

Highlights

- ▷ Sie erfahren, wie man SOA-Anwendungen modellgetrieben und damit hocheffizient entwickeln kann.
- ▷ Der Beitrag zeigt den Weg von der Geschäftsprozessmodellierung mit BPMN bis zur Transformation des fachlichen Modells in ein plattform-spezifisches Modell mit Code.
- ▷ Sie lernen den Kern der Lösung – ein spezielles UML-Profil für SOA-Anwendungen – kennen.

Keywords

SOA
MDD
UML Profil
BPMN
Web-Services

MDD trifft SOA

Dr. Roman Nagy
microTOOL GmbH, Berlin

Abstrakt

Modellgetriebene Softwareentwicklung (kurz: MDD für Model-Driven Development) stellt eine hocheffiziente Entwicklungstechnik dar. Diese Technik ist für zahlreiche Arten von Software einsetzbar, die nach ganz unterschiedlichen Entwicklungsparadigmen erstellt werden. Dazu gehören unter anderem die eng mit den Geschäftsprozessen von Unternehmen verbundenen **Unternehmensanwendungen** (Enterprise Software), die heute zunehmend unter Verwendung des Architekturstils SOA (Service-orientierte Architektur) entwickelt werden. Der Entwurf dieser Anwendungen findet auf einem sehr hohen, fachlichen Abstraktionsniveau statt und muss in der Folge in technisch anspruchsvolle Softwarelösungen umgesetzt werden. Für diese Transformation ist das Konzept von MDD besonders geeignet.

In diesem Artikel wird eine Lösung zur modellgetriebenen Entwicklung von SOA-Anwendungen vorgestellt. Dabei werden die theoretischen Voraussetzungen für solche Anwendungen mithilfe eines Beispiels erläutert.

Inhalt

1. Einführung in MDD 2
 2. Profil einer SOA-Anwendung 2
 - 2.1 Abbildung eines Web-Services 3
 - 2.2 Beziehungen zwischen Web-Services 4
 - 2.3 Modellieren von Geschäftsprozessen 5
 3. Transformieren des PIM 6
 - 3.1 Eingänge der Transformation 6
 - 3.2 Ergebnisse der Transformation 6
 4. Beispiel einer SOA-Anwendung 6
 - 4.1 Architektur der Anwendung 7
 - 4.2 Externe Web-Services 8
 - 4.3 Modellieren des Geschäftsprozesses 10
 - 4.4 Transformieren des Geschäftsprozesses 11
 5. Fazit 12
 6. Referenzen 12
- Über den Autor 13

1. Einführung in MDD

Bei modellgetriebener Entwicklung ist die Hauptaufgabe einer Modelltransformation, ein Modell am Eingang des Transformierungsprozesses so zu konvertieren, dass das Ergebnis am Ausgang des Prozesses aus der Sicht der zu entwickelnden Anwendung einen höheren Wert als das ursprüngliche Modell hat. Den Mehrwert des Ergebnisses schafft das Know-how der Transformation.

In der Praxis wird bei der Softwareentwicklung als Eingang einer Transformation ein fachliches Modell der zu entwickelnden Anwendung verwendet. Dieses Modell bildet die Funktionalität der Anwendung unabhängig von der gegebenen Zielplattform ab und wird als *Platform Independent Model* (PIM) bezeichnet. Das PIM wird durch die Transformation in ein technisch ausgeprägtes Modell konvertiert, das die spezifischen Konzepte der Zielplattform enthält. Das Zielmodell wird auch *Platform Specific Model* (PSM) genannt.

Nach der Transformation wird aus dem PSM der Quellcode generiert, der nach dem Kompilieren und Ausführen die im PIM fachlich modellierte Funktionalität realisiert. Je nach Situation kann das PIM auch direkt in Quellcode transformiert werden. In diesem Fall wird kein PSM erzeugt. Bild 1 zeigt diese beiden Varianten.

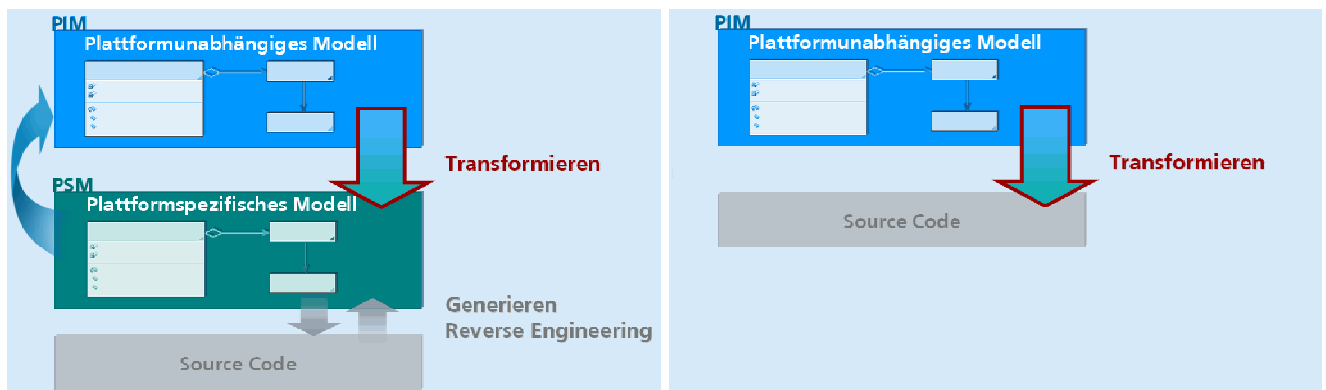


Bild 1: Links die Transformationsschritte mit PSM, rechts der direkte Weg vom PIM zum Code

2. Profil einer SOA-Anwendung

Jede Anwendung, die durch MDD entwickelt werden soll, muss in der Form eines Modells abgebildet werden. Zu diesem Zweck wird eine formale Modellierungssprache festgelegt. Diese Modellierungssprache deckt alle Elemente der fachlichen Domäne ab, zu der die zu entwickelnde Anwendung gehört. Für jedes Element und seine Eigenschaften ist eine spezifische Form der Abbildung definiert. Dies betrifft alle Elemente, die zum Transformieren des PIM in das PSM notwendig sind.

Eine Modellierungssprache kann entweder neu definiert werden. Oder sie kann bestehende Konzepte verwenden bzw. erweitern. Als Grundlage dafür sind alle existierenden Modellierungssprachen geeignet, die eine streng definierte formale Notation besitzen. In der Praxis wird häufig die UML [1] verwendet.

