Zimmermann, Frank

Working Paper

Modellgetriebene Entwicklung von Schnittstellenprogrammen für die SOA Lösung SAP PI

Arbeitspapiere der Nordakademie, No. 2008-03

Provided in cooperation with:
Nordakademie - Hochschule der Wirtschaft

Arbeitspapiere der Nordakademie

ISSN 1860-0360

Nr. 2008-03

Modellgetriebene Entwicklung von Schnittstellenprogrammen für die SOA Lösung SAP PI

Prof. Dr. Frank Zimmermann
Michael Neuenstadt
Jan Henning Banneitz
Stefan Tanck
Stephan Töpel

April 2008

Eine elektronische Version dieses Arbeitspapiers ist verfügbar unter:
http://www.nordakademie.de/index.php?id=ap

Köllner Chaussee 11
25337 Elmshorn
http://www.nordakademie.de

Modellgetriebene Entwicklung von Schnittstellenprogrammen für die SOA Lösung SAP PI

Frank Zimmermann[1]
Michael Neuenstadt[2]
Jan Henning Banneitz[1,3]
Stephan Töpel[1,4]
Stefan Tanck[1,5]

[1] FH NORDAKADEMIE, Köllner Chausse 11, 25337 Elmshorn
[2] cimt AG, Burchardstr. 17, 20095 Hamburg
[3] RWE Dea, Überseering 40, 22297 Hamburg
[4] Philips Medizinsysteme GmbH, Röntgenstr. 24, 22335 Hamburg
[5] Otto GmbH & Co KG, Wandsbeker Str. 3-7, 22172 Hamburg

Zusammenfassung SAP bietet mit der Komponente PI eine Enterprise Application Integration-Plattform, die als Umsetzung einer Service Orientierten Architektur verstanden werden kann. SAP und Nicht-SAP Systeme werden über Nachrichten miteinander verbunden. Dabei können unterschiedliche Nachrichtenformate mit Adapterprogrammen in das interne SAP Format umgesetzt werden. In der SAP Auslieferung sind die Konverter für weit verbreitete EDI Formate nicht enthalten. In diesem Papier wird eine Open Source Lösung diskutiert, die es ermöglicht, flexibel Formate einzulesen und zu konvertieren. Dabei wird auf den modellgetriebenen Ansatz zurückgegriffen. Es wird gezeigt, welche Unterschiede zwischen einer generischen Softwarearchitektur und einer modellgetriebenen generierten Architektur bestehen. Wir beschreiben die Umsetzung auf Basis des Eclipse Modelling Projects. Dabei kommen die Komponenten Xtext und Xpand des Open Source Frameworks open ArchitectureWare zum Einsatz. In einer Bewertung werden bessere Wartbarkeit und bessere Testbarkeit als Vorteile des modellgetriebenen Ansatzes gegenüber einem generischen dargestellt.

1 Problemstellung

SAP Process Integration (PI)[6] ist eine Plattform für die Integration von SAP und Nicht-SAP Systemen. Sie wird für A2A[7] Szenarien und für B2B[8] Szenarien eingesetzt. SAP PI ist die Basis für die in der Realisierung befindliche Service orientierte Architektur und ermöglicht die Ausführung von anwendungsübergreifenden Geschäftsprozessen[9].

Die Anbindung der verschiedenen Anwendungen an PI erfolgt über Adapter, die den Nachrichtenaustausch über das spezielle Protokoll des angebundenen Systems realisieren und für eine Übersetzung der zu verarbeitenden Nachricht in das PI interne SOAP Format sorgen.

