werden eher von Ingenieuren oder Softwareexperten verwendet werden, als zum Beispiel von einem Sachbearbeiter in der Schadensfallbearbeitung einer Versicherung.

¹²Modellierung bedeutet dabei nicht zwingend die Verwendung einer graphischen Notation. In der MDSD kann eben auch textuell modelliert werden, zum Beispiel mit XML und entsprechend zugehörigen validierbaren Schemata. Auch in der MDA ist Quellcode die textuelle Repräsentation eines Modells.

Damit sind wir bei der Frage, was sich nun besser eignet. Dazu können wir leider auch nur die typische Beraterantwort geben: Es kommt darauf an.

- Wer soll die DSL einsetzen?
- Was ist das Ziel der Modellierung, der DSL, zum Beispiel Dokumentation oder Generierung?
- Was ist im Unternehmen bekannt und etabliert, was hat gut funktioniert, was ist akzeptiert?
- Gibt es die Notwendigkeit, Modelle mit Partnern, Lieferanten, Herstellern, etc. auszutauschen?
- Möchte ich mir sicher sein, dass ich am Markt viele Ressourcen mit entsprechendem Know-how bekomme, wenn mein Lieferant nicht mehr verfügbar ist?
- Gibt es in meiner Domäne eventuell schon etablierte Standards, die ich nutzen kann?

Letzten Endes muss jeder für sich selbst die Möglichkeiten prüfen und abwägen, mit welchen Mitteln die Ziele unter den gegebenen Randbedingungen am besten erreicht werden können.

Modellierungssprachen

In MDSOA werden hauptsächlich die BPMN 2.0 und die UML2 eingesetzt. Unabhängig davon möchten wir ihnen im Folgenden eine kurze Einführung in die verschiedenen Notationen geben, ohne dabei allzu sehr in die Tiefe zu gehen. Die Abhandlung geht so weit, dass die Verwendung der Notationen in den nachfolgenden Kapiteln verständlich ist. Für die Leser, die ihr Wissen um die einzelnen Notationen weiter vertiefen möchten, haben wir an geeigneter Stelle auf entsprechende Literatur verwiesen.

UML und Dialekte

Neben der Standard-UML gibt es noch weitere Dialekte,¹³ die auf die UML aufbauen. Stellvertretend hierfür beschreiben wir die SoaML¹⁴ etwas genauer. Zu nennen wären aber noch weitere, wie die SysML¹⁵ oder auch das CWM.¹⁶

¹³Wir haben uns erlaubt hier den Begriff Dialekt zu verwenden. Im Grunde handelt es sich um MOF-basierende Sprachen oder um UML-Profile, die den Sprachumfang erweitern.

¹⁴Service Oriented Architecture Modeling Language.

¹⁵OMG System Modeling Language.

¹⁶Common Warehouse Metamodel.

UML

Die Unified Modeling Language ist eine formale graphische Notation zur Modellierung, Spezifikation, Dokumentation und Visualisierung von Strukturen und Abläufen. Sie ist technikunabhängig und liegt derzeit in Version 2.3 vor [OMG]. Mit der UML lässt sich eine vollständige Beschreibung von Systemen aller Art bewerkstelligen. Die UML verfügt zur Modellierung von Systemen über mehrere Diagrammartentypen. Man unterscheidet dabei zwischen statischen Diagrammen und dynamischen Diagrammen. Erstere beschreiben die Struktur, letztere das Verhalten des zu modellierenden Systems. Einen Überblick über die Diagramme im Einzelnen gibt Tabelle 2.2.

Tabelle 2.2 UML-Diagrammartentypen

| Statische Diagramme | Dynamische Diagramme |
|------------------------------|-------------------------------|
| Klassendiagramm | Anwendungsfalldiagramm |
| Objektdiagramm | Zustandsdiagramm |
| Paketdiagramm | Aktivitätsdiagramm |
| Kompositionsstrukturdiagramm | Sequenzdiagramm |
| Komponentendiagramm | Kommunikationsdiagramm |
| Verteilungsdiagramm | Timingdiagramm |
| | Interaktionsübersichtdiagramm |

Zur Anpassung der UML an den fachlichen Gegenstandsbereich steht seit UML 2.0 der Profilmechanismus zur Verfügung. Ein Profil stellt eine Erweiterung der UML auf Metamodellebene dar. Die Erweiterung erfolgt dabei mittels den sogenannten Stereotypen. Ein Stereotyp ist eine Metamodellklasse, die beschreibt, auf welche Art eine andere Metamodellklasse erweitert werden darf.

In der MDSOA werden hauptsächlich Klassen- und Komponentendiagramme verwendet. Wir haben dabei die Möglichkeit der Erstellung eigener Profile genutzt. So wurden zum Beispiel für EJBs eigene Stereotypen erstellt und mit weiteren Eigenschaften für das objektrelationale Mapping versehen. Für die Dokumentation werden auch das Kompositionsstrukturdiagramm (Servicearchitektur) und Sequenzdiagramm (Darstellung der architekturkonformen Aufrufreihenfolge) verwendet.

Für die Darstellung einer Serviceorientierten Architektur mit der UML sind derzeit Bestrebungen zur Spezifizierung von standardisierten UML-Profilen im Gange. Die OMG unterstützt den Ansatz der SOA Modeling Language (SoaML) [SOAML], welcher sich derzeit noch in Erarbeitung befindet.¹⁷

SoaML

Seit 2005 sind die Bemühungen der OMG, mit dem „UML Profile and Metamodel for Services“ (UPMS) ein standardisiertes Metamodell für die Entwicklung

¹⁷ Aktueller Stand ist die 1.0 Beta 2 vom Dezember 2009.

von Diensten zu etablieren, im Gange [UMPS]. Mit diesem Profil können künftige Generationen von MDA-Werkzeugen auf einen vereinheitlichten Rahmen zur Service-Modellierung zurückgreifen. Als Spezifikation für ein Metamodell, welches der UPMS entspricht, wurde im April 2009 die SOA Modeling Language (SoaML) von der OMG veröffentlicht. Es handelt sich dabei noch um eine vorläufige Version, da die Spezifizierung noch nicht abgeschlossen ist.

Ziel der SoaML ist die Unterstützung von Design und Modellierung von Services und ihrer Anforderungen. Zudem ermöglicht die Sprache die Spezifikation des Zusammenspiels von mehreren Services, die eine Serviceorientierte Architektur bilden. Da SOA als ein Architekturparadigma verstanden wird, kann SoaML in der Architektur-Schicht der MDA verwendet werden, um die Beziehungen der Dienste einer SOA zu beschreiben. Die Spezifikation technischer Details der Implementierung ist ausdrücklich kein Ziel der Sprache, weil diese auf einer anderen Ebene der MDA behandelt werden. Es soll darüber hinaus, ganz im Sinne der MDA, möglich sein, Artefakte daraus zu generieren.

Um eine Serviceorientierte Architektur aus den Unternehmensanforderungen zu entwickeln, ist zunächst zu klären, welche Anforderungen durch die IT unterstützt werden sollen. Die SoaML bietet mit den sogenannten *Capabilities* eine Möglichkeit, anhand der Requirements Dienste zu identifizieren. Durch die Capabilities können die Fähigkeiten eines Services in abstrakter Weise beschrieben werden, bevor der Service existiert oder bekannt ist. Die Darstellung der Capabilities ähnelt der Darstellung von Anforderungen in der SysML.

Mit der SoaML können Serviceorientierte Architekturen sowohl mit einem Schnittstellen-orientierten, wie auch mit einem Servicekontrakt-orientierten Ansatz modelliert werden. Zu deren Unterstützung enthält die Sprache mehrere Elemente, wie z. B. `ServiceInterface` oder `ServiceContract`. Da die beiden Ansätze eine SOA nur aus unterschiedlichen Blickwinkeln betrachten, kommt es bei den Elementen teilweise zu Überlappungen, die sich auch in Redundanzen im Modell äußern, wenn man beide Ansätze gleichzeitig anwendet.

Interface-basierter Ansatz

Beim Interface-basierten Ansatz geht es darum, die SOA aus Sicht der Teilnehmer zu modellieren. Die Definition der Interaktion der Teilnehmer erfolgt anhand der Beschreibung der Schnittstellen und deren Operationen. Diese Art der Beschreibung der Dienste kann für die Bereitstellung der Funktionalitäten existierender Systeme genutzt werden. Bei einer unidirektionalen Schnittstelle, in welcher nur eine Seite einen Dienst in Anspruch nimmt und es keine geregelten Abläufe zwischen den Teilnehmern gibt, genügt dazu ein UML-Interface. In diesem Fall gibt es auch keine Rückfragen (Callback) von Seiten des Dienstansbieters. Diese einfache Art der Serviceinteraktion gleicht dem aus der Objektorientierten-Programmierung oder aus Remote-Procedure-Calls (RPC) bekanntem Schema.

Es besteht aber auch die Möglichkeit, dass beide Seiten für die Nutzung eines Services die Dienste des anderen in Anspruch nehmen. Solche mehrseitigen Interaktionen laufen meist auch mit einer nach einem Protokoll festgelegten Reihenfolge

der Operationen ab. Zur Darstellung einer solchen bidirektionalen Interaktion existiert in der SoaML das Konzept des *ServiceInterface*. Mit diesem kann aber auch eine unidirektionale Kommunikation spezifiziert werden. Es beinhaltet sowohl die von einem Dienst angebotenen als auch die von ihm genutzten Schnittstellen. Die Spezifikation der Schnittstellen selbst ist nicht Teil der Serviceschnittstelle, da diese eine Dienstschnittstelle auf einem abstrakteren Niveau beschreibt. Die Spezifikation wird stattdessen mit UML-Schnittstellen beschrieben. Das Verhalten zwischen den Teilnehmern der Serviceschnittstelle kann anhand von Serviceprotokollen genauer beschrieben werden. Es handelt sich dabei um Aktivitätsdiagramme, welche die Ablaufreihenfolge von Operationsaufrufen definieren. Diese spezifizieren folglich die Choreographie der Serviceschnittstelle. Die nähere Beschreibung der Dienstchoreographie ist für die Beschreibung der SOA-Anwendung nicht unbedingt notwendig, wenn alle Operationen der Dienste ohne Einhaltung einer bestimmten Reihenfolge durchgeführt werden können.

Servicekontrakt-basierter Ansatz

Der Servicekontrakt-basierte Ansatz der Servicemodellierung beschreibt die Interaktion der Dienste aus der Sicht einer Zusammenarbeit. Der Fokus liegt also nicht auf dem einzelnen Dienst, sondern vielmehr in den Rollen der Teilnehmer in der Kooperation und der abstrakten Funktionalität die einen geschäftlichen Mehrwert erbringt. Der kontraktbasierte Ansatz eignet sich daher für Szenarien, in denen eine Serviceorientierte Architektur neu entworfen wird oder wenn komplexere Zusammenhänge dargestellt werden müssen. Zur Beschreibung der Zusammenarbeit mehrerer Schnittstellen existiert in der SoaML der Stereotyp *ServiceContract*, welcher mit einer UML-Kollaboration benutzt wird. Ein Servicevertrag wird zwischen den Teilnehmern geschlossen und definiert die Bedingungen, die von den Teilnehmern zur Dienstnutzung und Bereitstellung eingehalten werden müssen. Die Bedingungen werden anhand der von beiden Seiten verwendeten Serviceschnittstellen und durch die Darstellung der Choreographie mit Aktivitätsdiagrammen beschrieben. Zudem wird für jede Seite anhand der Serviceschnittstelle eine Rolle angegeben, welche diese in der Zusammenarbeit verkörpert.

Service Architektur

Zur Beschreibung des Zusammenhangs aller Dienste einer SOA gibt es darüberhinaus den Stereotyp *ServiceArchitecture*, der mit einem Kollaborationsdiagramm verwendet wird. Die Servicearchitektur stellt den Gesamtkontext dar und veranschaulicht das Zusammenspiel mehrerer Teilnehmer. Die Zusammenarbeit wird durch Serviceverträge zwischen den Teilnehmern dargestellt. Die Art in welcher ein Teilnehmer mit einem anderen Teilnehmer zusammenarbeitet, wird durch Angabe seiner Rolle angegeben. Ergänzend kann die Orchestrierung der Dienste durch ein Aktivitätsdiagramm beschrieben werden. Abbildung 2.6 illustriert beispielhaft die Servicearchitektur des Investitionsantragsprozesses. Dabei stellen die Rechtecke die Teilnehmer dar. Verbunden werden sie mit den ovalen Service-Kontrakten.

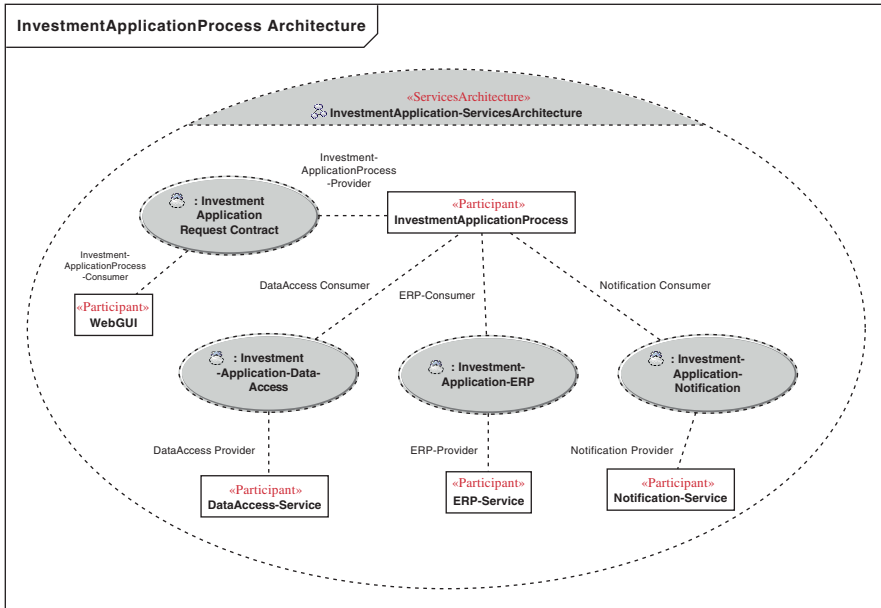


Abb. 2.6 Service-Architektur des Investitionsprozesses

Serviceteilnehmer können außerdem mit einem Komponentendiagramm näher beschrieben werden, welches die vom Teilnehmer benutzten und bereitgestellten Interfaces enthält. Ein Teilnehmer kann in einer SOA als zusammengesetzter Service auch aus mehreren anderen Diensten bestehen. Um diesem Umstand Rechnung zu tragen, besteht die Möglichkeit die Zusammensetzung eines Dienstes in einer ParticipantArchitecture mittels eines Kompositionsdiagramms zu modellieren. Diese entspricht im Wesentlichen einer ServiceArchitecture die den Teilnehmer darstellt. Zusätzlich kann aber dem Teilnehmer noch ein Interface zugeordnet werden. Einen guten Einstieg in die praktische Anwendung der SoaML gibt [SOAA].