Die UML ist eine formale Sprache ohne Bezug auf eine spezifische Domäne. Durch ihre Erweiterungsmöglichkeiten ist sie jedoch an eine konkrete Domäne anpassbar und kann dadurch Elemente dieser Domäne genauer und präziser abbilden. Zur Erweiterung des Sprachumfangs der UML können Stereotypen und benutzerdefinierte Eigenschaften verwendet werden. Eine Menge von Stereotypen und benutzerdefinierten Eigenschaften, die zum Zweck der Anpassung der UML an eine konkrete fachliche Domäne definiert wurde, wird auch als *UML-Profil* bezeichnet. In der hier vorgestellten Lösung wird die UML zusammen mit einem definierten UML-Profil als Modellierungssprache für das PIM angewendet.

Die Qualität des verwendeten UML-Profiles ist häufig das Hauptkriterium für einen Erfolg oder einen Misserfolg des ganzen Einsatzes von MDD in konkreten Softwareprojekten. Beim Definieren eines UML-Profiles müssen alle relevanten Elemente der modellierten Anwendung betrachtet werden. Die wichtigsten Elemente einer SOA-Anwendung kann man in zwei Gruppen aufteilen: In der ersten Gruppe befinden sich Elemente, die das dynamische Verhalten der SOA-Anwendung realisieren. Dazu gehört der Geschäftsprozess mit allen seinen Elementen. Die zweite Gruppe besteht aus Elementen, die die statische Struktur der SOA-Anwendung repräsentieren. Dazu gehören alle Elemente der Geschäftspartner, die in dieser Anwendung vorkommen und die durch den eigentlichen Geschäftsprozess orchestriert werden. In der Praxis sind solche Elemente der Geschäftspartner durch *Web-Services* implementiert.

In nächsten Teilen dieses Beitrags wird das UML-Profil näher beschrieben, das für das Modellieren aller wesentlichen Elemente einer SOA-Anwendung entworfen wurde.

2.1 Abbildung eines Web-Services

Ein Web-Service bietet eine bestimmte Funktionalität an, ohne zu wissen, wo und in welchem Kontext diese in Anspruch genommen wird. Eine SOA-Anwendung kann beim Realisieren der eigenen Geschäftslogik die angebotene Funktionalität von einem oder mehreren Web-Services benutzen. Dadurch wird aus einzelnen funktionalen Teilen eine hochwertige Gesamtanwendung implementiert. Man sagt, dass die Web-Services durch die SOA-Anwendung *orchestriert* werden.

Um einen Web-Service im PIM abzubilden, wird in der hier präsentierten Lösung die UML-Metaklasse *Package* verwendet. Ein Web-Service, der die Rolle eines externen Anbieters einnimmt, ohne eigene Geschäftslogik zu veröffentlichen, wird durch den Stereotyp `<<PartnerService>>` spezifiziert. Wenn es sich um einen Web-Service handelt, der die zu entwickelnde Geschäftslogik realisieren soll, erhält er den Stereotyp `<<BusinessService>>`.

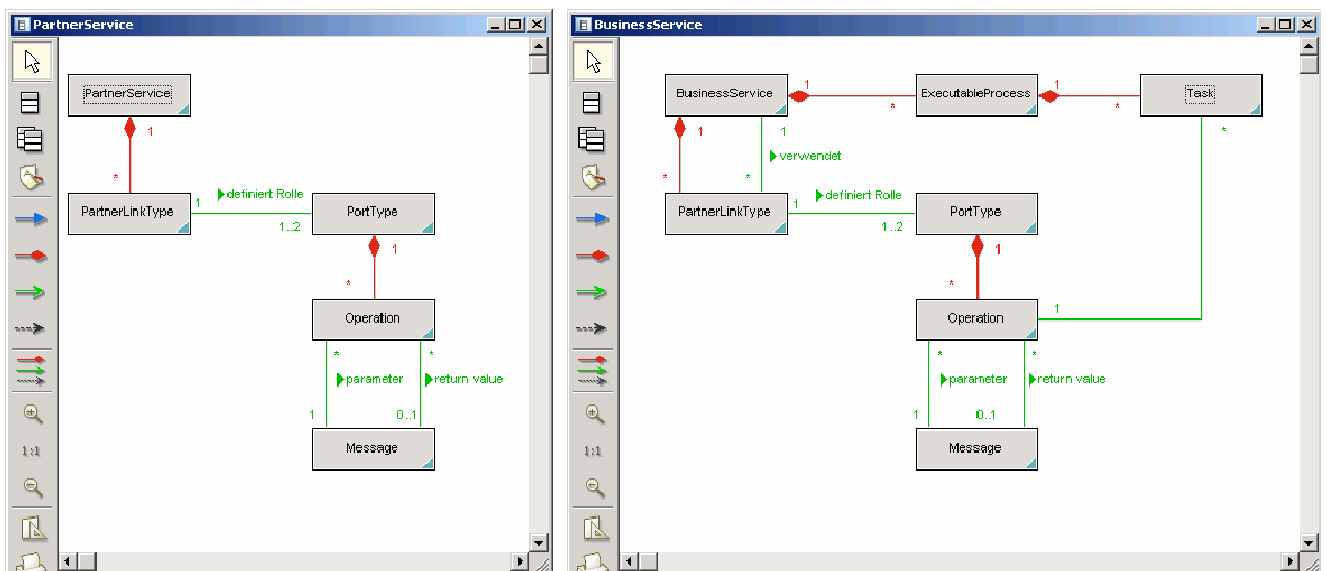


Bild 2: Die wichtigsten Stereotypen des UML-Profiles eines Web-Services: links für *PartnerService*, rechts für *BusinessService*

Um den Inhalt eines *PartnerService* zu modellieren, wurden folgende Stereotypen definiert:

- <<Message>> basiert auf der UML-Metaklasse *Klasse*
- <<PartnerLinkType>> basiert auf der UML-Metaklasse *Klasse*
- <<PortType>> basiert auf der UML-Metaklasse *Klasse*
- <<Operation>> basiert auf der UML-Metaklasse *Methode*

Elemente mit diesen Stereotypen können die gesamte Schnittstelle und die Verwendungsmöglichkeiten eines *PartnerService* abbilden.

Bei einem *BusinessService* handelt es sich auch um einen Web-Service, der genauso wie ein *PartnerService* vom außen verwendet werden kann. Deshalb werden die vier oben genannten Stereotypen auch für das Modellieren eines *BusinessService* verwendet. Für einen *BusinessService* wurden darüber hinaus weitere Stereotypen definiert, die das dynamische Verhalten seiner Geschäftslogik definieren:

- <<ExecutableProcess>> basiert auf der UML-Metaklasse *Aktivität*
- <<Task>> basiert auf der UML-Metaklasse *Aktion*

Für alle Stereotypen des hier präsentierten UML-Profiles wurde eine eigene Semantik im Rahmen des PIM festgelegt:

Durch die Klasse mit dem Stereotyp <<Message>> wird der Inhalt einer Nachricht zwischen zwei Web-Services modelliert. Sie enthält Attribute, durch die einzelne Teile der Nachricht abgebildet werden.

