Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Refactoring einer Experience Base auf eine SOA-Architektur

Bachelorarbeit

im Studiengang Informatik

von

Stefan Radomski

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Nicola Henze

Hannover, 29. März 2006

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst habe und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 28.03.2005 Stefan Radomski

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Die Anforderungen an Softwaresysteme ändern sich während des Betriebes. So müssen Unternehmen flexibel auf Änderungen ihrer Domäne reagieren, um einen Vorteil gegenüber Mitbewerbern zu haben. Geschäftsprozesse müssen angepasst werden um neue Technologien und Entwicklungen zu berücksichtigen, Altsysteme müssen gepflegt und eventuell ersetzt werden.

Diese Flexibilität stellt Anforderungen an ein Softwaresystem, welche mit herkömmlichen Architekturen oftmals nur schwer zu erreichen sind. Die bestehenden Lösungen sind häufig auf einer speziellen Plattform ausgeführt und stark integriert. Schnittstellen zu anderen Softwaresystemen sind in der Regel nur innerhalb der jeweiligen Plattform verbreitet und aufwendig zu implementieren, sodass eine Integration mit Softwaresystemen anderer Anbieter schwierig ist.

Um die Eignung eines Softwaresystems zur Problemlösung in einer Domäne überhaupt bewerten zu können und sich ändernde Anforderungen festzustellen bedarf es einer Rückmeldung durch die Anwender des Systems. Es sollte eine Möglichkeit geben, Erfahrungen und Kritik für eine kontinuierliche Verbesserung des Systems den Entwicklern und anderen Anwendern zur Verfügung zu stellen.

Bisherige Ansätze hier einen „Feedback Cycle“ entstehen zu lassen, hatten oftmals das Problem einer zu hohen Hemmschwelle und fehlender Akzeptanz. Anwender müssen externe Programme bemühen, ihr Problem aufwendig kategorisieren und beschreiben.

Eine Möglichkeit direkt zu einem Prozessschritt aus dem angewendeten Programm heraus Anmerkungen und Erfahrungen zu hinterlegen ist selten gegeben, da eine Integration einer solchen Komponente aufwendig erscheint.

1.2 Problemstellung

Im Rahmen dieser Arbeit wird eine bestehende Experience Base[1] um Schnittstellen für Web Services erweitert, um als Teil einer “Service Oriented Architecture” (SOA) von anderen Applikationen genutzt werden zu können.

Die Funktionalität der Experience Base steht dann für andere Softwaresysteme bereit, um Erfahrungen und Kritik zu einem Prozessschritt aus der Applikation heraus zu hinterlegen. Erfahrungen und Expertenwissen stehen den Beteiligten dadurch unmittelbar zur Verfügung.