Fazit

Abschließend kann festgestellt werden, dass die SoaML alle Möglichkeiten bereitstellt, die Architektur einer SOA darzustellen. Da die Sprache sich allerdings noch im Prozess der Standardisierung befindet, ist sie noch nicht sehr weit verbreitet. Dementsprechend gibt es nur wenige Tools, die die Modellierung mit der SoaML ermöglichen. Der in diesem Buch verwendete *Innovator* gehört zu den wenigen Ausnahmen. Da auch im Allgemeinen noch keine Praxiserfahrung mit der Modellierungssprache vorliegt, wird die SOA in diesem Buch noch nicht mit der SoaML beschrieben.

BPMN 2.0

Business Process Model and Notation (BPMN)¹⁸ ist der Standard für die grafische Beschreibung von Geschäftsprozessen, der sich auch in Deutschland immer weiter durchsetzt. Neben der Repräsentation von reinen Prozessabläufen spezifiziert die BPMN auch Details, um technische Abläufe zu modellieren und eignet sich daher am besten von allen Geschäftsprozessnotationen, sowohl fachliche als auch technische Prozesse in einer „Sprache“ zu modellieren.

Die Notation wurde maßgeblich von Stephen A. White (IBM) entwickelt und 2004 zunächst durch die Business Process Management Initiative (BPMI) veröffentlicht. Im Jahr 2005 wurde die Weiterentwicklung von der Object Management Group (OMG) übernommen. Seit 2006 ist die BPMN ein offizieller OMG-Standard, wie die bereits erwähnte UML für die Softwareentwicklung.

Seit Januar 2011 ist die Notation in der Version 2.0 offiziell durch die OMG freigegeben und veröffentlicht.

Eine Kurzeinführung für BPMN-Neulinge

Um BPMN-Diagramme zu lesen, bedarf es keines großen Aufwands. Jeder der sich bereits mit Prozessmodellierung beschäftigt hat, ist sogar einen Schritt voraus und kann sich die erforderlichen Elemente anhand einer Legende aneignen, aber wie erwähnt nur zum Lesen!

Das Rad wird mit der BPMN nicht neu erfunden, sondern nur verbessert, daher gibt es als Grundstock die Basiselemente, die immer zur Modellierung eines Prozesses erforderlich sind.

Ein Prozess muss immer irgendwie Starten und Enden, dazwischen müssen einzelne Prozessschritte abgearbeitet werden. Um den Prozessverlauf darzustellen und die einzelnen Prozessschritte zu einem Ablauf zusammenzufügen, braucht man eine Art Verbinder. Da es auch Ausnahmen oder Negativfälle in einem Prozess gibt, sind Elemente für Verzweigungen des Prozessverlaufs erforderlich, sowie Elemente für eine spätere Zusammenführung. Meist werden nicht alle Prozessschritte von einer Person, Abteilung, System, etc. durchgeführt, daher muss es eine Darstellungsform geben, um verschiedene Verantwortliche bzw. Rollen abzubilden. Auch Daten oder Produkte spielen eine wichtige Rolle in Prozessen, so gibt es in ihnen und natürlich auch in der BPMN ein Notationselement um sie darzustellen.

Die folgende Kurzlegende in Abb. 2.7 gibt einen Überblick über diese Basiselemente, die anschließend genauer beschrieben werden.

¹⁸Ein Übersichtsposter erhalten sie zum Download auch unter [MID1].

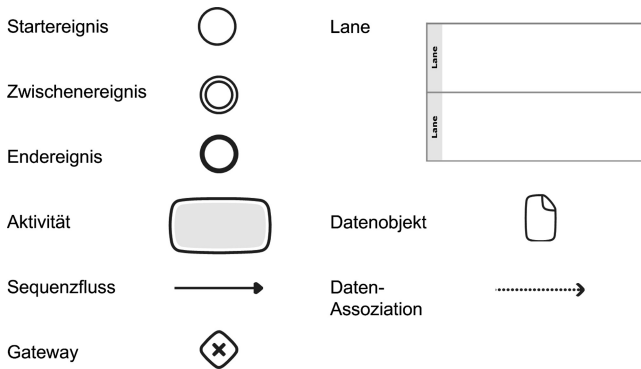


Abb. 2.7 BPMN-Basiselemente

- Ein Startereignis (engl. Start Event) instanziiert den Prozess, löst ihn sozusagen aus. Es gibt unterschiedliche Arten von Startereignissen, auf die wir bei Bedarf noch eingehen. Es wird mit einem einfachen Kreis dargestellt.
- Das Endereignis beendet den jeweiligen Prozessfluss. Auch hier gibt es unterschiedliche Arten. Die Darstellungsform ist ebenfalls ein Kreis, diesmal mit einem dicken Rahmen.
- Zwischenereignisse: Neben Ereignissen, die zu Beginn und zum Ende eintreten, kommt es immer wieder vor, dass Ereignisse den Prozessablauf beeinflussen. Diese Ereignisse werden über Zwischenereignisse abgebildet und mit einem doppelt umrandeten Kreis gezeichnet.
- Mit einer Aktivität werden Prozessschritte dargestellt. Die BPMN unterteilt die Aktivität grob in autonome Schritte in einem Prozess, den Tasks und den Teilprozessen, die zur Verfeinerung dienen und wieder einen ganzen Prozess enthalten können. Sie werden als Rechteck mit abgerundeten Ecken dargestellt.
- Der Sequenzfluss beschreibt den Übergang von einem Prozessknoten zum nächsten. Er ist ein durchgezogener Pfeil mit einer ausgefüllten Pfeilspitze.
- An einem Gateway verzweigt der Sequenzfluss oder wird zusammengeführt. Das dargestellte Gateway repräsentiert eine datenbasierte exklusive Verzweigung (XOR). Auch hierfür werden die Variationen der möglichen Verzweigungen weiter unten dargestellt.
- Lanes (Bahnen) gehören zur sogenannten Schwimmbahn-Darstellung. Mit ihnen werden u.a. Rollen oder Prozessbeteiligte repräsentiert, in deren Verantwortung die in der Bahn enthaltenen Aktivitäten ausgeführt werden. Sie trennen den Prozess, vergleichbar mit den einzelnen Begrenzungen im Schwimmbassin, die verschiedene Schwimmer während eines Wettkampfes auf ihren Bahnen halten sollen.
- Datenobjekte repräsentieren benötigte Ein- oder Ausgaben für Aktivitäten oder ganze Prozesse.
- Datenassoziationen verbinden Datenobjekte mit Aktivitäten oder Ereignissen.

Die OMG bietet mit der Spezifikation unter [BPMN] die ausführlichste Quelle zur BPMN. Aber auch [ALL09] eignet sich sehr gut als Einführung in die BPMN.

BPMN-Wissen zum Buch

Der folgende Abschnitt beschreibt die im unseren Buchbeispielen vorkommenden BPMN-Notationselemente im Detail.

Blanko-Ereignisse



Abb. 2.8 Blanko-Ereignisse

Bereits in der Kurzeinführung wurden Start-, Zwischen und Endereignisse beschrieben, die auch in Abbildung 2.8 dargestellt sind. Dabei handelte es sich um die einfachste Form von Ereignissen, um die Blanko-Ereignisse (engl. Plain-Event). Es sind untypisierte Ereignisse, die zum Einsatz kommen, wenn (noch) kein explizites Ereignis definiert ist und keines der in diesem Abschnitt weiter unten beschriebenen Ereignistypen verwendet werden sollte, um den Prozessablauf treffender oder deutlicher zu beschreiben. Abbildung 2.8 zeigt die auftretenden Blanko-Ereignisse.

Neben der Unterteilung der Ereignisse in Start, Zwischen und Ende unterscheidet die Spezifikation die Ereignisse nach Eintretend (engl. Catching, in der Abbildung hellgrau) und Auslösend (engl. Throwing, in der Abbildung schwarz). Start-Ereignisse können immer nur eintretend und Ende-Ereignisse immer nur auslösend sein. Für Zwischenereignisse kann es beide Formen geben.

Für das untypisierte Blanko-Zwischenereignis gibt es nur die eintretende Form.

Nachrichten-Ereignisse

Der Empfang und der Versand von Nachrichten können in der BPMN mit Hilfe von Nachrichten-Ereignissen (engl. Message-Events) modelliert werden. Diese zeigt Abbildung 2.9.

Bei den Nachrichten-Zwischenereignissen gibt es beide Formen. Alle Elemente, die für auslösende Ereignisse stehen, sind im Gegensatz zu eintretenden Ereignissen immer ausgefüllt. Deshalb haben sowohl das auslösende Zwischenereignis „Rückfrage gestellt“, als auch das sendende Endereignis „Bestätigung verschickt“ ein ausgefülltes Briefsymbol.



Abb. 2.9 Nachrichten-Ereignisse

Zeit-Ereignisse

Zeitereignisse gibt es nur als Start- und als (eintretendes) Zwischenereignis. Mit ihnen kann man semantisch einen Zeitpunkt oder eine Zeitspanne ausdrücken. Abbildung 2.10 illustriert die Verwendung der Zeitereignisse.



Abb. 2.10 Zeit-Ereignisse

Hier als Starterereignis vom Typ Zeit modelliert, wird periodisch zu einem definierten Zeitpunkt („jeden 1. des Monats“) der Prozess gestartet.

Als zweites Zwischenereignis ist eine Zeitspanne definiert, wodurch der Prozess (ähnlich einer Eieruhr) erst „nach drei Tagen“ fortgesetzt wird. Die modellierte Zeitspanne ist immer eine vom Akteur bestimmte Verzögerung und kann hier nicht durch äußere Umstände beeinflusst werden.

Signal-Ereignisse

Eine weitere Möglichkeit, Ereignisse mit der BPMN zu modellieren, geht über Signale. Die folgende Abbildung 2.11 zeigt die Signalereignisse der BPMN. Signale aus einem Prozess zu senden, also das auslösende Zwischen- und Endereignis einzusetzen, ist vergleichbar mit einer Radiosendung die ausgestrahlt wird.

Signale zu empfangen entspricht prinzipiell dem Empfang einer Sendung des Radioprogramms mit einem Empfänger. Dies wird mit dem Start- und dem eintretenden Zwischenereignis modelliert.



Abb. 2.11 Signal-Ereignisse

Dadurch wird eine 1:N-Beziehung möglich. Ein Sender kann also einen oder mehrere Empfänger haben, die in demselben oder in einem ganz anderen Prozess liegen können.

Weitere Beispiele wären zum Beispiel Zeitungsannoncen, Meilensteine in einem Projekt oder eine Ampel.

Link-Ereignisse

Sie kommen nur paarweise vor und sind die äquivalente Darstellung eines Sequenzflusses.

Um eine Verbindung nach der Aufteilung des Sequenzflusses herstellen zu können, müssen sowohl an der Quelle, als auch am Ziel dieselbe Ereignisdefinition

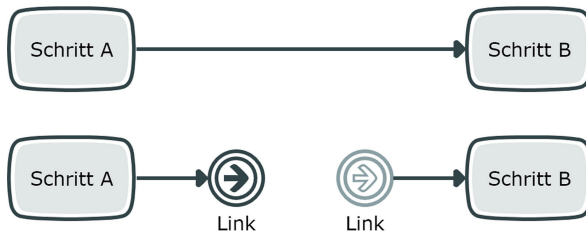


Abb. 2.12 Link-Ereignisse

verwendet werden. Sie haben daher eine identische Bezeichnung. Beim Linkereignis gibt es auch nur diese beiden Zwischenereignisformen. Prozessstart und -ende können damit nicht modelliert werden. Die Link-Ereignisse dürfen auch nur innerhalb desselben Prozesses verwendet werden. Ein Einsatz über Prozess-Grenzen hinweg ist nicht erlaubt.

Sie werden häufig als Rücksprünge verwendet, um in komplexen Diagrammen die Übersichtlichkeit zu wahren. Je umfangreicher das Diagramm wird, desto sinnvoller ist der Einsatz der Links, um zum Beispiel Sequenzflüsse nicht über das ganze Diagramm zurück an den Anfang modellieren zu müssen.

Link-Ereignisse sind ebenso ein praktisches Hilfsmittel für umfangreiche Prozessmodellierungen, die sogenannten „Prozessketten“, da sie zur Verbindung von seitenübergreifenden Diagrammen eingesetzt werden können.

Wichtig ist dabei zu beachten, dass jede Quelle nur ein Ziel haben kann. Ein Ziel darf aber von mehreren verschiedenen Quellen aus angesprungen werden. Abb. 2.12 illustriert die Verwendung von Link-Ereignissen.

Aktivitäten und deren Ausprägungen

In der BPMN untergliedern sich die Aktivitäten in Tasks (den atomaren Schritten in einem Prozess) oder in einer Sammlung von mehreren Schritten, die entweder einen Teil- oder einen Unterprozess darstellen. In Abbildung 2.13 sehen wir die einzelnen Ausprägungen der Aktivität.

Eingebettete Unterprozesse dienen hauptsächlich der Zusammenfassung von Details und einer übersichtlichen Gestaltung von komplexen Prozessen. Ein eingebetteter Unterprozess besitzt immer nur einen Oberprozess. Von diesem Oberprozess ist er abhängig, und ohne ihn ist er meistens auch nicht sinnvoll. Er wird daher auch nur mit ihm im Zusammenhang verwendet und kann ausschließlich durch ihn gestartet werden.

Aus diesem Grund können eingebettete Unterprozesse ausschließlich mit einem Blanko-Startereignis beginnen, da weitere externe Umstände ihn nicht auslösen können. Auch die Verwendung von Pools und Lanes ist ausgeschlossen. Sequenzflüsse dürfen die Grenzen des eingebetteten Unterprozesses nicht überschreiten, nur bei Nachrichtenflüssen ist dies erlaubt.