Eine Klasse mit dem Stereotyp <<PortType>> definiert – wie der Name sagt – einen so genannten Port-Typ. Ein Port-Typ beschreibt die Kommunikationsmöglichkeiten eines Web-Service. Die Klasse <<PortType>> enthält Methoden mit dem Stereotyp <<Operation>>. Jede dieser Methoden kann einen Parameter und Rückgabewert definieren. Beide müssen vom Typ <<Message>> sein. Dieses Konstrukt definiert eine Operation des modellieren Port-Typs zusammen mit den Nachrichten, die als Eingang und Ausgang der Operation verwendet werden können. Durch einen Aufruf dieser Operation wird der modellierte Web-Service angesprochen. Die Operation selber dient dem Nachrichtenaustausch.

Eine Operation muss genau einen Parameter besitzen. Falls die Operation nur Parameter ohne Rückgabewert definiert, wird sie zum asynchronen Überreichen einer Nachricht an den Web-Service oder für das Abfragen einer Nachricht vom Web-Service verwendet. Eine Operation mit Parameter und Rückgabewert wird zur synchronen Kommunikation mit dem Web-Service benutzt. Dabei wird eine Nachricht an den Web-Service verschickt. Danach wird so lange gewartet, bis eine Antwort vom Web-Service empfangen werden kann.

Die Klasse mit dem Stereotyp <<PartnerLinkType>> ordnet einzelnen Port-Typen des Web-Services Rollen zu, mit denen man Operationen eines konkreten Port-Typs aufrufen kann. Jede Klasse <<PartnerLinkType>> darf maximal zwei Rollen definieren und jede Rolle ist genau für einen Port-Type zuständig.

Eine Klasse <<PartnerLinkType>> kann zu der Klasse <<PortType>> entweder durch eine *Komposition* oder durch eine *Assoziation* in Beziehung gesetzt werden. Beide Elemente sind Metaklassen der UML. Eine *Komposition* bedeutet, dass der zugeordnete Port-Typ mit seinem Service intern auf dem modellierten Web-Service implementiert ist. Durch seine Operationen können Nachrichten an diesen Web-Service gesendet werden. Eine *Assoziation* dagegen bedeutet, dass der Service des Port-Typs auf dem Client laufen muss, und dass Operationen des Port-Typs zum Abholen der Antwort auf eine vorherige Anfrage verwendet werden.

Falls die Klasse <<PartnerLinkType>> nur eine Rolle für einen Port-Typ definiert, wird zwischen den beiden Klassen eine *Komposition* angelegt und die Operationen des Port-Typs werden synchron aufgerufen. Falls die Klasse <<PartnerLinkType>> zwei Rollen für je einen Port-Typ definiert, handelt es sich um Port-Typen für eine asynchrone Kommunikation. Durch den mit einer *Komposition* assoziierten Port-Typ wird eine Nachricht an den Web-Service gesendet. Durch den mit einer *Assoziation* assoziierten Port-Typ wird eine Antwort von dem Web-Service empfangen.

2.2 Beziehungen zwischen Web-Services

Im Abschnitt 2.1 wurde schon angedeutet, dass ein Web-Service in der hier vorgestellten Lösung durch ein *Package* abgebildet wird. Um alle Beziehungen eines Web-Service modellieren zu können, wird er in einem Packagediagramm dargestellt – zusammen mit allen Web-Services (bzw. Packages), mit denen dieser Web-Service im Rahmen der SOA-Anwendung kommunizieren soll.

Wenn ein Client-Web-Service <<*BusinessService*>> Dienste eines Partner-Web-Services <<*PartnerService*>> in Anspruch nehmen will, muss er dessen Partner-Link-Typen kennen. Deshalb werden alle Klassen mit dem Stereotyp <<*PartnerLinkType*>> aus dem *PartnerService* veröffentlicht. Diese veröffentlichten Klassen werden dann vom *BusinessService* durch einen Typ-Import mit dem Stereotyp <<*PartnerLink*>> importiert. So wird zum einen deutlich gemacht, dass der *BusinessService* Dienste des *PartnerService* in Anspruch nehmen wird. Zum anderen wird dadurch ausgedrückt, welche konkreten Port-Typen des *PartnerService* die Kommunikation zwischen den beiden Web-Services realisieren.

Ein Web-Service kann sowohl als Client als auch als Partner vorkommen. Er kann Dienste von anderen Web-Services in Anspruch nehmen und gleichzeitig eigene Dienste zur Verfügung stellen.

2.3 Modellieren von Geschäftsprozessen

Ein Geschäftsprozess besteht aus Aufgaben, die in einer bestimmten Reihenfolge durchgeführt werden müssen. Es gibt mehrere Notationen, mit denen man Geschäftsprozesse modellieren kann. Jede Notation hat eine eigene Menge von Elementen, die sowohl die Aufgaben des Geschäftsprozesses als auch zusätzliche Informationen abbilden können. In der hier präsentierten Lösung wird für das Modellieren von Geschäftsprozessen die *Business Process Modeling Notation* (BPMN) [2] verwendet.

Die BPMN wurde im Jahr 2002 erarbeitet. Im Jahr 2005 wurde sie durch die Object Management Group (OMG) als Standard übernommen. Sie definiert Notation und Semantik einzelner Symbole, durch die ein Geschäftsprozess modelliert und fachlich beschrieben werden kann.

Einer der Vorteile in der BPMN ist, dass alle Aufgaben des Geschäftsprozesses auf so genannte *Pools* verteilt werden. Ein Pool stellt somit ein Container für Aufgaben dar. Jeder Pool ist einer prozessbeteiligten Einheit – typischerweise einer Organisation – zugeordnet, in der die modellierten Aufgaben realisiert werden. Ein Pool bestimmt den Verantwortungsbereich der prozessbeteiligten Einheit. In der präsentierten Lösung ist für jeden Pool ein Web-Service verantwortlich. Auf diese Weise kann modelliert werden, welche Aufgaben durch welche prozessbeteiligte Einheit des verantwortlichen Web-Services realisiert werden.