[1] T. Buchloh, “Erstellung eines Baukastens für Experience Bases”, 2005

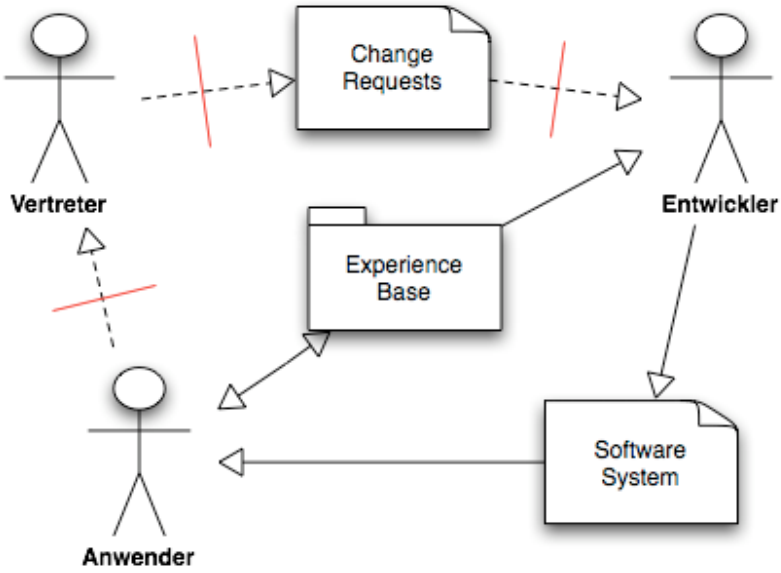

Abbildung 1: Die Experience Base im Feedback Cycle

Durch die direkte Kommunikation der Anwender untereinander und der Zuordnung der Erfahrungen zu einem konkreten Prozessschritt soll der Feedback Cycle des verwendeten Softwaresystems verkürzt und effektiver gestaltet werden. Darüber hinaus können Entwickler die Probleme der Anwender an einem bestimmten Prozessschritt identifizieren und in einer späteren Version des Softwaresystems beheben.
Um die Integration der Experience Base in andere prozessunterstützende Softwaresysteme zu verdeutlichen, wird eine prototypische Applikation mit eingebetteter Experience Base ausgeführt.

1.3 Struktur der Arbeit

Nach einer Einführung in die Begriffe und Konzepte der Domäne werden die Anforderungen an eine Erweiterung der bestehenden Experience Base und an einen prototypischen Nutzer formuliert.

Darauf folgt der Entwurf, in welchem ein Vorgehen beschrieben wird, um die Anforderungen umzusetzen. In diesem Rahmen wird auch die Plattform und Architektur der bestehenden Experience Base vorgestellt und aufgezeigt, wie eine Schnittstelle für eine Integration in eine SOA aussehen kann.

In der Realisierung werden verschiedene Implementierungen für benötigte Teilsysteme vorgestellt und bewertet. Diese werden dann benutzt, um die Konzepte des Entwurfes umzusetzen und das Softwaresystem zu erstellen.

Danach wird die Arbeit in der Retroperspektive nochmals betrachtet um Unzulänglichkeiten und Probleme zu identifizieren und diese für weitergehende Arbeiten im Bereich um Web Services auf der vorgestellten Plattform vermeiden zu können.

Abschließend erfolgt eine Zusammenfassung und ein Ausblick der Möglichkeiten der Web Services im Allgemeinen und der Organisation der Experience Base als Web Service Anbieter im Besonderen.

2. Begriffe und Konzepte

Um die Aufgabenstellung weiter zu konkretisieren und in die Domäne einzuführen ist es notwendig Begriffe und Konzepte zu benennen und im Rahmen dieser Arbeit definieren.

2.1 Refactoring

Unter Refactoring versteht man das Ändern der internen Struktur eines Softwaresystems, ohne dabei sein externes Verhalten zu ändern:

"Refactoring is the process of changing a Softwaresystem in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written."[2]

Im Rahmen dieser Arbeit wird allerdings nicht so sehr das interne Design der konkreten Experience Base verbessert, sondern es werden neue Schnittstellen geschaffen, um diese in eine Service Oriented Architecture zu integrieren.

2.2 Experience Base

Eine Experience Base ist ein Sammelbecken für Wissen und Erfahrungen, welche vor allem durch die kontinuierliche Anwendung eines wiederkehrenden Prozesses gewonnen werden. Die Experience Base bietet dabei den Beteiligten die Möglichkeit gewonnene Erfahrung zu externalisieren und anderen Interessierten zur Verfügung zu stellen. Die Experience Base ist Teil eines Gesamtkonzeptes, der Experience Factory, welche es Unternehmen ermöglichen soll Analysemethoden und Bewertungskriterien für den Zustand eines Prozesses zu gewinnen.[3]

Die Experience Base als Forum für Anwender eines Softwaresystems sollte dabei möglichst eng in das System selbst eingebunden sein und sich an dem modellierten Prozess orientieren, um eine Bedienung zu vereinfachen und die Hemmschwelle für Beiträge möglichst gering zu halten. Ein solches Forum in die Applikation integriert würde darüber hinaus das Ausprägen einer User Community begünstigen, in der die Anwender Hilfestellung zu Prozessschritten oder relevanten Problemstellungen bekommen können.

2.3 Artefakte

"Produkt (Dokument, SourceCode, Programm), welches als Zwischen- oder Endergebnis der Softwareentwicklung entsteht"[4]

In der Softwareentwicklung werden die im Verlauf eines Prozesses entstehenden Produkte Artefakte genannt. Da dies wenig konkret ist kommt es für diese Arbeit zu einer Mehrdeutigkeit.