Abb. 2.13 Aktivitäten

Wiederverwendbare Unterprozesse

Sie sind im Gegensatz zu eingebetteten Unterprozessen für mehrere Prozesse verfügbar. Auch bei den wiederverwendbaren Unterprozessen gilt, dass sie ein Blanko-Startereignis haben müssen, aber zudem können sie auch durch externe Ereignisse, wie Nachrichten- und Zeitereignisse, ausgelöst werden. Auch die Modellierung mit mehreren Pools und Lanes ist bei ihnen erlaubt.

Durch ihre Wiederverwendbarkeit haben sie eine große Bedeutung bei der Automatisierung. Da sie aber autonom sind und einen eigenen „Daten-Scope“ besitzen ist es gegebenenfalls erforderlich, ein Mapping der Daten zu definieren, da diese Daten eine andere Bezeichnung als im aufrufenden Oberprozess tragen können.

Task

Die BPMN unterscheidet verschiedene Task-Typen, die wir in Abb. 2.14 aufgeführt haben.

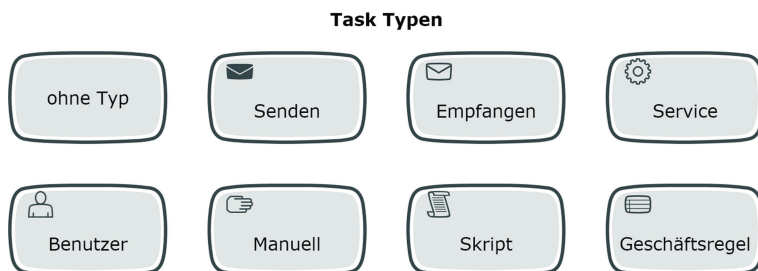


Abb. 2.14 Die Task-Typen der BPMN 2.0

- Ohne Typ: Default bei der Modellierung. Beschreibt einen Schritt in einem Prozessablauf ohne spezifische Aussage über den Typ des Tasks.
- Sende-Task: Sendet eine Nachricht an einen externen Teilnehmer. Entspricht einem Nachrichten-sendenden Ereignis.
- Empfangs-Task: Wartet auf eine Nachricht eines externen Teilnehmers. Entspricht einem Nachrichten-empfangenden Ereignis.
- Service-Tasks: Ein automatisierter Schritt. Zum Beispiel der Aufruf einer (Web-) Service-Operation oder eine Anwendungsfunktion.
- Benutzer-Task: Dieser Task erwartet einen Input von einem Benutzer.¹⁹ Teilautomatisiert in einem entsprechenden Workflow, zum Beispiel durch Eingabe von Daten in einer System-Maske.
- Manueller Task: Durchführen einer rein manuellen Tätigkeit ohne IT-Unterstützung. Zum Beispiel die Ablage einer Akte.
- Skript-Task: Ein auf einem Server ausgeführter automatisierter Task. Repräsentiert zum Beispiel eine Geschäftslogik, die auf einer Prozessengine ausgeführt werden kann. Ebenso gut aber auch die typischen Automatisierungsskripts die zum Beispiel mittels eines asynchronen Aufrufs über eine Queue angestoßen werden. Dabei erfolgt keine menschliche Interaktion.
- Geschäftsregel-Task: Er repräsentiert den Aufruf einer oder mehrerer Geschäftsregeln um eine Entscheidung herbei zu führen oder ein Ergebnis zu ermitteln. Dies kann zum Beispiel der Aufruf einer auf einer Rule-Engine bereitgestellten ausführbaren Geschäftsregel sein, die über eine Service-Schnittstelle aufgerufen werden kann.

Gateways

Gateways²⁰ - für die auch wir keine adäquate Übersetzung aus dem Englischen gefunden haben - können als Verzweigungen und Zusammenführungen des Prozessablaufs bezeichnet werden. Beides in einem Element! In diesem Abschnitt werden wir die einzelnen Arten der Gateways und ihre Verwendung wieder anhand eines Beispiels veranschaulichen.

Datenbasiertes exklusives Gateway (XOR)

Datenbasiert bedeutet hierbei, dass die Entscheidung, welcher Weg beziehungsweise welche ausgehende Kante bei einer Verzweigung gewählt wird, von Daten abhängig ist, die zu diesem Zeitpunkt verfügbar sein müssen. Abb. 2.15 zeigt dessen Verwendung.

¹⁹Stichwort: „Human Interaction“ (menschliche Interaktion)

²⁰Vielleicht: „Entscheidungsknoten“, „Zusammenführung“

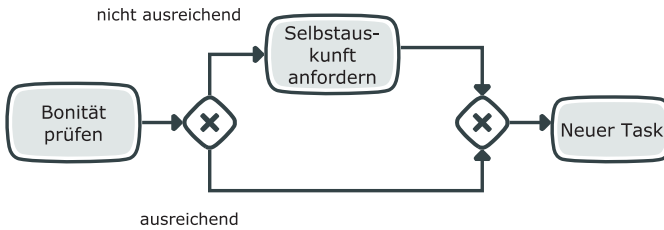


Abb. 2.15 Verwendung eines datenbasierten exklusiven Gateways

Das „Exklusiv“ in der Bezeichnung bedeutet, dass nur eine der ausgehenden Kanten aktiviert werden kann, also „entweder-oder“, auch „XOR“ genannt. Entweder die Bonitätsprüfung war „positiv“ (ausreichend) oder sie war „negativ“ (nicht ausreichend) und es muss eine Selbstauskunft angefordert werden.

Ereignisbasiertes exklusives Gateway

Auch im Ereignisbasierten exklusiven Gateway (Abb. 2.16) wird nur eine ausgehende Kante aktiviert (XOR). Hier aber nicht nach „Datenlage“, sondern danach, welches Ereignis zuerst eintritt.

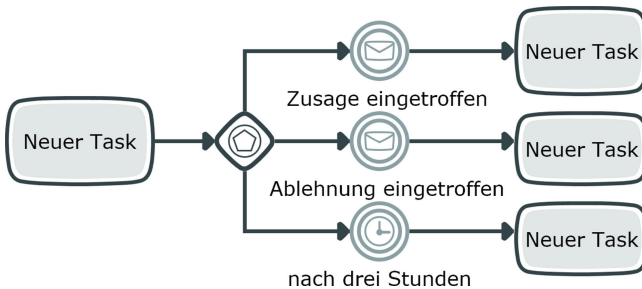


Abb. 2.16 Verwendung eines Ereignisbasierten exklusiven Gateways

Paralleles Gateway (AND)

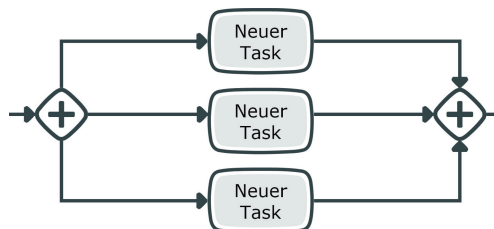


Abb. 2.17 Einsatz des parallelen Gateways

Ein paralleles Gateway (Abb. 2.17) ermöglicht die simultane Ausführung der folgenden Aktivitäten. Das Gateway wird also als „und“ eingesetzt und ist aus der technischen Welt auch als „AND“ bekannt. Alle ausgehenden Kanten nach dem parallelen Gateway müssen durchlaufen werden. Ein ankommendes Token²¹ wird an dieser Stelle dupliziert und über alle ausgehenden Pfade an die folgenden Aktivitäten weitergeleitet.

Wird das Gateway für eine Zusammenführung verwendet, wird auf alle eingehenden Pfade gewartet, bevor der Sequenzfluss fortgesetzt wird. Pro Kante muss also ein Token eingehen. Diese werden miteinander verschmolzen und als ein Token an die nachfolgende Aktivität weitergegeben.

Kollaborationen

Kollaborationen sind aus unserer Sicht eine der wirklich herausragenden Möglichkeiten der Modellierung mit der BPMN. Mit ihr lassen sich die Schnittstellen zwischen einem oder mehreren Beteiligten und deren Prozesse sehr elegant und kompakt abbilden.

Abbildung 2.18 zeigt eine Kollaboration mit zwei Beteiligten. Dabei sind die Prozesse nicht eingeblendet („Blackbox-Darstellung“). Es wird aber trotzdem deutlich, dass die Beteiligten über die Nachrichten „Frage“ und „Antwort“ miteinander kommunizieren. Üblicherweise würde im Prozess an den entsprechenden Stellen

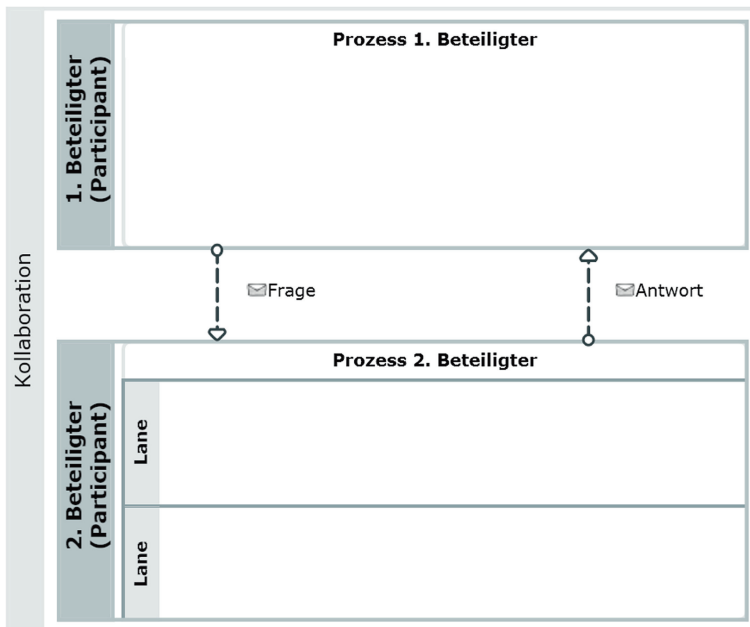


Abb. 2.18 Kollaboration mit zwei Beteiligten und Austausch von Nachrichten

²¹ Ein Token kann man sich wie einen Marker oder einen Chip vorstellen, der zu Beginn mit dem Start des Prozesses erzeugt wird und entlang des Sequenzflusses durch den Prozess „wandert“.

ein empfangendes und sendendes Nachrichten-Ereignis stehen oder analog dazu ein Sende- und Empfangs-Task.

Eine Kollaboration stellt also das Zusammenspiel einer oder mehrerer Prozesse dar, die nicht zentral gesteuert werden, sondern die sich über Nachrichten „synchronisieren“.²²

Die Beteiligten können dabei verschiedene Unternehmen oder Partner sein, aber auch unterschiedliche, eigenständige Prozesse verschiedener Abteilungen eines Unternehmens beinhalten.

MDSOA nutzt Kollaborationen zum Beispiel auch, um die Service-Choreographie in sogenannten „Service-Kollaborationen“ zu modellieren. Aus diesen Service-Kollaborationen werden dann entsprechende BPEL-Artefakte generiert.

Modellierungswerkzeuge

MID Innovator

Der *Innovator* ist eine integrierte Modellierungsplattform für die modellgestützte Entwicklung, die von der *MID GmbH* aus Nürnberg entwickelt wird. Der *Innovator* liegt derzeit in Version 11.3 vor und besteht aus mehreren Editionen. Die einzelnen Produkte ergänzen sich und decken den gesamten Softwareentwicklungsprozess von der Anforderungs-Analyse bis zum fertigen Softwaremodell ab. Tabelle 2.3 gibt eine Übersicht über die Editionen des *Innovator*.

Tabelle 2.3 MID Innovator Editionen

| Innovator Edition | Aufgabenbereich |
|-----------------------------------|--|
| Innovator for Business Analysts | Geschäftsprozessmodellierung auf Basis der BPMN 2.0, Use-Cases, Komponenten- und Klassenmodelle der UML 2 und mehr |
| Innovator Object eXcellence | Softwareentwicklung nach objektorientierten Methoden der UML 2, SoaML und SysML |
| Innovator for Database Architects | Datenmodellierung mit IMM/ERM/SERM |
| Innovator Function | Unterstützt strukturierte Analyse und Design |

Die Plattform ermöglicht dadurch eine durchgängige Entwicklung von Anwendungen beginnend bei der Anforderungsanalyse über die Geschäftsprozessmodellierung bis hin zur Datenmodellierung, sowie der Modellierung der Softwarearchitekturen. Das führt dazu, dass sowohl der Fachbereich als auch die IT mit demselben Werkzeug arbeiten können. Dies bietet Vorteile sowohl in der Kommunikation zwischen den Abteilungen, als auch im Entwicklungsprozess selbst. Eine Konsistenz der verschiedenen Entwicklungsphasen und in den Modellen ist gegeben.

²²Was auch als „Choreographie“ bezeichnet wird.

Mit der Erweiterung *Word PlugIn for Innovator* bietet *Innovator* eine Möglichkeit zur textuellen Aufnahme von Anforderungen an. Durch eine Integration in MS Word können Anforderungen in der gewohnten MS Office Umgebung formuliert und komfortabel mit *Innovator* abgeglichen werden.

Die breite Kombination der Einsatzmöglichkeiten und die Tatsache, dass all dies unter dem Dach desselben Tools möglich ist, stellt ein wesentliches Merkmal dar, welches *Innovator* von anderen Werkzeugen für die modellbasierte Entwicklung unterscheidet. Zudem lassen sich die Modellelemente verschiedener Modelle durch Abhängigkeiten verbinden, so dass zum Beispiel Zusammenhänge zwischen der fachlichen Modellierung und des technischen Designs der Anwendung dokumentiert sind. Dadurch ist nachvollziehbar, auf welche Anforderung ein spezifisches Feature zurückgeht (Traceability). Diese Verknüpfung ist mit allen Modellelementen der Geschäftsprozess- und Software-Analyse-Modelle möglich, wodurch der *Innovator* auch die Durchführung von Impact-Analysen auf allen Modellierungsebenen gestattet. Da die MID GmbH Kunden auch bei der Durchführung von Projekten unterstützt und für die modellgetriebene Entwicklung eine eigene Vorgehensweise beziehungsweise Methodik entwickelt hat (MID Modellierungsmethodik M³), ist die Plattform auf diese Methodik optimal abgestimmt.