Ein weiterer Vorteil der BPMN ist, dass sie gleichzeitig einen Weg definiert, wie die modellierte Notation mit ihrer Semantik in Quellcode transformiert werden kann. Als Zielplattform ist hier die Sprache BPEL (*Business Process Execution Language*) [3] vorgesehen, die speziell für das Programmieren der Abläufe von Geschäftsprozessen entworfen wurde. Da BPEL eine standardisierte Programmiersprache ist, wird ein Modell mit einer standardisierten Notation (BPMN) in Quellcode in einer standardisierten Sprache (BPEL) durch eine von BPMN standardisierte Transformation umgewandelt.

Um ein Geschäftsprozessdiagramm an das neu definierte UML-Profil anzubinden, wurde ein neuer Stereotyp <<*ExecutableProcess*>> definiert. Dieser Stereotyp basiert auf der UML-Metaklasse *Aktivität*. Eine Aktivität mit dem Stereotyp <<*ExecutableProcess*>> darf nur in einem *BusinessService* angelegt werden.

Unter jeder Aktivität mit dem Stereotyp <<*ExecutableProcess*>> wird ein Geschäftsprozessdiagramm erstellt, das die Geschäftslogik der zu entwickelnden SOA-Anwendung mit der Hilfe von BPMN abbildet. Dieses Diagramm enthält Aufgaben, die durch *Aktionen* mit dem Stereotyp <<*Task*>> dargestellt werden. Diese Aktionen repräsentieren einzelne Schritte des Geschäftsprozesses. Eine solche Aktion hat eine direkte Beziehung zu einer Operation eines Port-Typs (siehe Bild 2), weil sie den Aufruf dieser Operation darstellt. Von

welchem Web-Service die aufzurufende Operation angeboten wird, ist an der Pool-Zugehörigkeit der Aktion erkennbar.

3. Transformieren des PIM

3.1 Eingänge der Transformation

Die hier präsentierte Lösung setzt voraus, dass das PIM mit der Hilfe des neu definierten UML-Profiles und der BPMN Notation modelliert wird.

Für jede SOA-Anwendung wird mindestens ein *Package* mit dem Stereotyp `<<BusinessService>>` modelliert. Dieses *Package* stellt einen Web-Service dar, in dem die Geschäftslogik gespeichert ist. Das *Package* enthält Klassen, die die Kommunikation mit diesem Web-Service beschreiben (*PartnerLinkType*, *PortType*, *Message*). Gleichzeitig enthält es mindestens eine Aktivität mit dem Stereotyp `<<ExecutableProcess>>`. Unter dieser Aktivität liegt ein Geschäftsprozessdiagramm, das in BPMN modelliert wurde. Außerdem werden in dem *BusinessService* auch Typ-Importe zu *PartnerService*-Packages gespeichert, falls der *BusinessService* andere Web-Services für das Implementieren der Geschäftslogik benötigt.

Zusätzlich werden Packages mit dem Stereotyp `<< PartnerService >>` erzeugt. Diese Packages stellen Web-Services dar, die externe Dienste für den *BusinessService* anbieten. Solche Packages enthalten keine Aktivitäten mit dem Geschäftsprozessdiagramm und sie importieren auch keine weiteren Web-Services. Ein *PartnerService* enthält nur Klassen (*PartnerLinkType*, *PortType*, *Message*), die die Kommunikationsmöglichkeiten mit diesem Web-Service definieren.

3.2 Ergebnisse der Transformation

Aus den einzelnen Elementen des PIM werden Dateien transformiert, die die zu entwickelnde SOA-Anwendung zusammenstellen. Dabei handelt es sich um die folgenden Dateien:

- *WSDL-Datei*. Diese Datei ist in der *Web Service Definition Language* (WSDL) [4] definiert. Sie veröffentlicht alle Schnittstellen des Web-Services. Die Datei wird aus den Klassen mit den Stereotypen `<<Message>>`, `<<PortType>>`, `<<PartnerLinkType>>` des jeweiligen Web-Services transformiert.
- *BPEL-Datei*. In dieser Datei ist der gesamte Ablauf des Geschäftsprozesses durch die Sprache BPEL implementiert. Die BPEL-Datei wird aus dem Geschäftsprozessdiagramm transformiert.
- *PDD-Datei*. Der so genannte *Process Deployment Descriptor* beschreibt auf sehr technischem Niveau, wo und wie die einzelnen Komponenten der SOA-Anwendung verteilt werden müssen.

4. Beispiel einer SOA-Anwendung

In diesem Abschnitt wird eine SOA-Anwendung präsentiert, die mit MDD entwickelt wurde. Bei der Anwendung handelt es sich um ein virtuelles Reisebüro. Das Reisebüro bietet seinen Kunden an, über das Internet einen Flug und ein Fahrzeug vor Ort zu reservieren.

Um diese Dienste zu ermöglichen, kooperiert das Reisebüro *microTRAVEL* mit zwei externen Anbietern: Der erste Partner ist die Fluggesellschaft *microJET*, der zweite der Autovermieter *microDRIVE*. Beide Partner stellen Web-Services zur Verfügung, durch die das Reisebüro alle Dienste über das Internet in Anspruch nehmen kann.

Zu diesem Zweck wird auf der Seite des Reisebüros *microTRAVEL* eine SOA-Anwendung entwickelt. Im Rahmen dieser Anwendung wird die Geschäftslogik implementiert, die die Zusammenarbeit des Reisebüros mit seinen Partnern *microJET* und *microDRIVE* realisiert.

4.1 Architektur der Anwendung

An der SOA-Anwendung für das Reisebüro *microTRAVEL* sind folgende Komponenten beteiligt:

- *microJET* <<PartnerService>>: externer Web-Service. Er stellt Dienste für das Reservieren eines Fluges zur Verfügung.
- *microDRIVE* <<PartnerService>>: externer Web-Service. Er stellt Dienste für die Buchung eines Fahrzeuges zur Verfügung.
- *microTRAVEL* <<BusinessService>>: zu entwickelnder Web-Service. Er enthält die eigentliche Geschäftslogik der SOA-Anwendung.
- *Client* <<ServiceClient>>: externer Web-Client. Er ist nicht ein Bestandteil der Anwendung.

Wie im Bild 3 dargestellt, werden alle diese Komponenten durch Packages mit entsprechenden Stereotypen modelliert. Der Web-Service *microTRAVEL* ist sowohl als Anbieter als auch als Client von Diensten vorgesehen. Aus der Sicht des Package *Client* bietet *microTRAVEL* den Dienst für das Reservieren einer Reise an. Dazu wird der Partner-Link-Typ *ReiseBuchungService* aus dem Package *microTRAVEL* veröffentlicht und durch den Typ-Import <<PartnerLink>> in das Package *Client* importiert.