Die interne Verarbeitung von Nachrichten in PI, d.h. das Routing und die Konvertierung von einem Nachrichtenformat in ein anderes, ist XML basiert. Nachrichten in einem anderen Format müssen durch (Format-)Adapter beim Eingang in PI in XML konvertiert werden. Kunden, die SAP PI einsetzen, benötigen meist jedoch eine große Anzahl anwendungsspezifischer Adapter. Für einfache Konvertierungen, wie z.B. die Übersetzung von CSV Dateien in XML, liefert SAP die Module bereits mit aus. Handelt es sich allerdings um tiefer strukturierte EDI[10] Standards wie Edifact, ANSI X12, Tradacom oder VDA, so verweist SAP auf Partner wie Seeburger, Iway oder Informatica, die spezielle Adapterlösungen anbieten. Ist der Ankauf einer solchen kommerziellen Adapterlösung nicht gewünscht, so bleibt die Möglichkeit der Eigenentwicklung eines EDI zu XML Adapters. Solch ein EDI zu XML Konverter sollte mit möglichst vielen EDI Nachrichten in verschiedenen Versionen umgehen können, ohne dass jedes Mal eine komplette Neuentwicklung nötig ist.

[6] PI ist im Kern identisch mit der SAP Exchange Infrastructure (XI)
[7] Application to Application Anwendungen
[8] Business to Business Anwendungen
[9] vgl. SAP (2007)
[10] Electronic Data Interchange

2 Lösungsansatz

Die Modellgetriebene Softwareentwicklung bezeichnet ein Vorgehensmodell für die Entwicklung von Software, bei dem nicht direkt der ausführbare Programmcode erstellt wird, sondern zunächst auf die Problemstellung angepasste domänenspezifische Modelle erstellt werden. Aus diesen Modellen wird über hinterlegte Vorlagen ausführbarer Code generiert[11]. Adapterprogramme enthalten häufig auftretende Anwendungsbestandteile, die zwar umfangreich und aufwändig zu erstellen sind, aber wenig intellektuelle Leistung erfordern. Task Automation ist ein häufig genannter Grund für den Einsatz von Generatortechnologie und domänenspezifischen Sprachen[12], weshalb dieser Ansatz in diesem Papier verfolgt wird. Er kann als Entwicklung einer Softwareproduktlinie für Schnittstellenprogramme verstanden werden.

Für die Architektur einer Anwendung ist die geforderte Wiederverwendbarkeit der entscheidende Faktor. Insbesondere wird die Abstraktionsebene der verwendeten Klassen festgelegt. Je höher die geforderte Wiederverwendung, desto abstrakter werden die Klassen. Die OMG[13] verwendet vier Metaebenen (M0, M1, M2 und M3), um ihre Modellstrukturen zu beschreiben. Dabei beschreibt die höhere Ebene die jeweils darunterliegende. Tabelle 1 zeigt drei der vier Ebenen mit Beispielen. In dieses Schema werden die Programmartefakte einer generischen und einer modellgetriebenen Lösung einsortiert. Dies soll die unterschiedlichen Abstraktionsgrade verdeutlichen.

Ein konkreter Vorgang, zum Beispiel dass ein Kunde eine bestimmtes Produkt kauft, wird durch eine EDI Nachricht auf Ebene M0 beschrieben, in der die relevanten Daten über den Geschäftsvorfall enthalten sind.

Die Struktur dieser Nachrichten ist zum Beispiel in dem EDI Standard Edifact D97A ORDERS auf Metaebene M1 beschrieben. Durch den Übergang auf die Ebene M1 werden aus den Daten Informationen, die geschäftliche Bedeutung tragen.

In der Ebene M2 werden die Strukturen aller EDI Nachrichten beschrieben. In der Regel sind EDI Nachrichten aus Segmenten aufgebaut, die ihrerseits aus Datenelementen bestehen. Die Begriffe Segment und Datenelement tragen keinerlei fachliche Bedeutung, sondern sind rein technische Begriffe. In der modellgetriebenen Lösung werden die M2 Artefakte die Konzepte einer domänenspezifischen Sprache. Die Ebene M3 ist in der Umsetzung durch EMF Ecore realisiert, spielt aber für die Betrachtungen keine Rolle.