Der weiten Palette an Einsatzmöglichkeiten entsprechend, unterstützt *Innovator* auch ein breit aufgestelltes Portfolio an Standards. Die Anforderungsanalyse kann beispielsweise mit der SysML oder anhand textueller Requirements erfolgen. Die Organisationsstrukturen eines Unternehmens lassen sich mit Organigrammen modellieren. Mit Maskenfluss-Diagrammen kann der Ablauf von Eingabemasken spezifiziert werden. Zur Modellierung von Geschäftsprozessen kann die BPMN 2.0 genutzt werden. Die Lücken der BPMN 2 können durch eine Kombination mit der UML 2 geschlossen werden, wobei beide Modellierungssprachen integriert nutzbar sind. Dies betrifft insbesondere die Verwendung von Anwendungsfall- und Klassen-Diagrammen im *Innovator for Business Analysts*.

Zur Modellierung der Datensicht existieren mehrere Möglichkeiten. Der *Innovator* ermöglicht die weitverbreitete Entwicklung mit Entity-Relationship-Modellen (ERM) und erlaubt dabei die Verwendung der Chen-, SERM-, James-Martin-, DSA- oder UML-Notationen zur relationalen Datenmodellierung. Ferner können die Datenstrukturen durch die Anbindung externer DBMS abgeglichen werden.

Für das Design und die Entwicklung der Applikation unterstützt der *Innovator* die Standards UML 2 und über die M³ auch die MDA der OMG. Einzelne Modelle sind mittels XMI-Import und- Export auch mit anderen Modellierungswerkzeugen austauschbar.

Innovator verfügt aber auch über Fähigkeiten des Reverse-Engineerings, wodurch bereits vorhandener Quellcode eingelesen und in ein Modell importiert werden kann.

Auch im SOA-Umfeld ist der *Innovator for Business Analysts* mit der BPMN 2 einsetzbar. Außerdem wird mit der SoaML ein weiterer Standard der OMG zur Modellierung von serviceorientierten Architekturen unterstützt. Zudem gibt es Erweiterungen, die den Import und Export von BPEL, WSDL und XSD ermöglichen.

Von den SOA-Fähigkeiten des *Innovators* werden wir im weiteren Verlauf des Buches Gebrauch machen.

Die jeweiligen *Innovator*-Editionen sind eng integriert. Durch weitgehend automatisierte Modelltransformationen erlaubt die Modellierungsplattform einen konvergenten Entwicklungsprozess. Dieser stellt sicher, dass die Modelle der Geschäftsprozesse, Datenbankstrukturen und des objektorientierten Designs in sich konsistent sind. Zur Prüfung dieser Konsistenz im Rahmen der Qualitätssicherung, ist ein Tool zur automatisierten Modellverifikation enthalten. Mit diesem lässt sich auch prüfen, ob bei der Entwicklung die Modellierungsvorschriften eingehalten wurden. Es lässt sich damit kontrollieren, ob die Vorgaben sowohl semantisch, als auch syntaktisch eingehalten wurden.

Speziell für Modelltransformationen bietet der *Innovator* ein Tool an, das die Erstellung des Mappings zwischen Modellen vereinfacht (M2M-SDK). Modelltransformationen können aber auch noch auf andere Weisen erstellt werden. Der *Innovator* ist in dieser Hinsicht sehr flexibel. Da die Modellinhalte im besten Fall später zur Erzeugung von Artefakten führen sollen, wie zum Beispiel von Source-Code, ist im *Innovator* eine enge Anbindung an das *openArchitectureWare* Framework (oAW) gegeben. Damit können die Artefakte flexibel anhand anpassbarer Vorlagen erzeugt werden.

Die Code-Generierung kann aber auch über die API des *Innovators* erfolgen und setzt den Einsatz von oAW nicht zwingend voraus. Dies ist ein weiteres Beispiel dafür, dass der *Innovator* individuell an die Bedürfnisse des Anwenders anpassbar und verwendbar ist.

Innovator bietet daneben die Möglichkeit, aus den Modellinhalten automatisiert eine Dokumentation in verschiedenen Ausgabeformaten zu erstellen. Näheres dazu behandelt Kap. 10. Ein weiteres, mit der Dokumentation zusammenhängendes Feature ist die Möglichkeit, Dokumente und Bilder im Modell zu integrieren. Dadurch kann die Dokumentation oder Spezifikation von Modellelementen dort abgelegt werden, wo sie benötigt werden. Dies erleichtert es später auch anderen Personen, die Modellinhalte zu verstehen.

Der *Innovator* ermöglicht eine dezentrale Entwicklung, da er Repository-basiert ist. Das Repository wird dabei von einem zentralen Modell-Server zur Verfügung gestellt, auf den die Modellierungsclients zugreifen. So können mehrere Personen gleichzeitig an verschiedenen Teilen des Modells arbeiten ohne sich in die Quere zu kommen. Bei anderen Tools hat man stattdessen Probleme mit der Integration verschiedener, dateibasierter Versionsstände, wenn mehrere Entwickler auf ein gemeinsames Modell zugreifen. Dadurch sind oft aufwändige Konsolidierungsphasen notwendig. Durch das Repository wird eben dies verhindert, so dass sich der *MID Innovator* ideal für große, verteilte Entwicklungsteams eignet.

Die Plattform stellt ferner eine Architektur zur Verfügung, die durch Konfiguration und Erweiterungsmöglichkeiten mit Skriptsprachen an die Projektbedürfnisse anpassbar ist. Es besteht beispielsweise die Möglichkeit zur Spracherweiterung mit UML2-Profilen, was den Einsatz von domänenspezifischen Sprachen ermöglicht. Durch Veränderung des Profils sind die Modellelemente vollständig an die Modellierungsanforderungen der Anwender anpassbar. Dies ist

insbesondere im Zusammenhang mit der Model Driven Architecture (MDA) der OMG von Bedeutung, für die sich der *Innovator* sehr gut eignet.

Über die Konfiguration von *Innovator* kann genau gesteuert werden, welche Berechtigungen jeder Modellierer hat. Dadurch lassen sich auch für verschiedene Rollen unterschiedliche Rechteumfänge realisieren. Profile lassen sich dabei auch exportieren und als Vorlage für Modelle verwenden beziehungsweise nachladen.

Darüber hinaus bietet der *Innovator* Programmierschnittstellen für Java und .NET an, die einen tiefen Zugriff auf die Plattform ermöglichen. Dadurch kann der Entwicklungsprozess auf die individuellen Anforderungen eines Projekts angepasst werden.

Für den *Innovator* gibt es zudem noch Anbindungen an die populären Entwicklungsumgebungen Eclipse und Visual Studio, mit denen ein Zugriff auf das Modell und auf generierte Source Code Dateien möglich ist.

Es besteht die Möglichkeit den *Innovator* durch Plug-Ins zu erweitern, die sich direkt in die Oberfläche einklinken. Sogenannte Engineering-Aktionen können zusätzlich erstellt werden. Sie eignen sich für die Automatisierung, sind aber meist nicht visuell in der Oberfläche sichtbar. Mit beiden lässt sich ein breites Spektrum an Ergänzung der Funktionalität der Plattform erzielen. Es gibt bereits vorgefertigte Plug-Ins für die SOA-Entwicklung, Text-to-Model Transformationen und für die SAP-Entwicklung. Das SAP-Plug-In gestattet den Import von Prozessen aus dem SAP Solution Manager in ein BPMN 2.0 Prozessmodell. Des Weiteren gibt es speziell für SAP ein eigenes Profil für M³, wodurch eine auf die SAP-Begrifflichkeiten ausgelegte DSL zur Modellierung bereit steht.

Insgesamt kann gesagt werden, dass der *Innovator* sehr flexibel eingesetzt werden kann und sehr mächtig ist, was allerdings auch eine gewisse Einarbeitung in das Tool und dessen Möglichkeiten voraussetzt.

AndroMDA

AndroMDA ist ein auf der Model Driven Architecture (MDA) basierendes Generator-Framework. Es entstand bereits 2002 in dem Vorläufer-Projekt UML2EJB und ist als Open-Source frei erhältlich [ANDRO]. Bei *AndroMDA* handelt es sich um ein ausgereiftes Rahmenwerk, das auch bereits in größeren Projekten eingesetzt wurde. Mit ihm lassen sich aus UML-Modellen unter anderem ganze Java-Projekte mit Klassen und weiteren Deployment-Artefakten erzeugen. Damit kann zum Beispiel eine Java Enterprise Anwendung mit grafischem User-Interface (GUI), Geschäftslogik und Datenhaltung modellgetrieben entwickelt werden. Das Framework unterstützt von Haus aus die Generierung von Anwendungen für drei Plattformen: Java Enterprise Edition, Spring und .NET. Da die Generierung aber flexibel angepasst werden kann, beschränken sich die Möglichkeiten nicht darauf. Die Entwicklung mit *AndroMDA* wird allerdings im Java-Umfeld am lohnenswertesten sein, da hierfür bereits viele Erweiterungen verfügbar sind. Laut den Angaben

auf der Projektseite berichten Anwender - abhängig vom Anwendungsteil - von Generierungsraten zwischen 30 und 100% (zum Beispiel bei der Persistenzschicht).

Bei der Entwicklung setzt AndroMDA auf ein Top-Down-Vorgehen. AndroMDA benutzt Apache Maven und ist damit eng integriert. Das Framework ist aber auch ohne Maven benutzbar. Die Verwendung von Maven vereinfacht die Entwicklung jedoch sehr und ist daher ratsam. Als erster Schritt wird im Allgemeinen mit Maven ein neues Projekt angelegt, welches bereits aus der notwendigen Grundstruktur und den Projekt-Dateien besteht. Die Konfiguration des Projektes kann dann den individuellen Bedürfnissen angepasst werden.

Als nächstes erfolgt die Modellierung der Anwendung mit einem separaten UML-Werkzeug, wie zum Beispiel MagicDraw oder Poseidon (beide werden ausdrücklich von AndroMDA unterstützt). In Verbindung mit AndroMDA kann prinzipiell jedes Tool eingesetzt werden. Das Modellierungswerkzeug muss dafür XMI-Dateien mit einem UML 1.4 Metamodell erzeugen können. Zudem muss es UML Tagged Values und UML Constraints unterstützen.

Die Modellierung der Anwendung geschieht mittels Klassen- und Aktivitätsdiagrammen. Mit den Klassendiagrammen können die Datenhaltungsschicht und die Geschäftslogik dargestellt werden. Anhand von Aktivitätsdiagrammen werden die Maskenflüsse (Pageflows) der GUI entwickelt. Bei der Modellierung sind allerdings noch einige Einschränkungen zu beachten. Der Entwickler muss sich auf bestimmte Stereotypen und eine für AndroMDA verständliche Struktur beschränken. Es können auch keine plattformabhängigen Datentypen, wie zum Beispiel `java.lang.Integer` verwendet werden. Dies würde dem Prinzip der MDA zuwiderlaufen, da die UML-Diagramme die Anwendung auf der Ebene des PIM (Platform Independent Model) modellieren. Ein speziell an AndroMDA angepasstes UML-Profil, das die Modellierung vereinfacht, ist bereits Teil der von Maven erzeugten XMI-Datei.

Im nächsten Schritt erfolgt der Import der UML-Modelle und die Generierung der Artefakte.

An dieser Stelle wird das PIM in ein PSM transformiert, von welchem der eigentliche Quellcode erzeugt wird. Der Code-Generator von AndroMDA verfolgt einen generischen Ansatz bei der Generierung. Die Details der Code-Erzeugung werden dabei in Modulen gekapselt, den sogenannten Cartridges. Die Cartridges bestehen ihrerseits aus Vorlagen für das zu erzeugende Artefakt (Templates) und weiteren Helferklassen (Metafacades), welche die Arbeit der Templates vereinfachen.

Die eigentliche Generierung von Artefakten übernehmen die Templates, indem sie auf das Modell zugreifen und mit den Modell-Informationen die Ausgabedateien erstellen. Für die Definition der Templates werden Velocity Skripts der gleichnamigen Template-Engine von Apache genutzt, die von AndroMDA zur Generierung eingesetzt wird. In manchen Fällen stoßen die einfachen Skripts der Templates allerdings an ihre Grenzen. Die Traversierung des Modells oder die Überprüfung des Modells auf bestimmte Bedingungen sind beispielsweise mit den Skripten nicht einfach Hand zu haben. Um solchen Situationen zu begegnen wurden die Metafacades eingeführt. Die Metafacades sind in Java programmiert und kapseln ihre

Funktionalität mit dem Facade-Entwurfsmuster (daher auch der Name). Sie stellen für die Templates eine API bereit, mit der auf das Modell und weitere, von der Metafacade berechnete Werte, zugegriffen werden kann. Sie dienen auch als Abstraktionsschicht, die die Templates von den Details des Metamodells abschirmt. Dadurch sind die Templates einfacher zu erstellen und übersichtlicher, da Details der Generierung in den Metafacades ausgelagert werden.

Die Generator-Engine lädt zunächst die UML-Modelle in den Speicher. Dazu benutzt AndroMDA das Netbeans MDR Metadata-Repository, eine Implementierung der MOF der OMG, um das UML-Model in einem Objekt-Baum darzustellen. Der Baum des Modells wird dann auf der Suche nach Klassen durchlaufen, welche mit den AndroMDA-spezifischen Stereotypen gekennzeichnet sind. Wird eine solche Klasse identifiziert, so wird für jeden erkannten Stereotyp die entsprechende Cartridge aufgerufen. In der Cartridge erzeugen dann die Templates unter Zuhilfenahme der Metafacades den Quelltext. Anzumerken ist, dass aus einer Klasse des Modells, je nach Anzahl der Stereotypen, durchaus mehrere Artefakte erzeugt werden können.