Um diesen Dienst durchführen zu können, benutzt der Web-Service *microTRAVEL* Dienste von den externen Anbietern *microJET* und *microDRIVE*. Diese Anbieter sind ebenfalls als Packages modelliert. Aus jedem dieser Packages wird der entsprechende Partner-Link-Typ veröffentlicht und durch den Typ-Import <<PartnerLink>> in das Package *microTRAVEL* importiert.

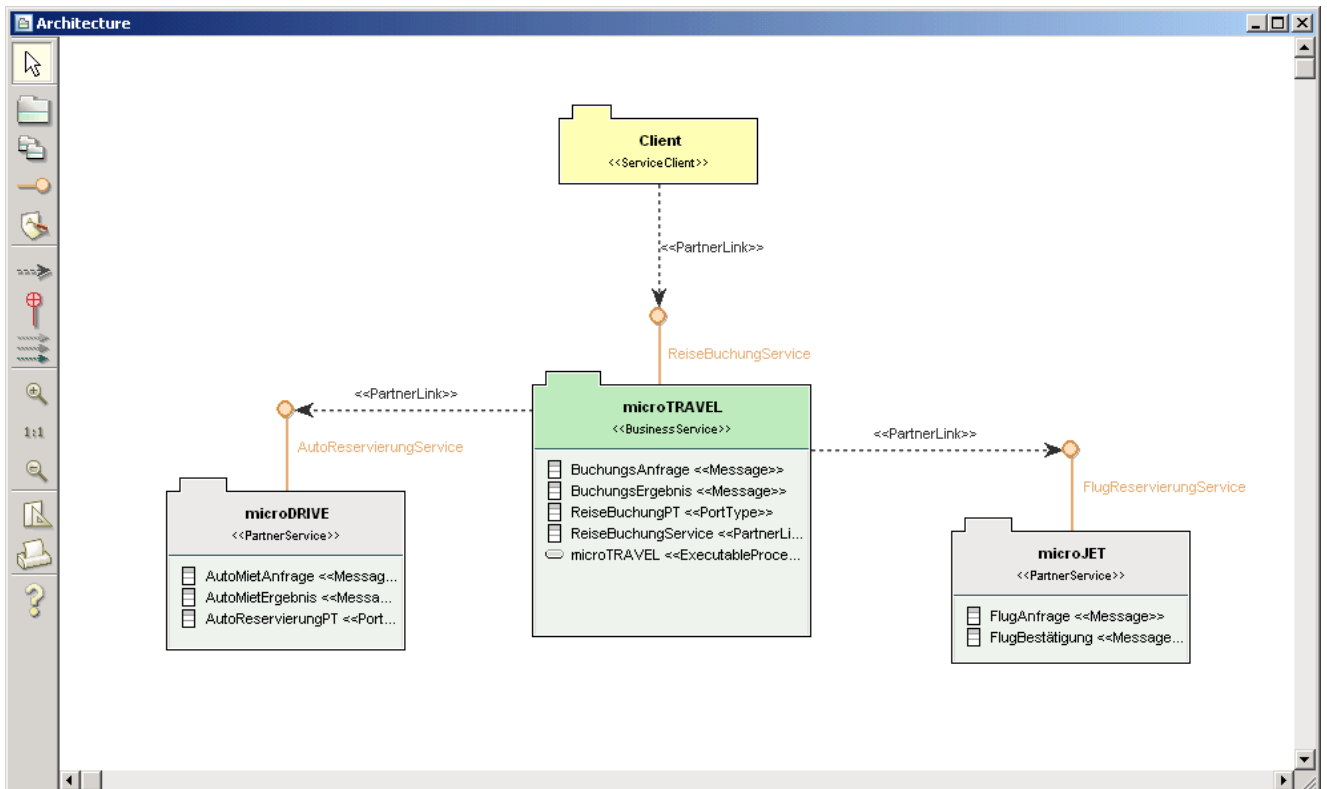


Bild 3: Architektur der Anwendung für das Reisebüro *microTRAVEL*

4.2 Externe Web-Services

Für jeden Web-Service mit dem Stereotyp `<<PartnerService>>` müssen seine Kommunikationsmöglichkeiten definiert werden. Dies beinhaltet die Klassen, die die Schnittstelle des Web-Service darstellen. Bild 4 zeigt diese Klassen für den *PartnerService microDRIVE*.