---

[2] M. Fowler "Refactoring: Improving the Design of Existing Code", 1999.
[3] Vgl. V. R. Basili, G. Caldiera, H. D. Rombach, The Experience Factory, University Maryland, 1994.
[4] http://de.wikipedia.org/wiki/Artefakt

Einerseits werden die Vorlagen und Beispiele, welche für einen modellierten Prozess in der Experience Base angeboten werden, Artefakte genannt, andererseits aber auch die benötigten Dateien welche im Verlauf des Entwickeln des eigentlichen Softwaresystems erstellt werden müssen. Beide Teilaspekte behandeln Artefakte, und belegen den Begriff mit schwer zu vereinbarenden Bedeutungen.

Gerade für die später vorgestellte Plattform des Softwaresystems stellt das Verwalten und Generieren zugehöriger Artefakte ein besonderes Problem da. Entsprechende Systeme zum Automatisieren dieses Vorganges nutzen oftmals den Begriff Artefakt in ihrer Dokumentation.

Da vergleichsweise mehr Zeit auf die Vorstellung der Artefakte des eigentlichen Softwaresystems verwendet wird, wird der Begriff Artefakt für die Vorlagen und Beispiele in der Experience Base vermieden und durch “Vorlagen und Beispiele” ersetzt.

2.4 Service Oriented Architecture

Eine Service Oriented Architecture (SOA) ist eine bestimmte Sicht auf ein Softwaresystem. Sie motiviert sich dabei nicht so sehr aus Problemstellungen des Software Engineerings sondern vielmehr aus dem Versuch, Aktivitäten eines Unternehmens prozessorientiert zu modellieren.

Ein Business Analyst überführt dabei die Aktivitäten eines Unternehmens in eine Beschreibung der verwendeten Prozesse. Diese werden dann von einem Software Architekten als ein System von Dienstleistern und Dienstnutzern modelliert, welche über eine gemeinsame Schnittstelle kommunizieren.

So wird zum Beispiel für einen Prozess zum Bestellen einer Ware ein Schritt ausgeführt, in dem die Bonität des Kunden oder die Validität der Kontaktinformationen geprüft wird. Das Arbeitsergebnis eines solchen Prozessschrittes wird als Dienstleistung von einem Service Provider aufgefasst. Dieser Dienst kann wieder als Prozess verstanden werden, welcher weitere Dienste anderer Dienstleister (Service Provider) nutzt.

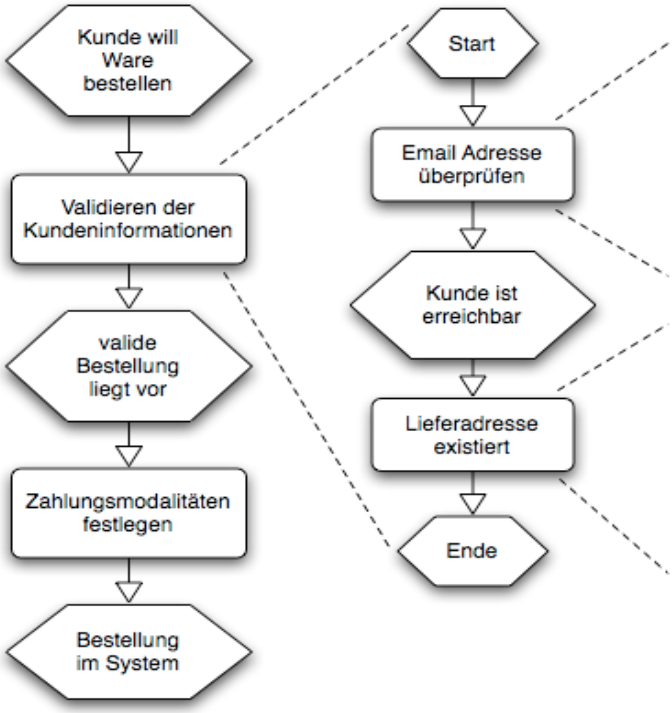

Abbildung 2: Ausschnitt einer Prozesskette

Auf diese Art entsteht eine Menge an Diensten, welche Problemstellungen aus der Domäne des Unternehmens lösen und von anderen Komponenten genutzt werden können. Jetzt ist es vorstellbar, dass ein Business Architekt durch ein geeignetes Zusammenstellen dieser Dienste (Orchestrierung) neue oder verbesserte Prozesse des Unternehmens abbilden kann. Darüber hinaus können ausgezeichnete Dienste auch anderen Unternehmen angeboten werden.