Für AndroMDA existiert bereits eine große Anzahl an implementierten Cartridges für unterschiedliche Artefakte. Dies ist ein Grund für die Attraktivität des Projekts, da viele Standardfälle im Java EE Bereich damit bereits abgedeckt sind und man sich um die Entwicklung projektspezifischer Details kümmern kann. Folgende Cartridges sind derzeit verfügbar:

- *BPM4Struts* – generiert Struts Webseiten aus Aktivitätsdiagrammen
- *EJB* – dient zur Generierung einer auf Container Managed EntityBeans basierenden Persistenzschicht und SessionBeans mit der Business Logik
- *Hibernate* – erzeugt eine mit Hibernate realisierte Datenhaltungsschicht, die ebenfalls optional mit aus SessionBeans bestehender Geschäftslogik erweitert werden kann
- *Java* – generiert POJO's aus Klassendiagrammen
- *jBPM* – erzeugt Prozessdefinitionen und -handler für JBoss jBPM
- *JSF* – generiert JSF Seiten aus Aktivitätsdiagrammen
- *Meta* – wird zur Erzeugung eigener oder der Anpassung bestehender Metafacades verwendet, die mit einem UML-Diagramm entworfen werden können
- *Spring* – generiert die Geschäftslogik und Persistenzschicht auf Basis der Spring Plattform
- *WebService* – erzeugt WSDL und WSDD (Axis Web Service Descriptor) für mit Apache Axis implementierte Webservices
- *XmlSchema* – generiert XML Schemata aus Klassendiagrammen

Der erzeugte Quelltext kann am Ende vom Entwickler von Hand weiter verfeinert, kompiliert, in Betrieb genommen und getestet werden. Mit AndroMDA sind iterative und inkrementelle Vorgehen grundsätzlich möglich. Zu beachten ist dabei allerdings, dass AndroMDA kein Reverse-Engineering beherrscht und daher nicht Roundtrip-fähig ist. Veränderungen müssen daher zuerst im Modell vollzogen werden, welches dann neu generiert wird. Quelltext, der bereits von

Hand angereichert wurde, kann bei der Neugenerierung zum Teil erhalten bleiben. Von den abstrakten Modell-Klassen abgeleitete Subklassen werden nicht verändert und behalten ihre Funktionalität. In anderen Fällen, wie beispielsweise Deployment-Deskriptoren, werden Änderungen aber nur teilweise übernommen.

AndroMDA ist ein ausgereiftes Framework für eine an der MDA orientierte Entwicklung. Es ermöglicht all jenen einen schnellen und unkomplizierten Einstieg, die einen Einblick in die Möglichkeiten der modellbasierten Entwicklung bekommen möchten. Da bereits viele Cartridges für häufig benötigte Funktionalität bestehen, können damit schnell Ergebnisse erzielt werden. Mit AndroMDA ist derzeit von Seiten der Open-Source Projekte der einfachste Zugang zur MDA möglich. Allerdings beschränkt sich die Entwicklung auf zwei Diagrammtypen und die Implementierung der Plattformspezifika erfolgt in Form von Cartridges und nicht in Modellen. Zudem können bestehende Anwendungen nur umständlich eingebunden werden, da die Entwicklung top-down erfolgt. Es eignet sich deswegen vor allem für neue Projekte von kleinem und mittlerem Projektumfang. AndroMDA ist aber aufgrund der Anpassbarkeit nicht darauf beschränkt. Bei einem größerem Umfang des Projektes bietet sich jedoch der Einsatz einer mächtigeren Plattform an, wie z. B. den Möglichkeiten des Eclipse Modeling Frameworks oder des MID *Innovators*. Dafür entfällt bei AndroMDA die Einarbeitung in die Komplexität eines solchen Frameworks. Weil sich die Cartridges von AndroMDA weitestgehend auf klassische Anwendungen im Java-Enterprise Umfeld konzentrieren, ist es zurzeit nur eingeschränkt in einer Serviceorientierten Anwendungslandschaft einsetzbar.

Eclipse

Die Eclipse Foundation unterstützt eine große Anzahl von Projekten Rund um das Thema Modellbasierte Softwareentwicklung. Ein erster Blick zeigt zunächst eine verwirrende Vielzahl an Frameworks: EMF, GMF, GEMS, GEF, Xtext. Dieser Abschnitt gibt einen Überblick über die wichtigsten Möglichkeiten zur modellbasierten Entwicklung mit Eclipse.

Aufgrund der vielen aufkommenden Ansätze wurden die gesamten Aktivitäten der Eclipse Foundation bezüglich der Modellierung in einem Projekt zusammengefasst. Das Eclipse Modeling Project verfolgt seitdem eine vereinheitlichte Strategie [EMP].

Den Kern der Modellierungsplattform bildet das *Eclipse Modeling Framework* (EMF) [EMF]. Dieses bildet die Basis für eine Vielzahl anderer Eclipse-Projekte. Die Eclipse Implementierung der XML Schema Definition (XSD) und das Web Tools Project (WTP) sind Beispiele hierfür. EMF entstand ursprünglich als Implementation der MOF der Object Management Group. Im Rahmen der Entwicklung des EMF wurden auch weitere für MDA notwendigen Modellierungsstandards der OMG, wie z. B. QVT, OCL oder XMI umgesetzt. Allerdings halten sich die Implementierungen zum Teil nicht allzu strikt an die Vorgaben der OMG.

Das Eclipse Modeling Framework besteht aus mehreren Bestandteilen. Die als Ecore bezeichnete Sprache ist das Meta-Meta-Modell des EMF und dient der

Beschreibung von Modellen. Es geht auf die MOF Implementierung zurück, welche jedoch im Verlauf der Entwicklung stark vereinfacht wurde und im Wesentlichen der EMOF entspricht. Die mit Ecore beschriebenen Modelle können mit einem in EMF enthaltenen Generator erzeugt werden. Dazu werden die sogenannten Java Emitter Templates (JET) verwendet, die in einer an JSP (Java Server Pages) angelehnten Sprache definiert werden. Mit JET werden Vorlagen für Java-Dateien angelegt, die bei der Generierung mit den Inhalten des Models gefüllt werden. Dabei beschränkt sich die Generierung mit JET nicht auf Java-Dateien. Es können prinzipiell alle Arten von Textdateien damit erzeugt werden, wie z. B. XML, HTML oder Properties-Dateien.

Zur Durchführung von Modelltransformationen gibt es mehrere Möglichkeiten. Einfachere Transformationen können ebenfalls mit den Java Emitter Templates definiert und durchgeführt werden. Für weitergehende Modelltransformationen wird die Atlas Transformation Language (ATL) verwendet.

Das Eclipse Modeling Framework enthält zudem weitere Funktionalitäten, um Veränderungen an Objekten im Modell zu beobachten und darauf zu reagieren, Objekte auszulesen und in Form von XMI zu serialisieren. Dies wird zum Beispiel von Editoren benutzt. Zudem erlaubt es die Bearbeitung und Veränderung eines Modells. Darüberhinaus gibt es eine Reihe von Subprojekten des Eclipse Modeling Frameworks für das Abfragen des Models (Model Query) und das Model-Management (Model Transaction). Die Integrität der ECore-Modelle kann mit einem Validierungsframework des EMF überprüft werden.

Um mit EMF eine Anwendung zu entwickeln ist es zunächst notwendig das zugehörige Modell zu spezifizieren. Hierfür gibt es drei vom Eclipse Modeling Framework unterstützte Wege: XMI-Dokumente, annotierte Java Interfaces oder ein XML Schema. Die Modell-Spezifikation wird importiert und kann z. B. mit einem Editor noch verfeinert werden. Aus dem Modell können dann Java-Klassen generiert werden. Zusätzlich gibt es noch von Drittherstellern weitere Generatoren für andere Artefakte, die nicht Teil des EMF sind. Die mit dem EMF erzeugten Java-Klassen können vom Entwickler erweitert und verändert werden. Hält er sich dabei an gewisse Konventionen, so werden diese Änderungen auch bei der erneuten Erzeugung nach einer Modelländerung übernommen.

Das Eclipse Modeling Framework bietet insgesamt eine fundierte Basis für die modellbasierte Entwicklung. Es enthält allerdings keinerlei graphische Toolunterstützung. Um ein Modell in einem UML-Diagramm zu spezifizieren, wird beispielsweise ein zusätzlicher Editor verwendet und das Ergebnis per XMI exportiert. Hierfür bieten sich von kommerzieller Seite z. B. Omondo EclipseUML oder der IBM Rational Software Modeler an. Es können aber auch Open-Source-Modellierungstools, wie das auf EMF und Eclipse aufbauende TOPCASED oder das noch in Entwicklung befindliche Eclipse-Projekt UML2Tools verwendet werden. Wichtig ist nur, dass das UML-Tool den XMI-Export unterstützt.

Das *Graphical Modeling Framework (GMF)* ist ein weiterer Bestandteil des Eclipse Modeling Projekts. Mit GMF ist es möglich grafische Editoren aus auf EMF aufbauenden Modellen zu generieren. Dazu werden die Anwendungsdomäne, die Diagramm-Elemente, die Werkzeuge zur Manipulation des Diagramms und

das Mapping zwischen den drei genannten Modellen modelliert. Aus den Modellen wird dann ein in Eclipse einsetzbarer Diagramm-Editor generiert, der bereits über Basis-Funktionalitäten und Komponenten verfügt. Die spezifischen Details der Tools können dann vom Implementierenden umgesetzt werden.

Xtext und *Xpand* runden die Möglichkeiten zur modellbasierten Softwareentwicklung von Eclipse in Richtung domänenspezifischer Sprachen (DSL) ab. Bei *Xpand* handelt es sich um eine Template-Sprache die z. B. für die Generierung benutzt werden kann. *Xtext* ist ein Framework für die Entwicklung von DSL. Beide sind in dem Open-Source Projekt „openArchitectureWare“ (oAW) entstanden, welches Teil vom Eclipse Modeling Project wurde. Näheres zu *Xpand* und *Xtext* findet sich im nächsten Abschnitt zu oAW.

Insgesamt erkennt man, dass es rund um Eclipse eine Vielzahl an Modellierungstools und Frameworks gibt. Die oben aufgeführten Beispiele sind dabei nur eine Auswahl der wichtigsten Projekte. Der Versuch, aus den einzelnen Bestandteilen eine Basis für die Entwicklung zu erstellen, gleicht einem Puzzlespiel. Dies erschwert den Einstieg in die Modellierung, sofern noch keine Grundkenntnisse vorhanden sind. Es bietet aber all denen Vorteile, die ihre Entwicklung an individuelle Bedürfnisse anpassen möchten. Leider ist die graphische Modellierung derzeit noch ein Schwachpunkt bei Eclipse. Es existieren zwar viele kommerzielle Angebote für einzelne Sprachen, wie z. B. UML, aber nur wenige integrierte Lösungen, wie sie zum Beispiel der *Innovator* darstellt. Die verfügbaren Open-Source Tools sind von unterschiedlicher Qualität. Für die Erstellung und Bearbeitung textueller Modelle, wie sie im Rahmen von domainspezifischen Sprachen (DSL) verwendet werden eignet sich Eclipse jedoch hervorragend. Trotz mancher Schwächen bieten die in Eclipse verfügbaren Technologien eine solide Grundlage für die modellbasierte Entwicklung, die bereits heute alle wesentlichen Aspekte abdeckt.

openArchitectureWare

Nachdem wir bereits einen Blick auf Werkzeuge gerichtet haben, mit denen Modelle erstellt und Code generiert werden kann, gehen wir in diesem Abschnitt genauer auf die Möglichkeiten zur Modell-zu-Text Transformation ein. Im *Innovator* wird hierfür unter anderem *openArchitectureWare* (oAW) genutzt. Da oAW inzwischen Teil des *Eclipse Modeling Projekts* ist, kann die folgende Einführung auch für Projekte auf Grundlage der Eclipse-basierenden Modellierungs-Technologien verwendet werden. Betrachten wir zunächst was unter openArchitectureWare zu verstehen ist.

Überblick über openArchitectureWare

Bei openArchitectureWare²³ handelt es sich um ein Framework für die modellbasierte Entwicklung, dass aus einem Open Source Projekt entstanden ist. Das Rahmenwerk ist mit der Eclipse Public License veröffentlicht und frei erhältlich.

²³<http://www.openarchitectureware.org/>

Im Jahr 2009 wurden die Kernkomponenten von oAW Teil des Eclipse Modeling Project. Seitdem findet die Entwicklung des Frameworks unter dem Dach der Eclipse Foundation statt. Naheliegender Weise glänzt openArchitectureWare daher durch eine sehr gute Integration in Eclipse, so dass für verschiedene Entwicklungsschritte bereits leistungsfähige Plug-Ins vorhanden sind.

Im Kern besteht oAW aus einer Familie von Sprachen, die für alle Aspekte der modellbasierten Entwicklung gebraucht werden sowie der Modeling Workflow Engine.

Die Modellierungssprachen dienen der Überprüfung und Transformation des Modells, sowie der Erzeugung von Artefakten aus dem Modell. Bei der Sprache *Check* handelt es sich um eine Validierungssprache, mit der, ähnlich zur OCL der OMG, Modelle auf bestimmte Spezifikationen und Einschränkungen geprüft werden können [Check].

Mit der Sprache *Xpand*²⁴ enthält openArchitectureWare eine Template-Sprache, welche die Spezifikation von Generierungsvorlagen erlaubt [XPAND]. Damit lässt sich also definieren, wie Quelltext-Artefakte erzeugt werden sollen. Xpand beschränkt sich allerdings nicht auf die reine Code-Generierung, sondern kann auch zur Modelltransformation eingesetzt werden. In diesem Buch werden zum Beispiel in Kap. 9 „Automatisierung“ oAW Templates mit Xpand definiert, aus denen dann Quelltext generiert wird.

Mit der Sprache *Xtend* kann das Framework um zusätzliche Logik erweitert werden, die in anderen Sprachen, wie zum Beispiel Check oder Xpand, implementiert ist [XTEND]. Beispielsweise können damit Operationen erstellt werden, die auf dem Modell ausgeführt werden, ebenso wie ganze Bibliotheken von Erweiterungen. Die Erweiterungen basieren entweder auf oAW Ausdrücken oder können in Java implementiert sein.

Mit der Modeling Workflow Engine können die einzelnen Aktivitäten während der Generierung oder Transformation von Modellen orchestriert werden. Die Aktivitäten werden dabei als Komponenten bezeichnet und können zum Beispiel Modelltransformationen, Modellvalidationen oder Code-Generatoren sein. Für viele Einsatzzwecke, wie zum Beispiel dem Auslesen von Modellelementen oder der Generierung von Source Code, bestehen bereits bei oAW mitgelieferte Workflow-Komponenten. Bei der Auswahl der Komponenten ist man aber nicht auf die von openArchitectureWare vorgegebenen Workflow-Komponenten beschränkt. Vielmehr können eigene Komponenten entwickelt und eingebunden werden. Der Ablauf der Generierung wird in einer XML-Datei beschrieben, welche die Modeling Workflow Engine steuert.