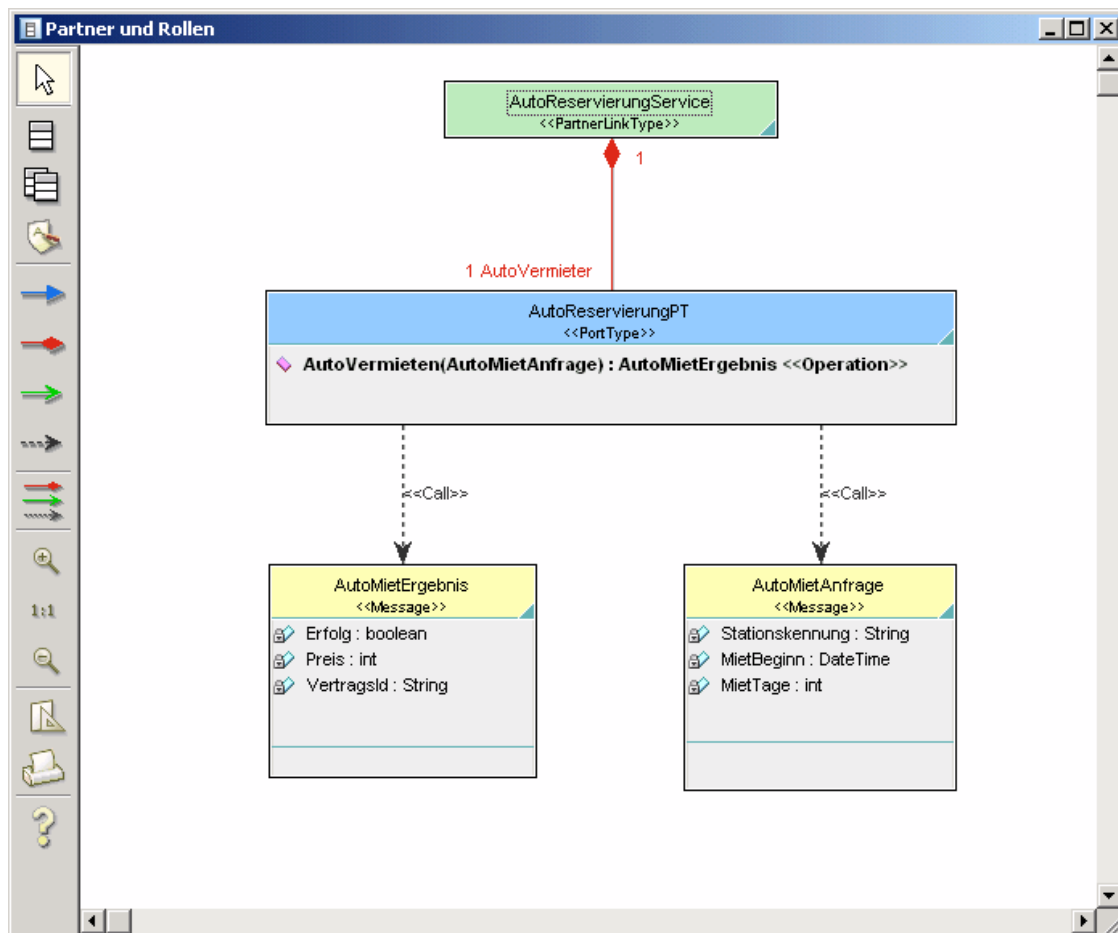
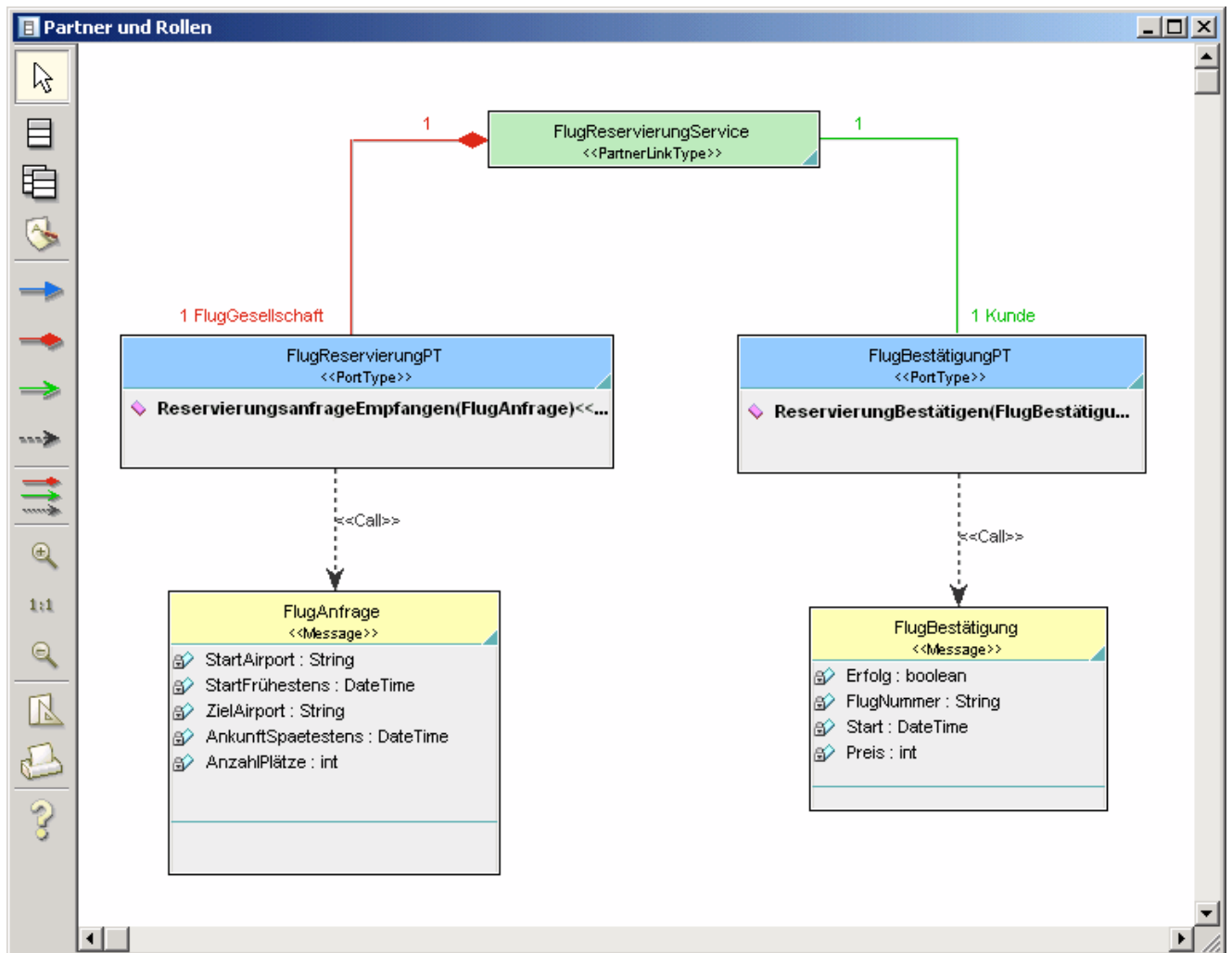


Bild 4: Die Schnittstelle des Web-Service *microDrive*

Mit *microDRIVE* kann man durch den Partner-Link-Typ *AutoReservierungService* kommunizieren. Dieser Partner-Link-Typ veröffentlicht den Port-Typ *AutoReservierungPT* und definiert die Rolle *AutoVermieter*, unter der *microDRIVE* während der Kommunikation angesprochen werden muss. Im Port-Typ *AutoReservierungPT* ist die Operation *AutoVermieten* modelliert, die man während der Kommunikation mit dem Web-Service aufrufen kann. Diese Operation hat als Eingangsparameter eine Nachricht des Typs *AutoMietAnfrage* und als Rückgabewert eine Nachricht des Typs *AutoMietErgebnis*.

Da die Operation *AutoVermieten* sowohl Parameter als auch Rückgabewert definiert, wird sie durch einen synchronen Aufruf verwendet.

Im Bild 5 sind die Klassen dargestellt, die den Inhalt des *PartnerServices microJET* definieren.

Bild 5: Die Schnittstelle des Web-Service *microJET*

Mit *microJET* kann man analog zum *microDRIVE* durch den Partner-Link-Typ *FlugReservierungService* kommunizieren. Dieser Partner-Link-Typ definiert zwei Port-Typen: *FlugReservierungPT* und *FlugBestätigungPT*. Im Port-Typ *FlugReservierungPT* ist die Operation *ReservierungsanfrageEmpfangen* mit einem Eingangsparameter des Typs *FlugAnfrage* modelliert. Der Port-Typ *FlugBestätigungPT* enthält die Operation *ReservierungBestätigen* mit einem Eingangsparameter des Typs *FlugBestätigung*.