Es gibt verschiedene Technologien, um eine SOA aufzubauen. Ansätze aus der Vergangenheit hatten oftmals das Problem, dass durch mangelnde Standardisierung und einem hohen Implementierungsaufwand die Verbreitung nur innerhalb einer Plattform stattfand. Viele Dienste wurden nicht mit entsprechenden Schnittstellen ausgestattet, sodass sich Synergieeffekte nicht voll entfalten konnten.

Um die aufgezeigten Probleme zu minimieren, kann man eine SOA als Menge von "Web Services" (WS) aufbauen. Dienste für den modellierten Prozess werden dabei über ein standardisiertes Protokoll als Komponenten für andere Softwaresysteme angeboten. Durch eine Standardisierung können Implementierungen für Softwaresysteme der verschiedensten Plattformen angeboten werden.
Ein Refactoring von Altsystemen beschränkt sich dann auf das Umsetzen der konkreten Schnittstelle zwischen Altsystem und der Implementierung der Web Services.

2.5 Web Services

"A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description"[5]

[5] Aus IBM - Web Services Conceptual Architecture (WSCA 1.0)

Mit Web Services lassen sich Komponenten einer SOA realisieren. Der Vorteil einer solchen Ausführung liegt vor allem in der einfachen Integrierbarkeit der Funktionalität in andere Komponenten und der Austauschbarkeit durch eine formale Beschreibung der Schnittstellen. Ein Web Service würde dann zum Beispiel Dienste zum Validieren der Kundeninformationen anbieten.

Die Hoffung ist, dass Web Services für Softwaresysteme einen ähnlichen Effekt erzeugen, wie das World Wide Web (WWW) für menschliche Anwender. Ein Anwender nutzt die Dienste des WWW, um Informationen für die Lösung einer Problemstellung zu erhalten. So kann man zum Beispiel Fahrplaninformationen der Eisenbahn oder Suchergebnisse zu Schlüsselwörtern im WWW abrufen.

Diese Informationen werden im klassischen WWW mit der Hypertext Markup Language (HTML) für die grafische Aufarbeitung durch einen Browser beschrieben. Domänenspezifische Informationen über die Informationen selbst werden nicht übertragen, wodurch eine Weiterverarbeitung außerhalb der Domäne der grafischen Aufarbeitung im Allgemeinen nicht möglich ist.

Im Rahmen einer SOA sind Web Services dezentrale Dienstanbieter, die jederzeit gegeneinander austauschbar sind. Dabei definieren Entwickler von Systemen nach und nach ein Modell für die Dienste einer abzubildenden Domäne und standardisieren die Schnittstellen. So wäre es zum Beispiel möglich, in einer Applikation aus einer auf diese Art formalisierten Domäne, die Dienstanbieter untereinander zu tauschen, Dienste selber zu implementieren oder die Funktionalität von anderen Anbietern zu beziehen.

Web Services selbst werden typischerweise in einer speziellen Architektur ausgeführt, um die Austauschbarkeit im Betrieb zu ermöglichen und Anfragen an einzelne Dienste nach Bedarf delegieren zu können.