Um mit oAW aus einem Modell Source-Code zu erzeugen, sind im einfachsten Fall drei Schritte notwendig:

1. Einen Workflow erstellen, indem die Abläufe in einer XML-Datei beschrieben werden.

²⁴<http://wiki.eclipse.org/Xpand>

2. Die Xpand Templates erstellen, die das Gerüst der zu generierenden Source-Dateien darstellen.
3. Die Generierung ausführen.

Ein herausragendes Feature von oAW ist die breite Unterstützung von Modellen. Mit OpenArchitectureWare können beinahe alle mit UML-Werkzeugen erstellten Modelle geparkt werden. Darüberhinaus unterstützt oAW alle EMF Modelle. Es können sogar mit Microsoft Visio erstellte Modelle oder textbasierte Modelle verwendet werden.

Aus den Modellen kann Source-Code einer beliebigen Sprache erzeugt werden. Mit Xtext verfügt das Framework sogar über die Möglichkeit eigene domänenspezifische Sprachen (DSL) und die dazugehörigen sprachspezifischen Werkzeuge zu erstellen.

OpenArchitectureWare und Xpand sind eng in den *Innovator* eingebunden. Es existiert hierfür ein spezieller Adapter, der über die Java-API des *Innovators* einen Zugriff auf Innovator-Modelle erlaubt. Da das Generator-Metamodell eng mit dem *Innovator* gekoppelt ist, kann die Generierung sehr performant ausgeführt werden. Somit kann man auch mit dem *Innovator* die volle Bandbreite der Möglichkeiten von oAW nutzen.

Zusammenfassend ermöglicht openArchitectureWare ein breites Spektrum an Aktivitäten der modellbasierten Softwareentwicklung:

- Das Einlesen und die Initialisierung von Modellen
- Modellevaluation und die Verifikation des Modells auf Verletzungen von Constraints
- Modell-zu-Modell-Transformationen (M2M-Transformationen)
- Modell-zu-Code-Transformationen (Generierung)
- Text-zu-Modell-Transformationen
- Die Definition von DSLs und die Generierung entsprechender Werkzeuge

Webservice-Technologien

WSDL

Die Web Service Description Language (WSDL) ist eine abstrakte Sprache zur Beschreibung von Dienstschnittstellen und Dienst Anbietern [WSDL], die vom World Wide Web-Konsortium (W3C) verwaltet wird. Sie basiert auf XML und ist unabhängig von einer bestimmten Technologie verwendbar. Derzeit findet die Version 1.1 verbreitet Anwendung, auf welche im Folgenden Bezug genommen wird. Die derzeit aktuelle Version 2.0 verwendet eine abweichende Terminologie. Die WSDL beschreibt einen Webservice in zwei Teilen. Die Aufteilung erfolgte im Hinblick auf eine Wiederverwendung der beiden Teile. Somit kann ein Service beispielsweise mit derselben Schnittstelle von mehreren Endpunkten realisiert werden ohne dass diese jedesmal erneut definiert werden müsste. Den Aufbau einer WSDL-Datei illustriert Abb. 2.19.

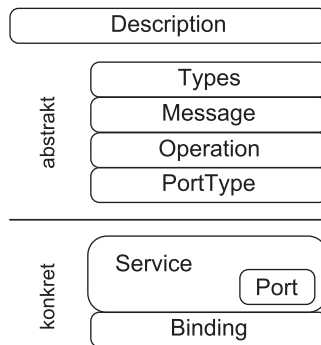


Abb 2.19 WSDL-Komponenten

Im abstrakten Teil beschreibt die WSDL die Funktionalität des Dienstes. Hierzu werden zunächst die verwendeten Datentypen (*Types*) definiert. Dies geschieht meist in Form von XML Schemata. Darüberhinaus ist es über den Erweiterungsmechanismus von WSDL möglich, auch andere Formate zur Beschreibung zu verwenden. Diese sind aber ungebräuchlich, so dass gegenwärtig in Webservice-Beschreibungen nur XSD vorkommt. Zudem können externe Definitionen über das Element *Import* eingebunden werden. Mit dem *Description*-Element kann der Service in Textform beschrieben werden.

Die Daten, die zwischen Dienstanutzer und Dienstanbieter ausgetauscht werden, können in Form von XML-Nachrichten (*Message*) festgelegt werden. Sie bestehen aus mehreren Teilen (*Parts*), welche die Parameter darstellen. Die Parts sind durch einen der Datentypen typisiert. Die Bedeutung der Parts steht auch mit der Bindung im Zusammenhang. Es ergeben sich beispielsweise bei Verwendung des SOAP Document-Style RPC an dieser Stelle Einschränkungen.

Bis zu zwei Nachrichten werden zu einer *Operation* zusammengefasst. Eine Operation entspricht in der Terminologie der Objektorientierten Programmierung der Signatur einer Methode. Jede Nachricht kann dabei entweder als eingehende (Input) oder als ausgehende Nachricht (Output) definiert werden. Damit ergeben sich vier von WSDL 1.1 unterstützte Kombinationen von Nachrichten für eine Operation, die ab Version 2.0 auch als Message Exchange Patterns bezeichnet werden:

- *One-way*: Der Dienst empfängt eine eingehende Nachricht
- *Request-response*: Der Dienst empfängt eine eingehende Nachricht und sendet eine ausgehende Nachricht als Antwort
- *Solicit-response*: Der Dienst sendet zuerst eine ausgehende Nachricht und empfängt dann eine eingehende Nachricht
- *Notification*: Der Dienst sendet eine ausgehende Nachricht

Allerdings gibt es für die letzten zwei Patterns in WSDL 1.1 kein Standard Binding. Um diese zu nutzen, ist eine eigenentwickelte Erweiterung notwendig. Zusätzlich zu den Operationen können des Weiteren noch Fehlernachrichten (*Faults*) definiert werden. Da diese nicht bei jeder Operation zwingend benötigt werden, ist ihre Verwendung optional. Die Operationen und Faults eines Dienstes werden

schließlich in einem sogenannten *Port-Type* zu einer Schnittstelle zusammengefasst. In Java entspricht dies einem Interface.

Der konkrete Teil dient der Beschreibung der Lokalisation und der Art und Weise in der ein Dienst angeboten und genutzt wird. Mit dem *Service*-Element findet eine logische Gruppierung aller Ports statt, die ein Dienst anbietet. Ein *Port* ist ein Bestandteil des *Service*-Elements, der die Details der Adressierung eines *Service*-Endpunkts (Endpoint) beschreibt. Er legt mit einer Netzwerkadresse den Ort fest, an dem ein Dienst erreicht werden kann. Es ist durchaus möglich, denselben Dienst von mehreren Endpunkten aus anzubieten und in demselben WSDL-Dokument zu beschreiben.

Im *Binding*-Element werden schließlich das Nachrichtenformat und die Protokolldetails für Operationen einer Schnittstelle (*Port-Type*) definiert, die zum Nachrichtenaustausch verwendet werden. Es legt also fest, auf welche Art die Daten kodiert und transportiert werden. WSDL 1.1 unterstützt dabei mit SOAP 1.1, HTTP und MIME drei Arten der Bindung. Die gebräuchlichste Form für Webservices ist dabei SOAP (Simple Object Access Protocol) [SOAP]. Für REST-Webservices eignet sich das HTTP-Binding, wobei in Version 1.1 nur die HTTP-Methoden GET und POST unterstützt werden.

Änderungen in WSDL 2.0

In der Nachfolgeversion von WSDL 1.1 wurden einige Verbesserungen der Verständlichkeit und Wiederverwendbarkeit durchgeführt. Unter anderem wurde ein neues Komponenten-Modell eingeführt, dass alle Elemente der WSDL beinhaltet. Für das *Port-Type* Element der Version 1.1 wurde die treffendere Bezeichnung *Interface* eingeführt und der *Port* wird als *Endpoint* bezeichnet. Zudem wurde mit der Entfernung der *Messages* und *Parts* die Beschreibung der Operation vereinfacht. Die Definition der Nachrichten erfolgt in WSDL 2.0 in Form eines *Types*, wobei die *Parts* nicht mehr existieren.

Bei der Beschreibung der Serviceschnittstellen ergaben sich einige Änderungen. Zum Beispiel wird nun die Vererbung von Interfaces unterstützt. Die Kommunikationsmuster werden in Version 2.0 als *Message Exchange Pattern* bezeichnet und von vier auf acht erweitert. Neu ist außerdem, dass ein *Service* nur ein einziges Interface umsetzen darf. In WSDL 1.1 besteht diese Einschränkung nicht.

Hinsichtlich des Bindings wurden einige Verbesserungen eingeführt. So wurde beispielsweise die Beschreibung des SOAP-Bindings vereinfacht. Zudem wird SOAP 1.2 unterstützt. Eine bessere Unterstützung für REST ist ebenfalls Teil von WSDL 2.0. Es werden nun alle HTTP-Methoden unterstützt.

WS-BPEL

Die Web Services Business Process Execution Language 2.0 (kurz: WS-BPEL oder BPEL) ist eine Prozessdefinitionssprache der OASIS (Organization for the Advancement of Structured Information Standards). Sie kann zur Komposition

von Webservices in Prozesse verwendet werden [BPEL]. Dies ist insbesondere im Rahmen des Geschäftsprozess-Managements (BPM) von Bedeutung.

Prinzipiell lassen sich mit BPEL durch die fachliche Seite eines Unternehmens Geschäftsprozesse aus Diensten modellieren, welches einen zentralen Gedanken einer SOA darstellt. Solche zusammengesetzte Dienste bezeichnet man auch als Composite-Services. Die modellierten Prozessdefinitionen können dann in einer BPEL-Laufzeitumgebung (Workflow-Engine) zur Steuerung von Diensten eingesetzt werden. Mit BPEL wird so eine Trennung der Ablauflogik und -steuerung erzielt, wobei die Logik in den Diensten gekapselt wird.

Im Rahmen der Steuerung einer Komposition von Webservices wird zwischen der sogenannten „Orchestrierung“ und der „Choreografie“ unterschieden. Bei der Orchestrierung steuert eine Instanz eines dedizierten, zentralen Dienstes den Ablauf aller Aktivitäten *eines* Prozesses. Im Gegensatz dazu bezeichnet der Begriff der „Choreografie“ das Zusammenspiel *mehrerer* Prozesse, welche sich sequentiell aufrufen. Man kann Choreographie auch als das Zusammenspiel mehrerer Geschäftsprozesse betrachten, während Orchestrierung die Abläufe eines Prozesses festlegt. Mit WS-BPEL ist sowohl eine Orchestrierung, wie auch eine Choreographie möglich.

Im Folgenden soll ein kurzer Überblick über eine Teilmenge von WS-BPEL gegeben werden, der für das Verständnis des Beispiels des Buches notwendig ist.

Im Gegensatz zu BPMN ist BPEL keine graphische Modellierungssprache. Es gibt hierfür auch keine standardisierte Notation. Die Prozessdefinition erfolgt in BPEL vielmehr anhand von Dokumenten. Diese beschreiben die Abläufe in einer an imperative Programmiersprachen angelehnten Sprache. Zur Bearbeitung eines BPEL-Dokuments stellen die gängigen Editoren zwei Sichten zur Verfügung: Eine graphische Modellierungsansicht und eine detaillierte textuelle Ansicht, welche die Bearbeitung des BPEL Quellcodes erlaubt. Die graphische Notation eines beispielhaften BPEL-Prozesses in Eclipse zeigt Abb. 2.20.

Das Wurzelement eines BPEL-Dokuments ist der Process-Tag, der als äußerster Container der Prozessdefinition dient. Er enthält die Definition der Imports, Partnerverbindungen, Variablen und der Prozesslogik. Als Attribute müssen mindestens der Name und der Target-Namespace angegeben werden. Neben den erwähnten Elementen können noch weitere fortgeschrittene Tags enthalten sein, wie z. B. für Faulthandler und Message-Exchanges. Jeder Prozess muss darüberhinaus mindestens eine Aktivität enthalten (z. B. Sequence). Die folgende Grafik 2.21 veranschaulicht die vereinfachte Struktur eines BPEL-Dokuments.

Mittels des *Import*-Elements können externe XML-Schema oder WSDL Definitionen referenziert und importiert werden. Die importierten Definitionen stehen dem gesamten Prozess zur Verfügung. Externe Dienste können so anhand von WSDL-Beschreibungen eingebunden werden. Die ausführbaren BPEL-Prozesse stellen ebenfalls Dienste dar, die über Webservices ansprechbar sind. Ihre externe Schnittstelle wird gleichfalls in WSDL beschrieben.

Nach dem Konzept von WS-BPEL stellt die Interaktion von Prozessen und anderen Diensten eine Zusammenarbeit von Geschäftspartnern dar. Externe Services und BPEL-Prozesse werden daher als Partner bezeichnet, deren Kommunikation in spezifischen Rollen durchgeführt wird. Dabei stehen die Dienste, die den

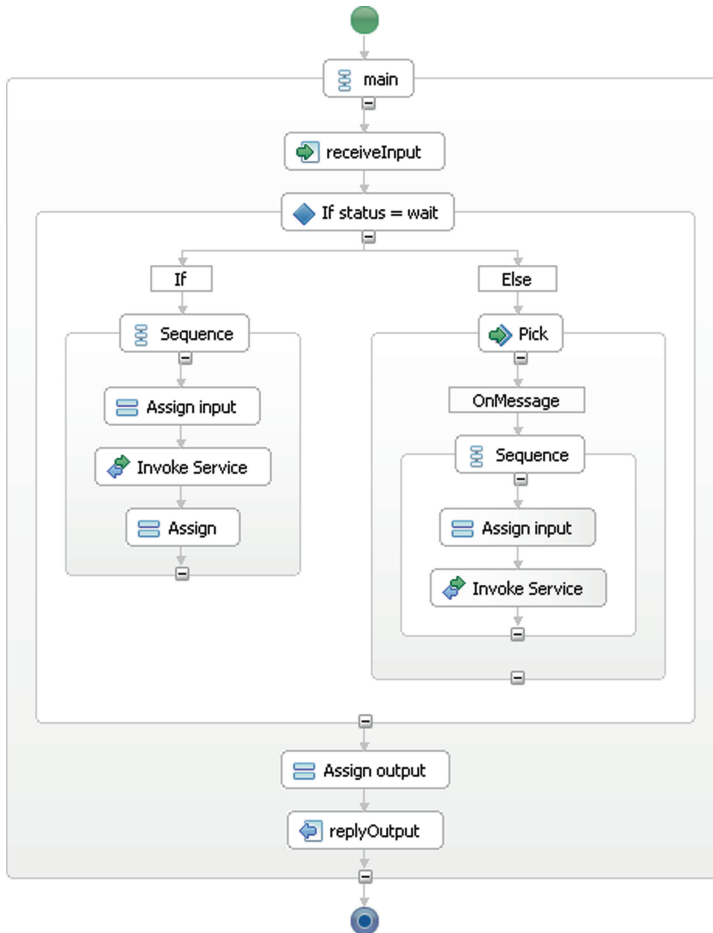


Abb. 2.20 Visualisierung eines einfachen BPEL-Ablaufs in Eclipse

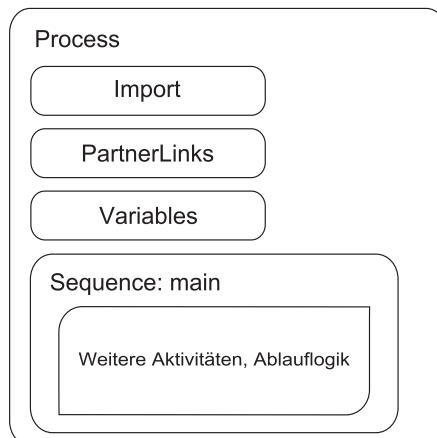


Abb. 2.21 Vereinfachte Struktur eines BPEL-Dokuments

Geschäftsprozess nutzende und der Prozess ebenfalls in einer partnerschaftlichen Verbindung. Eine Rolle ist eine innerhalb der Verbindung angebotene oder genutzte Schnittstelle. Die Eigenschaften einer Partnerbeziehung werden in Partnerlinks und Partnerlink-Types beschrieben.