Damit wird modelliert, dass man mit dem Web-Service *microJET* über den Partner-Link-Typ *FlugReservierungService* asynchron kommunizieren muss. In erstem Schritt wird durch die Operation *ReservierungsanfrageEmpfangen* eine Reservierungsanfrage an den Web-Service gesendet. Dass es sich um ein Einreichen einer Nachricht handelt, ist daran erkennbar, dass der Port-Typ *FlugReservierungPT* durch eine Komposition mit dem *FlugReservierungService* assoziiert ist. Dabei muss der Web-Service in der Rolle *FlugGesellschaft* angesprochen werden.

Durch den Aufruf der Operation *ReservierungsanfrageEmpfangen* wird das Reservierungsverfahren beim *microJET* angestoßen. Die Bestätigung wird dann später durch den Aufruf der Operation *ReservierungBestätigen* abgeholt. Dass es sich um das Empfangen einer Antwort handelt, ist daran erkennbar,

dass der Port-Typ *FlugBestätigungPT* durch eine Assoziation mit dem *FlugReservierungService* assoziiert ist. Dabei muss der Aufrufer die Rolle *Kunde* einnehmen. So wird das Reservierungsverfahren beendet.

4.3 Modellieren des Geschäftsprozesses

Der eigentliche Geschäftsprozess wurde unter Verwendung vom BPMN modelliert. Dazu wurde im Package *microTRAVEL* eine Aktivität mit dem Stereotyp <<ExecutableProcess>> angelegt. Diese Aktivität wurde anschließend durch ein Geschäftsprozessdiagramm verfeinert.

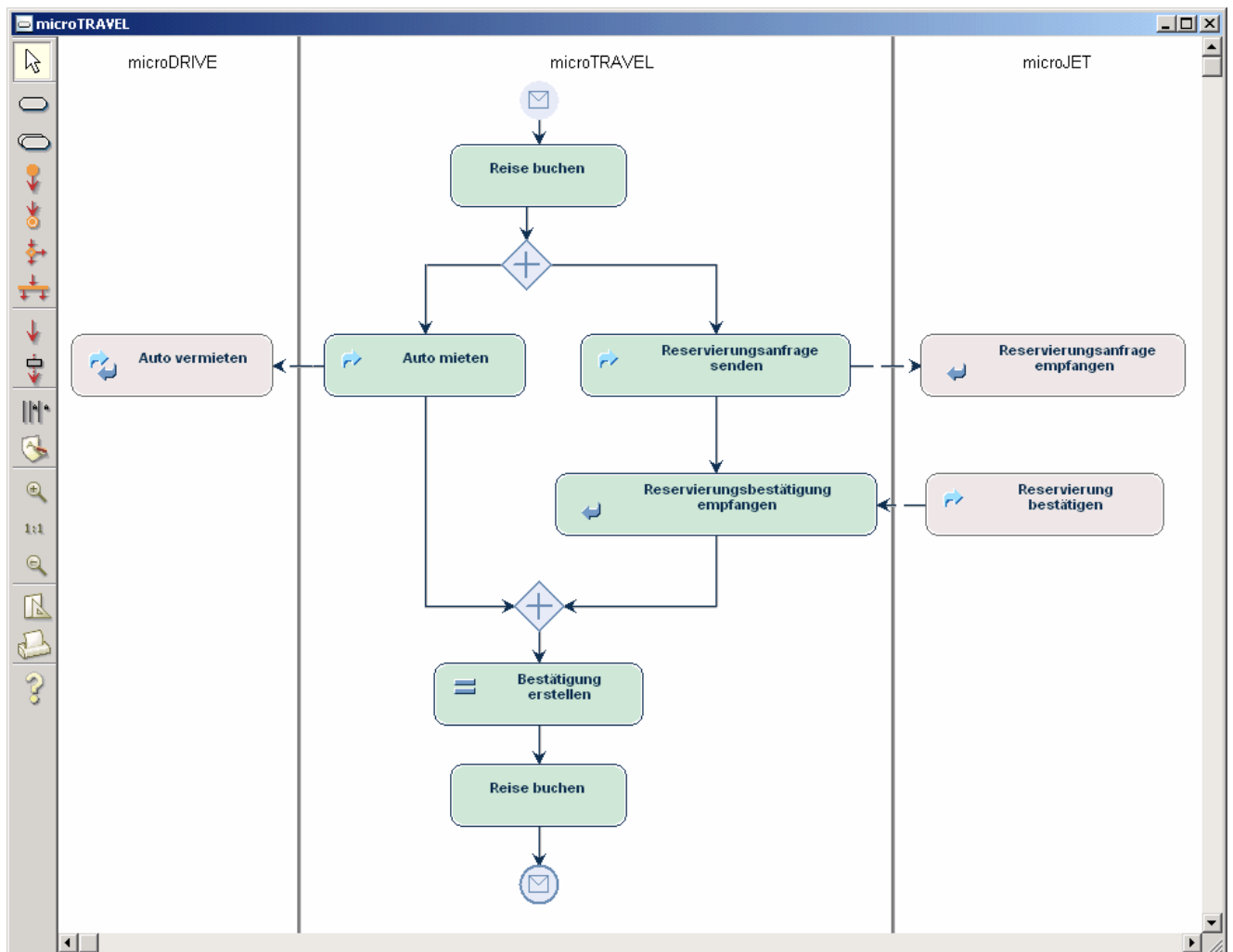


Bild 6: Das Geschäftsprozessdiagramm der SOA-Anwendung für das Reisebüro *microTRAVEL*

Im Bild 6 ist das Geschäftsprozessdiagramm der SOA-Anwendung für das Reisebüro *microTRAVEL* abgebildet. An der Ausführung des Prozesses sind drei Web-Services beteiligt: *microDRIVE*, *microTRAVEL*, *microJET*. Für jeden dieser Web-Services wurde ein Pool modelliert, in dem die Aufgaben (*Tasks*) des jeweiligen Web-Service dargestellt sind.

Im Pool *microDRIVE* befindet sich der *Task* „Auto vermieten“, der den Aufruf der Operation *AutoVermieten* des Port-Typs *AutoReservierungPT* im Web-Service *microDRIVE* repräsentiert. Der detaillierte Verlauf dieser Vermietung bei *microDRIVE* ist dabei aus der Sicht der zu entwickelnden Anwendung uninteressant.