Tabelle 1. Metaebenen

Ebene	Domainobjekte	Beispiele	generischer Ansatz	modellgetriebener Ansatz
	Realität	Bestellung des Kunden Müller		
M0	Nachrichten	Konkrete Nachricht		
M1	Nachrichten-definition	D97A ORDERS	Datenbeschreibung, z.B. in Form von DB Inhalten	Modell des Nachrichtenstandards, generierte Klassen
M2	Generischer Nachrichtenaufbau	Nachricht, Segment, Datenelement	Klassen des Programms	DSL, Generatorvorlagen, Basisframework

Bei einer generischen Lösung des Problems müssen die Klassen des Programms auf der Ebene M2 gewählt werden, um die erforderliche Wiederverwendbarkeit zu gewährleisten. Diese Klassen besitzen also keine fachliche Bedeutung, sondern sind Produkt informationstechnischer Überlegungen. Diese Klassen werden Objekte der Metaebene 2 erzeugen oder interpretieren und dadurch in der Lage sein, Nachrichten der Ebene 1 zu verarbeiten. Diese 3 Metaebenen überspannenden Aufgaben erfordern ein hohes Abstaktionsvermögen vom Entwickler und lassen die Programme komplex und schwer wartbar

[11] vgl. Stahl u. a. (2006)
[12] vgl. Mernik u. a. (2005)
[13] vgl. Object Management Group (OMG) (2002)

werden. Der modellgetriebene Ansatz ermöglicht es, das Adapterprogramm auf der Metaebene M1 zu generieren. Die so erzeugten Klassen haben fachliche Bedeutung, was sie für einen Anwendungsfall besonders verständlich macht. Die Ebene M2 liegt dabei nicht als ausführungsfähiges Programm vor, sondern ist in dem domänenspezifischen Modell enthalten. Der Übergang von M1 Modell nach M1 Java Programm wird durch die Generierung gemäß der definierten Vorlagen gemacht. Diese stellen den generischen Code auf Ebene M2 dar. In dieser Architektur wird die häufig geforderte Trennung zwischen fachlichen Inhalten (M1) und technischen Komponenten (M2) besonders deutlich. Die Trennung vereinfacht die Aufteilung der Programmieraufgaben und verbessert den Entwicklungsprozess.

3 Umsetzung

Die Erstellung des Adapters erfolgte unter Verwendung von openArchitectureWare[14] (oAW). oAW ist Teil des Eclipse Modeling Projects[15] und stellt Generatorwerkzeuge zur Transformation von Modellen, die auf dem Eclipse Modelling Framework[16] (EMF) basieren, zur Verfügung. Durch den Einsatz des EMF sind flexible Bearbeitungsmöglichkeiten der Artefakte garantiert, so dass die Verarbeitbarkeit verschiedener EDI Standards gewährleistet ist.

Die für dieses Projekt relevanten Komponenten des oAW-Frameworks sind Xtext und Xpand. Xtext[17] ist ein Framework zur Entwicklung von textuellen domänenspezifischen Sprachen (DSL). Xtext benutzt kontextfreie Grammatiken in Extended-Backus-Naur-Form (EBNF), um die Syntax einer DSL zu beschreiben. Aus dieser Grammatik werden ein Parser, ein EMF Ecore Metamodell und ein Eclipse-Texteditor für die DSL generiert.

Xpand[18] ist eine Templatesprache für die Generierung von Artefakten aus EMF Modellen. Polymorphe Aufrufe innerhalb der Templates sowie die Verwendung von Aspekten sind möglich. Den Workflow der konkreten Umsetzung zeigt Abbildung 1.