Ein *Partnerlink-Type* legt fest, welche Rolle durch die Schnittstelle einer Partnerverbindung abgebildet wird. Zu jeder Rolle wird hierzu ein WSDL-PortType angegeben, der die Rolle repräsentiert. Es findet an dieser Stelle also die Definition eines Servicevertrages statt. Der BPEL-Prozess findet hierdurch die zu verwendenden Schnittstellen. Der Partnerlink-Type ist nicht Teil eines BPEL-Dokuments. Vielmehr stellt er eine Erweiterung von WSDL dar, die entweder in einem WSDL-Dokument oder einer externen Datei untergebracht werden kann.

Ein *Partnerlink* ist ein Element von BPEL, welches angibt, wer die Partner einer Interaktion sind, welche Rolle durch den Prozess und welche durch seine Partner verwirklicht wird. Hierzu wird spezifiziert, welcher PartnerLink-Typ für die Verbindung gilt. Zudem muss mindestens die Rolle, die der Prozess übernimmt (*myRole*) oder die Rolle des Gegenübers (*partnerRole*) angegeben werden. Abbildung 2.22 veranschaulicht die Zusammenhänge.

Zum Speichern und Verarbeiten von Daten, sowie zur Repräsentation des Zustands eines Prozesses stehen in WS-BPEL Variablen zur Verfügung. Die Variablen werden in dem *Variable*-Tag deklariert. Der BPEL Standard knüpft eng an WSDL an und benutzt ebenfalls XSD zur Definition der Datentypen. Variablen können mit XML-Schema-Typen (Simple oder Complex-Types), XML-Schema-Elementen oder WSDL-Message-Typen typisiert sein. Die Zuweisung von Werten erfolgt mit der Assign-Aktivität (siehe unten). Zur Abfrage von Variablenwerten wird XPath 1.0 verwendet [XPATH]. Die Gültigkeit ist bei Variablen grundsätzlich global, sofern sie nicht innerhalb eines *Scope*-Elements definiert werden. Darüber hinaus kann der Scope-Tag den Gültigkeitsbereich von anderen, in ihm eingeschlossenen Elementen definieren.

Zur Strukturierung von Prozessen dienen in BPEL Aktivitäten. Man unterscheidet dabei zwischen Basisaktivitäten und strukturierten Aktivitäten. Die Basisaktivitäten stellen elementare Einheiten eines Prozesses dar:

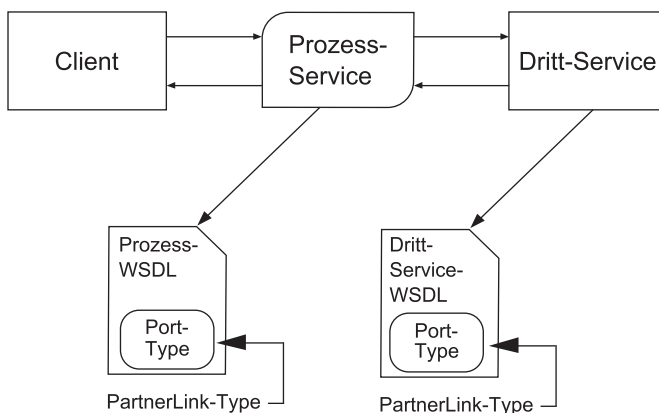


Abb. 2.22 Beziehungen der Partnerlinks eines BPEL-Dokuments

- *Invoke*: Ermöglicht den Aufruf eines Dienstes durch den BPEL-Prozess
- *Receive und Reply*: *Receive* dient dem Nachrichtenempfang durch den Prozess, *Reply* wird zum Versand einer Antwortnachricht verwendet
- *Assign*: Realisiert die Zuweisung von Werten und die Aktualisierung von Variablen
- *Throw und Rethrow*: Sind Teil der Ereignisbehandlung. *Throw* erzeugt einen Fault (vgl. Exceptions, Ausnahmefall des Geschäftsprozesses), *Rethrow* wird zur Weitergabe eines Faults verwendet, der durch einen Fault-Handler (Fehlerbehandlung) gefangen wurde
- *Wait*: Pausiert den Prozess für einen angegebenen Zeitraum oder bis zu einem speziellen Zeitpunkt
- *Empty*: Entspricht einer „No-Operation“, es wird also nichts ausgeführt
- *Exit*: Beendet eine Prozess-Instanz unverzüglich
- *Extension-Activity*: Kann zur Erweiterung von BPEL um eine neue Aktivität verwendet werden

Die Aufgabe der strukturierten Aktivitäten ist die Flusssteuerung anderer Aktivitäten. Sie ermöglichen somit die Darstellung von Prozessstrukturen. Dabei steht wiederum XPath 1.0 zur Auswertung von Ausdrücken zur Verfügung. Folgende strukturierte Aktivitäten stehen in BPEL zur Verfügung:

- *Sequence und Flow*: Zur Definition von Abläufen werden die Aktivitäten „sequence“ und „flow“ benutzt. Dabei definiert *Sequence* seriell verarbeitete und *Flow* parallel ablaufende Aktivitäten
- *If*: Wählt eine Aktivität zur nachfolgenden Ausführung aus (z. B. eine Sequenz)
- *While*: Führt eine Aktivität solange aus, wie eine spezifizierte Bedingung wahr ist
- *Repeat-Until*: Führt eine Aktivität solange aus, bis eine spezifizierte Bedingung wahr wird
- *Pick*: Wartet auf das Eintreffen einer von mehreren Nachrichten oder das Auftreten eines Timeouts und unterbricht den Prozess solange. Die „pick“ Aktivität ermöglicht nicht-deterministische Verzweigungen aufgrund von externen Nachrichten
- *For-each*: Führt eine Aktivität n+1 mal aus, wobei die Iterationen parallel ausgeführt werden können

Zum Sprachumfang von WS-BPEL gehören des Weiteren Möglichkeiten zur Ereignisbehandlung, die mit dem Element *Event-Handler* definiert werden. Dabei kann mit *On-Message* auf das Eintreffen einer Nachricht und mit *On-Alarm* auf das Auftreten eines zeitlichen Ereignisses reagiert werden. Analog zu einem Catch-Block in Java können in BPEL mit dem *Fault-Handler*-Tag zudem Abschnitte zur Fehlerbehandlung definiert werden. Ein interessantes Sprachfeature ist ein sogenannter *Compensation-Handler*, mit dem sich vorher durchgeführte Aktivitäten zurückrollen lassen. Auf diese Weise verfügt WS-BPEL über eine Form von Transaktionalität.

Darüberhinaus existieren noch weitere Elemente, die zur Korrelation von Nachrichten genutzt werden können, wie es z. B. bei einer Session notwendig ist. Somit

kann unter anderem sichergestellt werden, dass die richtige Instanz eines Prozesses adressiert wird. Dafür können sogenannte *Properties* definiert werden, die bestimmten Feldern der Nutzdaten einer Nachricht entsprechen (z. B.: SessionID). In einem *Correlation-Set* werden anschließend diejenigen Properties gruppiert, die eine Konversation identifizieren. Mittels des *Correlation*-Elements wird festgelegt welche Korrelation für die betreffende Nachricht gilt. Das *Correlation*-Tag kann in Verbindung mit den Kommunikationsaktivitäten Receive, Reply, On-Message (als Teil von Pick) und Invoke verwendet werden. Die Workflow-Engine findet anhand der Daten einer Korrelation dann die dazugehörige Prozess-Instanz.

Die SOA Plattform

Nachdem wir in den vorangehenden Abschnitten die Sprachen und Werkzeuge betrachtet haben, die für die Entwicklung einer SOA genutzt werden können, gehen wir in diesem Abschnitt auf die Plattform ein, die als Basis für die Beispielanwendung dient.

Als SOA Plattform versteht man die Tools und Infrastruktur, die für den Betrieb einer Serviceorientierten Architektur notwendig sind. Die Plattform dient zur Integration der einzelnen Dienste zu einem Ganzen. Zumeist besteht eine SOA Plattform aus einem Enterprise Service Bus (ESB), einer Service Registry und einer Komponente zur Prozesssteuerung und Service-Orchestrierung. Zudem bilden Tools für Entwicklung und Inbetriebnahme (Deployment) der Dienste einen Teil einer Plattform. Verschiedene Hersteller stellen darüberhinaus weitere Werkzeuge zur Verwaltung und Überwachung der Services, sowie der Integration von Produkten Dritter zur Verfügung.

Die im Weiteren beschriebene SOA Plattform ist die Zielplattform für die Generierung von Artefakten im Rahmen der modellbasierten Entwicklung der Beispielanwendung des Buches. Sie bildet auch die Grundlage für die Demonstrations-Anwendung, die auf der Website des Buches heruntergeladen werden kann. Dabei reflektiert die Plattform die Heterogenität, die durch den Einsatz unterschiedlicher Technologien und der Produkte verschiedener Hersteller heute in vielen Unternehmen eine Tatsache ist. Dem SOA-Ansatz folgend, zeigt das Beispiel, wie eine SOA zur Integration verschiedener Service-Anbieter und Technologien führen kann.

Für die Realisierung einer Serviceorientierten Architektur stehen mittlerweile viele Möglichkeiten zur Verfügung. Neben den Plattformen kommerzieller Anbietern, wie z.B. SAP, IBM oder Oracle, gibt es heute auch SOA Plattformen vergleichbarer Funktionalität aus dem Open-Source-Bereich. Hier sind beispielsweise Mule-ESB, OpenESB, SOPERA oder die JBoss Enterprise SOA Plattform zu nennen. Angesichts der verwirrenden Vielfalt stellt sich die Frage welche Plattform nun in Frage kommt. Da das Beispiel des Buchs dem Leser auf unserer Website zum Download zur Verfügung stehen sollte, kamen nur Open-Source Lösungen in Betracht.

Die Plattform des Buches besteht daher aus drei Bestandteilen. Die eigentliche SOA-Plattform bildet die Advanced Service Factory von *SOPERA*, die für die neu

zu entwickelnden, internen Services genutzt wird. Externe Dienste und Altsysteme werden in einem JBoss Application Server simuliert. Die Prozessautomatisierung und –orchestrierung mit BPEL übernimmt ein Apache ODE Server, der im JBoss betrieben wird. Zur Einbindung von Human Tasks wird ein Webinterface genutzt, welches ebenfalls im JBoss AS gehostet wird.

SOPERA

Bei SOPERA handelt es sich um die seit 1999 entwickelte SOA-Plattform der Deutschen Post AG [SOPERA]. Die Firma wurde im November 2010 von Talend übernommen. SOPERA ist seit 2007 in einer „Community Edition“ als Open Source veröffentlicht. Daneben existiert eine kommerzielle „Enterprise Edition“, welche darüberhinaus proprietäre Erweiterungen und Support-Dienstleistungen bietet.

Die Plattform von SOPERA wurde nach dem Best-Of-Breed-Ansatz entwickelt und setzt konsequent auf die Benutzung von Standards. Aufgrund der langjährigen Benutzung im Unternehmen ist das Rahmenwerk sehr ausgereift. Die Plattform wird als Advanced Service Factory (ASF) bezeichnet und ist derzeit in Version 3.3 erhältlich. Sie besteht aus vier Teilen: Einem Enterprise Service Bus (Service Backbone, SBB), den technischen Serviceteilnehmern (Technical Service Participants, TSP), einer Entwicklungsumgebung (ASF Toolsuite) und einer Programmierschnittstelle (Participant-API, PAPI). Abbildung 2.23 gibt einen Überblick über die Plattform.