Im Pool *microJET* sind zwei *Tasks* definiert. Der erste ist fürs Empfangen der Flugreservierungsanfrage zuständig, durch den zweiten wird asynchron die Reservierungsbestätigung empfangen. Der *Task* „Reservierungsanfrage empfangen“ repräsentiert der Aufruf der Operation *ReservierungsanfrageEmpfangen* des Port-Typs *FlugReservierungPT* im Web-Service *microJET*. Der *Task* „Reservierung bestätigen“ repräsentiert den Aufruf der Operation *ReservierungBestatigen* des Port-Typs *FlugBestatigungPT* desselben Web-Services. Der konkrete Verlauf der Reservierung ist auch im diesem Fall uninteressant.

Im mittleren Pool *microTRAVEL* sind die eigentlichen Aufgaben des zu entwickelnden Geschäftsprozesses modelliert. Hier wird die Orchestrierung der einzelnen Komponenten der Anwendung dargestellt. Dabei werden sowohl interne *Tasks* als auch Aufrufe der externen *Tasks* verwendet. Alle *Tasks*, die im Verantwortungsbereich des *BusinessServices* *microTRAVEL* liegen, sind grün dargestellt. Dagegen sind die *Tasks* von *PartnerServices* *microDRIVE* und *microJET* grau markiert.

Aus dem Bild 6 ist erkennbar, dass der Geschäftsprozess durch den Empfang einer Nachricht im ersten *Task* „Reise buchen“ gestartet wird. Anschließend werden parallel ein Flug und ein Fahrzeug bei den *PartnerServices* reserviert. Nach dieser Reservierung wird die Reisebestätigung erstellt und durch den letzten *Task* „Reise buchen“ zurück an den Aufrufer gesendet. Dann wird der Geschäftsprozess beendet.

4.4 Transformieren des Geschäftsprozesses

Bei der Transformation wird der Geschäftsprozess in BPEL-Quellcode umgesetzt. Gleichzeitig werden WSDL-Dateien sowohl für die *PartnerServices* als auch für den *BusinessService* generiert. Die WSDL-Dateien werden aus den Klassen im Abschnitt 4.2 transformiert. Aus den Klassen und ihren Beziehungen entstehen XML-Elemente. Hier ein Beispiel für das XML-Element *PartnerLinkType* des Web-Service *microJET*:

```
<plnk:partnerLinkType name="FlugReservierungService">
  <plnk:role name="FlugGesellschaft">
    <plnk:portType name="mtj:FlugReservierungPT" />
  </plnk:role>
  <plnk:role name="Kunde">
    <plnk:portType name="mtj:FlugBestaetigungPT" />
  </plnk:role>
</plnk:partnerLinkType>
```

Der BPEL-Quellcode, der den Geschäftsprozess auf technischem Niveau beschreibt, wird aus einzelnen *Tasks* des Geschäftsprozessdiagramms transformiert. Ein Beispiel der Verwendung eines Dienstes des *PartnerServices* *microDRIVE* zeigt, wie durch den Aufruf der Operation *AutoVermieten* des Port-Typs *AutoReservierungPT* die Buchung eines Fahrzeuges realisiert wurde:

```
<invoke name="Auto_mieten"
  partnerLink="AutoReservierungServicePL"
  portType="mtd:AutoReservierungPT"
  operation="AutoVermieten"
  inputVariable="AutoRequest"
  outputVariable="AutoReply"/>
```

Die einzelnen Tasks des Geschäftsprozesses wurden durch die XML-Elemente `<flow>` und `<sequence>` in die richtige Reihenfolge eingeordnet.

Die letzte Datei, die aus dem PIM transformiert wird, ist der *Process Deployment Descriptor*. In dieser Datei ist die physische Verteilung einzelner Komponenten der SOA-Anwendung definiert.

5. Fazit

Die modellgetriebene Entwicklung ist eine hocheffiziente Technik, die für die Automatisierung der Softwareentwicklung einen großen Beitrag leistet. Je komplizierter und fehleranfälliger die zu entwickelnde Anwendung ist, desto sinnvoller ist der Einsatz von MDD. Zu dieser Art von Software gehören auch die Anwendungen im Bereich der service-orientierten Architektur.

In diesen Beitrag wurde ein Weg vorgestellt, wie man eine SOA-Anwendung mit MDD entwickeln kann. Mit Hilfe eines neu entwickelten UML-Profiles wurde gezeigt, wie einzelne Elemente einer SOA-Anwendung zu modellieren sind. Diese Elemente stellen zusammen mit dem eigentlichen Geschäftsprozess, der in Business Process Modeling Notation abgebildet ist, das komplette fachliche Modell der SOA-Anwendung dar. Anschließend kann man das fachliche Modell in sofort ausführbaren Quellcode in Business Process Execution Language transformieren.

Die praktische Verwendung des UML-Profiles zusammen mit der Modellierung in BPMN wurde anhand eines Beispiels erläutert. Das Beispiel beschreibt eine SOA-Anwendung für ein Reisebüro. Durch übersichtliches und fachliches Modellieren wurde ein Modell der SOA-Anwendung entwickelt, das zum technisch komplizierten Quellcode in XML Format transformiert wurde. Durch dieses Verfahren wurde sowohl die Geschwindigkeit der Entwicklung und der Erweiterung der Anwendung erhöht, als auch die Fehleranfälligkeit des zu entwickelnden Quellcodes reduziert.

6. Referenzen

[1] The Object Management Group: Unified Modeling Language Specification, Version 2.0, <http://www.uml.org>, 2003

[2] The Object Management Group: BPMN (Business Process Modeling Notation) - final adopted specification, Version 1.0, <http://www.bpmn.org>, 2006

[3] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems: BPEL4WS (Business Process Execution Language for Web Services), Version 1.1, <http://www-128.ibm.com/developerworks/library/specification/ws-bpel>, 2002

[4] World Wide Web Consortium: WSDL (Web Services Description Language), Version 1.1, <http://www.w3.org/TR/wsdl.html>, 2001

Kontakt

microTOOL GmbH
Voltastraße 5
D-13355 Berlin

Tel. (+49 30) 467 086-0

Fax (+49 30) 464 47 14

info@microTOOL.de

www.microTOOL.de

Über den Autor

Nach seinem Studium der Informatik an der Slowakischen Technischen Universität in Bratislava war Dr. Nagy von 1997 bis 2000 bei verschiedenen Softwareherstellern in der Slowakei als Entwickler und Chief Analyst tätig. Seit 2001 ist er bei microTOOL beschäftigt und leitet seit 2005 die **objectiF®**-Entwicklung. Im Jahr 2005 promovierte er an der Slowakischen Technischen Universität zum Thema "Automatisches Generieren von Testszenarien aus der UML-Spezifikation".

Copyright

© microTOOL GmbH. Berlin 2006.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Whitepaper berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei betrachtet werden.