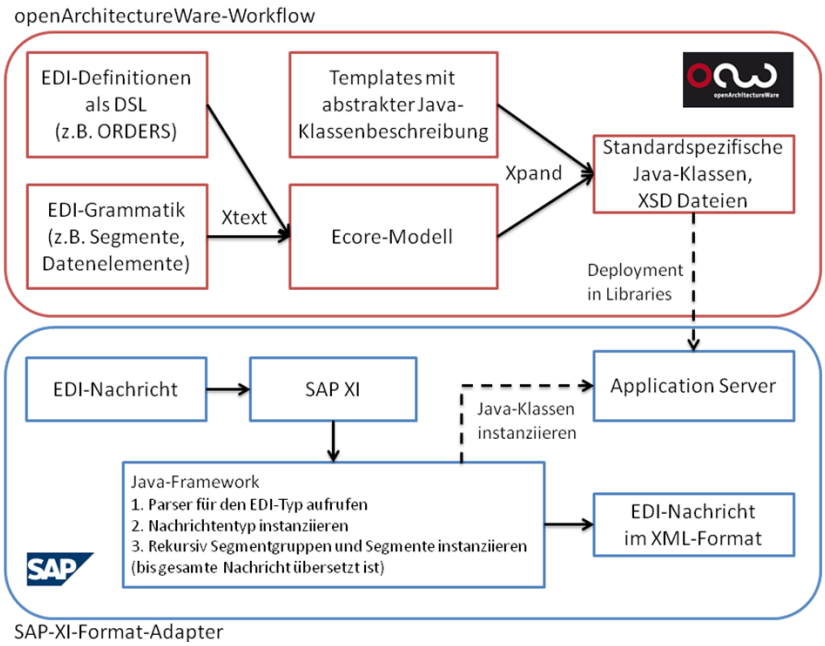

Abbildung 1. Workflow Adaptergenerierung

[14] http://www.openarchitectureware.org
[15] http://www.eclipse.org/modeling/
[16] http://www.eclipse.org/modeling/emf/
[17] vgl. Efftinge u. a. (2007)
[18] vgl. Efftinge u. a. (2007)

Für jeden EDI Standard wird eine Xtext-Grammatik als Ausgangspunkt benötigt. Diese Grammatik wird so gewählt, dass der zugehörige Parser die verfügbaren Beschreibungen der Nachrichtendefinitionen verarbeiten kann. Die Nachrichtendefinitionen beschreiben den Aufbau , Reihenfolge und Eigenschaften von Segmentgruppen und Segmenten mit ihren zugehörigen Datenelementgruppen und Datenelementen. Der Parser erzeugt als Ergebnis des Parsing Vorgangs aus der Nachrichtendefinition ein Modell als Instanz des zugehörigen Ecore Metamodells.

Dieses Modell wird benutzt, um gemäß der Xpand-Templates Java-Klassen für die einzelnen Nachrichtentypen, Segmentgruppen und Segmente zu generieren. Diese Java-Klassen besitzen Methoden, die die Umwandlung einer EDI-Nachricht in eine XML-Nachricht und umgekehrt ermöglichen. Ein zweites das Xpand-Template erzeugt XSD-Dateien für die jeweiligen Nachrichtentypen, die zur Validierung der einzelnen XML-Dateien benötigt werden.

Die Umwandlung einer konkreten EDI Nachricht geschieht durch ein manuell entwickeltes Java-Framework, dessen Klassen von den generierten Klassen erweitert werden. Dieses Basis-Framework enthält als Kern-Komponente zunächst einen Parser für die konkreten Nachrichten des jeweiligen EDI Standards. Beim Einlesen einer Nachricht durch den Parser werden zunächst der verwendete EDI Standard und der Nachrichtentyp bestimmt. Abhängig von der eingelesenen EDI-Nachricht wird die jeweilige generierte Java-Nachrichten-Klasse mittels eines Reflection-Mechanismus instanziiert. Im Zuge des weiteren Einlesevorgangs wird die Nachrichten-Klasse mit Segment- oder Segmentgruppen-Objekten versehen. Auch diese Objekte werden per Reflection gemäß der ausgelesenen Informationen der EDI-Nachricht erzeugt. Bei ihrer Erzeugung erhalten sie den Inhalt eines ausgelesenen Segments respektive einer Segmentgruppe als Konstruktor-Argument übergeben. Die weitere Zerlegung des übergebenen Inhalts erfolgt danach innerhalb des jeweiligen Objektes. Bereits während des Einlesevorgangs der EDI-Nachricht übernimmt die Nachrichten-Klasse die Umwandlung der eingelesenen Informationen in XML.