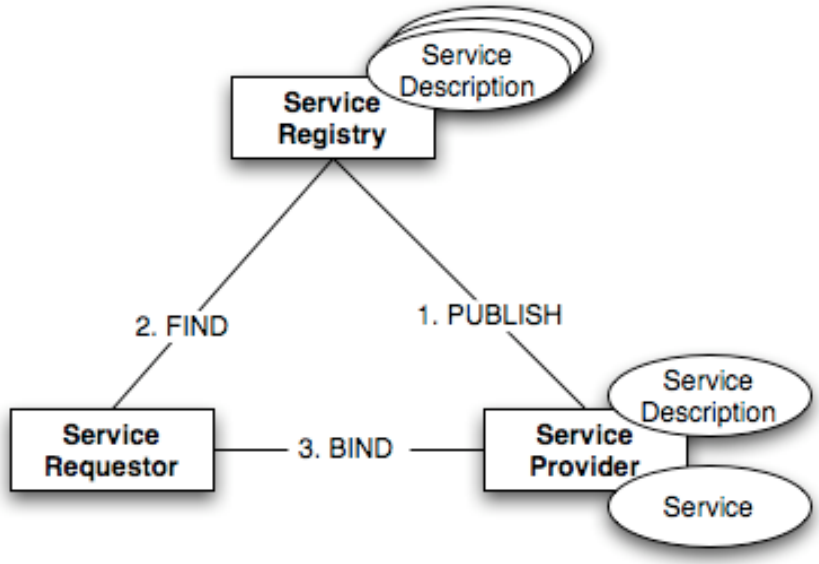

Abbildung 3: Web Service Architektur

Eine Implementierung eines Dienstes (Service Provider) registriert ihre Schnittstelle (Service Description) in einer Service Registry. Nutzer der Dienste (Service Requestor) beziehen dann aus der Service Registry die Adressen der Service Provider. Mit dem Wissen um die Adresse eines Service Providers kann der Service Requestor dann letztlich den Dienst nutzen.

2.5.1 Protokolle

Die Kommunikation zwischen Softwaresystemen im Internet ist als Schichtenmodell organisiert. Während die unteren Schichten Details der Übertragung einzelner Bits regeln, realisieren die höheren Schichten Dienste wie Fehlerkorrektur oder Verbindungsmanagement. Web Services definieren nun weitere Schichten oberhalb der Anwendungsschicht in einem solchen Schichtenmodell und nutzen damit die Eigenschaften und Dienste der darunter liegenden Schichten.

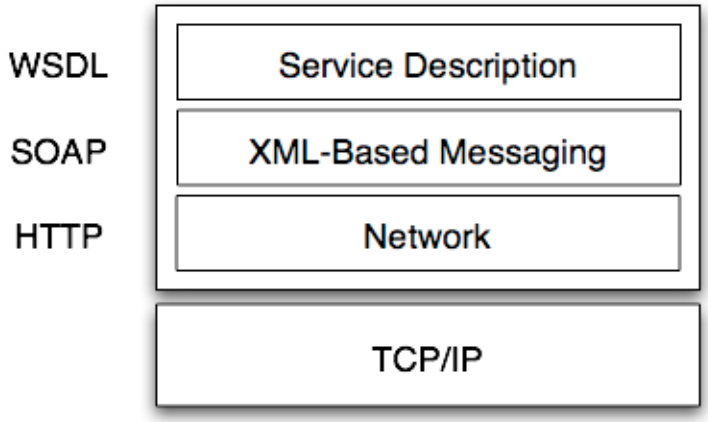

Abbildung 4: Web Service Stack

Als Protokoll der Netzwerkschicht wird für Web Services vorwiegend HTTP genutzt. Dieses findet auch im WWW Verwendung, um die Beschreibungssprache für Internetseiten HTML zu transportieren. Aber auch andere Protokolle sind denkbar, solange sie die Anforderungen an Datenintegrität und asynchroner Kommunikation erfüllen. So kommen auch Protokolle für der Dateiübertragung (FTP) oder sogar Protokolle für den Versand von Emails (SMTP) zum Einsatz.

Durch eine Standardisierung der verwendeten Technologien und Protokolle soll eine Interoperabilität der Implementierungen gewährleistet werden. Dadurch kann zum Beispiel ein Service Provider auf einem Mainframe und der Service Requestor auf einem Mobilfunktelefon ausgeführt werden. Das Gebiet um Web Services ist allerdings noch recht jung und wenn auch die Kerntechnologien gut verstanden und umgesetzt sind, so gibt es doch immer noch viele Bewegungen in unmittelbar angrenzenden Gebieten.

2.5.1.1 HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein sehr verbreitetes und etabliertes Protokoll für die Kommunikation zweier Softwaresysteme und findet vor allem im WWW Verwendung. Im Juni 1999 wurde es in der Version 1.1 vom W3C, dem Standardisierungsgremium des WWW festgeschrieben.