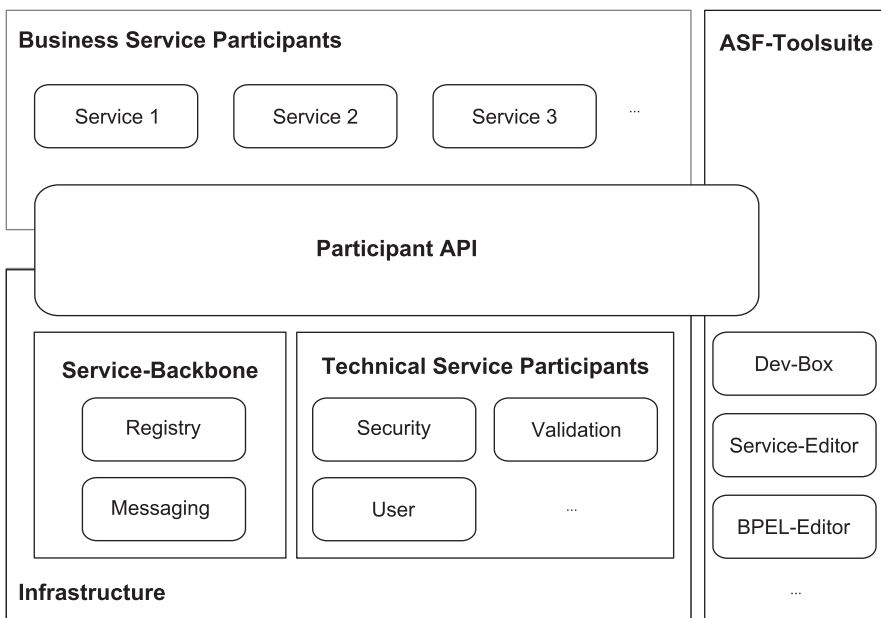


Abb. 2.23 Übersicht über die SOPERA-Plattform

Der Service-Backbone liefert in Form eines verteilten Enterprise Service Bus die zentrale Integrationsinfrastruktur von SOPERA. Er gliedert sich in zwei Bestandteile, die jeweils in Apache-Tomcat²⁵ Webcontainern betrieben werden: Ein Service-Repository (Registry) und einer Message-Queueing Komponente (Messaging). Mit Version 3.3.1 kam die Möglichkeit hinzu, alle TSP in einem Tomcat zu betreiben. Zusätzlich können weitere technische Dienste, in Form von sogenannten Plug-Ins, die Basisdienste erweitern. Seit Version 3.3 werden die kommerziellen Erweiterungen auch als Optionpacks bezeichnet. Zu nennen wären beispielsweise ein Prozess Management mit BPEL und BPMN (SOPERA BPM, basierend auf Intalio|BPM) oder ein Systemmanagement Plug-In (SOPERA HQ, basierend auf SpringSource Hyperic HQ).

In der Service-Registry wird die Syntax der Serviceschnittstelle in Verbindung mit Meta-Informationen über den Dienst verwaltet. Dies können zum Beispiel Daten zur Sicherheitskonfiguration, Bindung, Versionierung oder der Service Level Agreements (SLA, Policies) sein. Die Dienste werden anhand logischer Adressen angesprochen, deren konkrete Bindung dynamisch zur Laufzeit durch die Registry aufgelöst wird.

Bei der Messaging-Komponente handelt es sich um eine Nachrichtenorientierte Middleware (MOM), die durch einen Java Messaging Dienst (JMS) zur Verfügung gestellt wird. Es handelt sich dabei prinzipiell um einen Message Broker, der Transport, Vermittlung und Transformation der SOAP-Nachrichten übernimmt. Insbesondere wird der asynchrone und persistente Nachrichtenaustausch unterstützt. Die Kommunikationsmöglichkeiten von SOPERA-Diensten sind daher vielfältiger als bei herkömmlichen Webservices.

Die Technical Service Participants stellen klar definierte Integrationsfunktionen für die Infrastrukturkomponenten zur Verfügung. Dies können beispielsweise Sicherheitsdienste, Benutzerverwaltung oder Nachrichtenvalidierung sein. Die Teilnehmer-Dienste, welche die Geschäftslogik kapseln, werden als Business Service Participants (BSP) bezeichnet. Der Betrieb der TSP und die BSP erfolgt in der Regel jeweils in eigenen Webcontainern. Die Skalierbarkeit der Plattform ist daher gegeben.

Die ASF Toolsuite besteht aus Werkzeugen für die Entwicklung und Administration von SOPERA-Diensten, die in die Entwicklungsumgebung Eclipse integriert sind [ECLIPSE]. Sie deckt den gesamten Lebenszyklus eines Dienstes vom Entwurf über die Implementierung und den Betrieb ab. Zudem erlaubt die Toolsuite den Import und Export von WSDL-Dateien. Zur Erzeugung der notwendigen Zugriffslogik für Dienstanutzer und -anbieter enthält die Toolsuite einen Code-Generator.

Die Participant-API kapselt die Anbindung der Dienst-Logik an den SBB und stellt Funktionalitäten für Service Consumer und Provider zur Verfügung. Die PAPI-Bibliothek existiert für Java und Microsoft .NET. Es wird dabei zwischen einer typisierten und einer untypisierten PAPI unterschieden. Die typisierte PAPI erlaubt den vereinfachten Umgang mit JAXB Objekten (Java-API for XML Binding)

²⁵<http://tomcat.apache.org>

[JAXB]. Im Gegensatz dazu bietet die untypisierte PAPI tiefergehende Kontrolle über Anbindung an den SBB und über die serialisierten XML-Dokumente.

Eine Besonderheit der SOPER-Plattform besteht in der Art der Beschreibung der Dienste. Diese entspricht im Wesentlichen einer um SOPER-spezifische Elemente ergänzten Erweiterung der WSDL. Es findet jedoch eine getrennte Speicherung des abstrakten und des konkreten Teils der Beschreibung statt. Dies geschieht vor dem Hintergrund, dass ein Dienst von mehreren Service-Providern angeboten werden kann. Die abstrakte Beschreibung wird in Service-Description-Dateien (SDX), die konkreten Details werden in Service-Provider-Description-Dateien (SPDX) gespeichert.

Für die Beispielanwendung wurde SOPER aus mehreren Gründen gewählt. Zum einen handelt es sich dabei um eine in der Praxis erprobte Enterprise Plattform, die neben der Deutschen Post AG und DHL bei größeren Unternehmen und Behörden im Einsatz ist. Zum anderen ermöglicht die Toolsuite eine rasche Entwicklung einer Webservice-basierten SOA, die über alle wesentlichen Eigenschaften verfügt. Ein besonderer Punkt ist die Möglichkeit, WSDL zu importieren und daraus Service-Rümpfe generieren zu können. Dieses führt den im Buch gezeigten modellbasierten Ansatz konsequent weiter und zeigt, wie weit es heute schon mit geeigneten Plattformen möglich ist, vom Modell zu einer lauffähigen SOA zu kommen.

Darüberhinaus ist SOPER als Open-Source frei verfügbar, wodurch der Leser das Entwickeln der Beispielanwendung selbst nachvollziehen kann, ohne sich in Unkosten zu stürzen. SOPER ist für Open-Source außergewöhnlich gut dokumentiert. Auch dies erleichtert den Einstieg in eine SOA Plattform. Zuletzt sei noch gesagt, dass die Autoren in einem Vorläufer-Projekt bereits gute Erfahrungen mit SOPER als SOA Plattform gemacht haben.

Apache ODE und JBoss Riftsaw

Bei Apache ODE (Orchestration Director Engine) handelt es sich um die Implementation einer BPEL-Engine der Apache Foundation [ODE]. Ursprünglich wurde diese unter dem Namen PXE durch die Firma FiveSight PXE entwickelt und 2005 als Open-Source veröffentlicht. Noch im selben Jahr wurde FiveSight von der Firma Intalio übernommen. Die BPEL-Engine PXE bildete mit Codezugaben von Sybase die Basis eines Projects, das 2006 unter dem Namen „Orchestration Director Engine“ bei der Apache Foundation beantragt wurde. Im Jahr 2007 erfolgte der offizielle Projektstart und kurz darauf die erste Veröffentlichung von Apache ODE. Heute findet ein großer Teil der Entwicklung durch Intalio und Entwickler von Apache ServiceMix statt.

Hauptsächliche Verwendung findet Apache ODE in BPEL Servern, in denen sie zur Orchestrierung von Diensten eingesetzt wird. Es lassen sich mit BPEL sogenannte „Composite Services“, also aus verschiedenen Diensten zusammengesetzte Dienste bilden, die beispielsweise zur Prozessautomatisierung und zum Business Process Management (BPM) genutzt werden können.

Apache ODE zeichnet sich durch eine performante Implementation aus und setzt die Standards WS-BPEL 1.1 und 2.0 bis auf wenige Ausnahmen um. Die Engine unterstützt die Kommunikation mit Webservices auf Basis von SOAP und REST und eignet sich daher für ein breites Spektrum der Entwicklung einer auf Webservices aufgebauten SOA.

Die Engine an sich kann aber auch als Komponente in SOA Plattformen oder Produkte integriert werden. Tatsächlich ist Apache ODE bereits Teil einiger Open-Source und kommerzieller Lösungen, wie z. B. Intalio BPM und JBoss Riftsaw [JBRIFTSAW]. Letzteres ist eine speziell auf den JBoss Application Server angepasste Version von Apache ODE, die sich aber auch unabhängig von der JBoss SOA-Plattform verwenden lässt. Im Einzelnen erweitert Riftsaw die BPEL-Engine um Fähigkeiten um mit dem JBoss ESB zusammenzuspielen und über den JBoss Webservice-Stack zu kommunizieren. Zudem beherrscht Riftsaw ein UDDI-Lookup und wird mit einer eigenen Management-Konsole ausgeliefert. Das alles wird als Gesamtpaket gebündelt ausgeliefert, das sich im JBoss AS installieren lässt.

Sowohl Apache ODE als auch das darauf aufbauende JBoss Riftsaw sind als Open-Source frei verfügbar, wodurch man sich, z. B. durch Nachvollziehen des Beispiels im Buch, selbst ein Bild davon machen kann. Apache ODE gilt als die derzeit beste Open-Source BPEL-Engine. Die Installation gestaltet sich durch das auf Apache-Ant basierende Skript sehr einfach und Riftsaw ist mit wenigen Schritten in unter 10 Minuten einsatzbereit. Für die Entwicklung, Verfeinerung und das Deployment von BPEL-Abläufen existieren bereits gute Tools für die Eclipse IDE, wie beispielsweise die JBoss-Tools [JBTOOLS]. Zudem lassen sich BPEL-konforme Dateien aller Hersteller damit ohne Modifikation verwenden, sofern diese den Standards entsprechen und tatsächlich ablauffähigen Code darstellen.

Literatur

- [ALL09] Allweyer T (2009) BPMN 2.0 – Business Process Model and Notation. Books on Demand GmbH, Norderstedt
- [DD2] Evans EJ (2003) Domain driven design: Tackling Complexity in the Heart of Software. Addison-Wesley Longmann, Boston, MA
- [EMF2] Steinberg D, Budinsky F, Paternostro M, Merks E (2008) EMF: Eclipse Modeling Framework, 2nd edn. Addison-Wesley Longman, Boston, MA
- [HEU07] Heutschi R (2007) Serviceorientierte Architektur – Architekturprinzipien und Umsetzung in die Praxis. Springer, Berlin
- [KRA07] Krafzig D, Banke K, Slama D (2007) Enterprise SOA – Wege und Best Practices für Serviceorientierte Architekturen. Mitp-Verlag, Heidelberg
- [SOAWS] Melzer I, et al (2010) „Service-orientierte Architekturen mit Web Services – Konzepte, Standards, Praxis“. Elsevier Spektrum Akademischer Verlag, Heidelberg
- [STA07] Stahl T, Völter M, Efftinge S, Haase S (2007) Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management. Dpunkt Verlag, Heidelberg
- [ZEP06] Zeppenfeld K, Wolters R (2006) Generative Software-Entwicklung mit der MDA. Elsevier Spektrum Akademischer Verlag, Heidelberg

Links

- [ANDRO] AndroMDA
<http://www.andromda.org> (29.03.2011)
- [BPEL] BPEL 2.0
<http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (29.03.2011)
- [BPMN] Business Process Model and Notation
<http://www.omg.org/spec/BPMN/> (04.04.2011)
- [CHECK] oAW Check
http://www.openarchitectureware.org/pub/documentation/4.1/r30_checkReference.pdf (29.03.2011)
- [DD1] Domain Driven Design
<http://www.domaindrivendesign.org/> (19.02.2011)
- [ECLIPSE] Eclipse
<http://www.eclipse.org/> (29.03.2011)
- [EMF] Eclipse Modeling Framework
<http://www.eclipse.org/modeling/emf/> (26.02.2011)
- [EMP] Eclipse Modeling Project
<http://www.eclipse.org/modeling/> (29.03.2011)
- [FDD] Feature Driven Development
<http://www.nebulon.com/articles/fdd/latestfdd.html> (19.02.2011)
- [JAXB] Java API for XML Binding (JAXB)
<http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index.html>
(29.03.2011)
- [JBRIFTSAW] JBoss Riftsaw
<http://www.jboss.org/riftsaw> (29.03.2011)
- [JBTOOLS] JBoss Tools
<http://jboss.org/tools> (29.03.2011)
- [MDA] Model Driven Architecture
<http://www.omg.org/mda/> (02.02.2011)
- [MID1] Notationsübersicht BPMN 2.0
<http://www.mid.de/fileadmin/mid/PDF/BPMN-Poster.pdf> (04.04.2011)
- [MOF] Meta Object Facilities
<http://www.omg.org/mof/> (13.02.2011)
- [OCL] Object Constraint Language
<http://www.omg.org/spec/OCL/2.2/> (13.02.2011)
- [ODE] Apache ODE
<http://ode.apache.org/> (29.03.2011)
- [OMG] Object Management Group
<http://www.omg.org/> (17.07.2011)
- [QVT] Query View Transformation
<http://www.omg.org/cgi-bin/doc?ad/2002-4-10> (02.02.2011)
- [SOAA] Modeling with SoaML
<http://www.ibm.com/developerworks/rational/library/09/modelingwithsoaml-1/index.html> (16.01.2011)
- [SOAML] SoaML Version 1.0 – Beta 1
<http://www.omg.org/spec/SoaML/1.0/Beta1/> (29.03.2011)
- [SOAP] SOAP
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/> (29.03.2011)
- [SOPERA] SOPERA
<http://www.sopera.de> (29.03.2011)
- [UML] Unified Modeling Language (UML)
<http://www.uml.org/> (29.03.2011)

- [UMPS] UML Profile and Metamodel for Services (UPMS)
<http://www.omg.org/docs/soa/06-09-09.pdf> (29.03.2011)
- [WSDL] WSDL 1.1
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315> (29.03.2011)
- [XPAND] oAW Xpand
http://www.openarchitectureware.org/pub/documentation/4.1/r20_xPandReference.pdf (29.03.2011)
- [XPATH] Xpath 1.0
<http://www.w3.org/TR/1999/REC-xpath-19991116> (29.03.2011)
- [XTEND] oAW Xtend
http://www.openarchitectureware.org/pub/documentation/4.1/r25_extendReference.pdf (29.03.2011)

Model Driven SOA

Anwendungsorientierte Methodik und Vorgehen in der
Praxis

Rempp, G.; Akermann, M.; Löffler, M.; Lehmann, J.

2011, XV, 432 S. 315 Abb., 300 Abb. in Farbe.,

Hardcover

ISBN: 978-3-642-14469-1