Das manuell entwickelte Java-Framework und die generierten Klassen werden in Libraries auf dem J2EE-Server der SAP XI deployed. Eingebunden wird der Adapter über eine J2EE stateless session bean, welche ebenfalls deployed wird. Diese bean wird beim Empfang einer EDI-Nachricht aufgerufen und stößt den Transformationsprozess der Nachricht an.

3.1 Projektablauf

Das Projektteam bestand aus sechs Studenten der Fachrichtung Wirtschaftsinformatik der FH Nordakademie in Elmshorn. Im Rahmen des Projektes wurden drei Projektteams á 2 Personen gebildet. Das Team Xtext war für die Erstellung und Optimierung der Grammatiken für die verschiedenen EDI Standards zuständig. Das Team Xpand erstellte die Templates zur Generierung der Java-Klassen und das Team Framework entwickelte die benötigten Kern-Komponenten, Interfaces und abstrakten Klassen. Um den Projekterfolg sicherzustellen, wurde Scrum als agiles Vorgehensmodell bei der Software-Entwicklung eingesetzt. Das Gesamtprojekt wurde in drei Iterationen von jeweils drei bis vier Wochen unterteilt. Insgesamt wurden pro Projektteam 1,5 Mann-Monate investiert.

4 Bewertung des modellgetriebenen Ansatzes

Wartbarkeit auf Ebene M2 ist bei beiden Ansätzen von der eigentlichen Implementierung abhängig. Interessant wird ein Vergleich, wenn es um Wartung auf Basis der Ebene M1 geht. In beiden Fällen gibt es natürlich die Möglichkeit der Anpassung des Modells an die fachlichen Bedürfnisse. Gibt es allerdings besondere Anforderungen, die sich nicht auf die Struktur der Nachrichten, sondern auf eine spezielle Verarbeitung beziehen, so kann man in generischem Fall nur eine generelle Lösung vorsehen, die möglicherweise nur in einem Spezialfall benötigt wird. In der Vergangenheit wird davon ausgegangen, dass generierte Programme nicht lesbar und vor allen Dingen nicht wartbar sind. Einmal generierter Code wurde nicht mehr verändert. Es wurden eine Reihe von Techniken entwickelt, um generierte Programme mit handgeschriebenem Code zu verbinden[19]. Dazu gehören:

- Verwendung von User Exits.

[19] vgl. Stahl u. a. (2006)

- Integration von geschützten Bereichen.
- Einsatz von Vererbung. Dabei ist sowohl die Variante generierter Code erbt von manuellem Code als auch die Variante manueller Code erbt von generiertem Code denkbar.
- Abgleich von generiertem Code mit handgeschriebenen Code mit JMerge.

Aufgrund der vielen Möglichkeiten ist davon auszugehen, dass im modellgetriebenen Ansatz Anforderungen auch auf der Ebene M1 leicht umgesetzt werden können. Modifikationen und Erweiterungen auf der Ebene M1 sind eine Metaebene konkreter als auf der Ebene M2. Das kann auch ein großer Vorteil sein, weil die Anpassung leichter wird. Erhöhung der Abstraktion ist ein häufig zitierter Vorteil modellgetriebener Softwareentwicklung. In dieser Situation erkennt man, dass die Konkretisierung des generierten Codes auch einen Vorteil darstellen kann.

Ein generischer interpretierender Ansatz hat immer das Problem, dass der Overhead der mit der Generalisierung verbunden ist, zu Performancenachteilen führt. Dieser Overhead wird jedoch gern in Kauf genommen, weil die Erstellungskosten einer generischen Lösung deutlich niedriger liegen als die von vielen spezifischen Lösungen. Ein generiertes Programm auf der Ebene M1 ist jedoch genau um diesen Overhead schlanker. Bei der großen Anzahl der zu verarbeitenden Nachrichten könnte sich dieser Vorteil auf messbare Beträge summieren. In keinem Fall jedoch sollte der generierte Code langsamer sein als ein generischer.

Neben der Generierung von Adaptern lassen sich auch weitere Artefakte problemlos generieren. Dazu gehören zum Beispiel XSDs und Testrahmen für den Unit- und Integrationstest. Diese Tests können dabei auch ohne eine umfangreiche Infrastruktur (Offline) durchgeführt werden, da die Anwendungen vollständig in den Programmen hinterlegt sind.

Das Modell der Nachrichtentypen wird nur zum Generierungszeitpunkt benötigt. Das vereinfacht das Deployment der Anwendung, weil zur Laufzeit weniger Artefakte benötigt werden.

5 Fazit

Mit dem oben beschriebenen Ansatz konnten Erfahrungen mit der modellgetriebenen Softwareentwicklung in Kombination mit SAP XI gesammelt werden. Die interne Aufteilung in kleine Projektteams erwies sich als vorteilhaft und machte die Kommunikation zwischen den Projektteams zu einem entscheidenden Erfolgsfaktor. Das agile Vorgehensmodell stellte kontinuierlich sicher, dass die Projektzeit nicht überschritten und die Anforderungen des Kunden korrekt umgesetzt wurden. Änderungswünsche konnten somit zeitnah eingebracht und integriert werden.

Die von SAP verwendete, ältere Java-Runtime 1.4.2 des J2EE-Application Servers stellte eine unerwartete Hürde im Projekt dar, weil keine Annotations und Generics im Java-Code verwendet werden können. Es werden auch zwei unterschiedliche Eclipse Instanzen benötigt, was eine integrierte Entwicklung von Adaptern unmöglich macht.

Der modellgetriebene Ansatz erwies sich als hilfreich für die Entwicklung des umfangreichen Projektes. Es musste nur äußerst wenig Code manuell geschrieben werden, ein Großteil konnte mithilfe der Templates generiert werden. Entscheidend ist auch die Einfachheit der generierten und manuell geschriebenen Klassen. Durch den modellgetriebenen Ansatz werden stark abstrahierte Java-Klassen, wie sie in einer für alle EDI Standards generischen Implementierung unverzichtbar wären, überflüssig. Eine gleichbleibend hohe Änderbarkeit und leichte Anpassbarkeit ist trotzdem gewährleistet.

Literaturverzeichnis

[Efftinge u. a. 2007] Efftinge, Sven ; Friese, Peter ; Haase, Arno ; Kadura, Clemens ; Kolb, Bernd ; Moroff, Dieter ; Thoms, Karsten ; Völter, Markus: *openArchitectureWare User Guide Version 4.2*. http://www.eclipse.org/gmt/oaw/doc/4.2/openArchitectureWare-42-reference.pdf. September 2007

[Mernik u. a. 2005] Mernik, Marjan ; Heering, Jan ; Sloane, Anthony M.: When and how to develop domain-specific languages. In: *ACM Comput. Surv.* 37 (2005), Nr. 4, S. 316–344. – URL http://fparreiras/papers/WhenHowDevelopDSL.pdf. – ISSN 0360-0300

[Object Management Group (OMG) 2002] Object Management Group (OMG): *Meta Object Facility (MOF) Specification*. http://www.omg.org/docs/formal/02-04-03.pdf. April 2002

[SAP 2007] SAP: *SAP NetWeaver Process Integration 7.1 Overview* . https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/706005a3-3bd6-2910-91ae-a2016239bdcf. October 2007

[Stahl u. a. 2006] Stahl, Thomas ; Voelter, Markus ; Czarnecki, Krzysztof: *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. – ISBN 0470025700