2.5.1.2 XML

"XML, the Extendible Markup Language, is a W3C-endorsed standard for document markup. It defines a generic syntax used to mark up data with simple, human-readable tags."[6]

XML definiert also eine Syntax, um Daten mit Metainformationen zu bereichern. Dadurch wird eine Serialisierung und Weiterverarbeitung der Daten durch andere Softwaresysteme der Domäne ermöglicht. Die spezielle Semantik der Metainformationen wird dabei in weiteren Standards wie zum Beispiel HTML oder auch SOAP beschrieben.

2.5.1.3 SOAP

"SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols."[7]

Das "Simple Object Access Protocol" (SOAP) definiert Dienste, um einen Austausch strukturierter Informationen zwischen zwei SOAP Implementierungen zu ermöglichen. SOAP nutzt dabei XML um diese Informationen zu beschreiben.

2.5.1.4 WSDL

Web Service Description Language (WSDL) ist die Beschreibungssprache für eine Service Description. Sie wurde dem W3C in der Version 1.1 zur Standardisierung vorgelegt und befindet sich zum Zeitpunkt der Entstehung dieser Arbeit noch in der Diskussion[8].

Letztendlich ist WSDL ein XML Dokument, welches typischerweise per HTTP zur Verfügung steht und die Details für eine Kommunikation zweier Web Service Implementierungen bereitstellt. Entsprechende Werkzeuge sind in der Lage, aus dieser Datei den Quellcode für eine Einbindung des Dienstes in ein anderes Softwaresystems zu generieren.

WSDL beschreibt Services als Menge von Ports. Diese bestehen aus einer Adresse des Service Providers im Netzwerk und einer Verknüpfung (Binding) zwischen definierten Datentypen und Operationen. Durch die Verknüpfung wird für einen Port festgelegt, welche Schnittstelle er beschreibt. Es macht Sinn zwischen der Definition eines konkreten Service Providers und seiner Schnittstelle zu unterscheiden. Damit wird es möglich, dass sich weitere Standards entwickeln, um Dienste und Datentypen von Softwaresystemen einer Domäne zu beschreiben und zu formalisieren.

2.5.1.5 UDDI

"The focus of Universal Description Discovery & Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services."[9]

---

[6] Elliotte Rusty Harold, W. Scott Means "XML in a nutshell" 2004.
[7] http://www.w3.org/TR/soap12-part1/
[8] http://www.w3.org/TR/wsdl
[9] http://uddi.org/pubs/uddi_v3.htm

Universal Description Discovery & Integration wurde ursprünglich von IBM und Microsoft entwickelt und beschreibt eine Menge an Diensten, um dynamisch Anbieter eines Web Services zu finden und auf der anderen Seite implementierte Dienste für Nutzer bekannt zu machen. In der oben beschriebenen Web Service Architektur beschreibt es die Schnittstelle zur Service Registry.

Die Vision von UDDI ist eine Art weltweites Branchenverzeichnis aus welchem sich ein Service Requestor den passenden Dienst raussuchen kann. Dabei werden die Beschreibungen der angemeldeten Services in einem Katalog für Suchanfragen aufgearbeitet.

Im Rahmen dieser Arbeit soll dieser Aspekt nicht weiter behandelt werden. Für ein Entwickeln von Web Services mit bekannten Adressen und lokalen Service Providern rechtfertigt die gewonnene Flexibilität wohl nicht das Aufsetzen und Pflegen einer Service Registry.

3. Anforderungen

Das zu erstellende Softwaresystem besteht aus der Erweiterung der vorhandenen Experience Base zum Service Provider und einer Proof-of-Concept Applikation als Service Requestor.

3.1 Experience Base als Service Provider

Um die Effekte einer Experience Base voll zu entfalten, wurde untersucht, welche Funktionalität mindestens angeboten und welche Vorraussetzungen eintreten müssen, damit ein Mehrwert entsteht[10]. Die vorliegende Experience Base bietet einen großen Teil dieser Funktionalität an und soll nun als Service Provider diese Funktionalität als Web Services zur Verfügung stellen.

Um den evolutionären Ansatz von SOA mit Web Services zu illustrieren, aber auch vor allem um keine Fehler in den bestehenden Quellcode einzuführen, ist es sinnvoll möglichst wenig Veränderungen direkt an der Experience Base vorzunehmen.

Wenn man auch die gesamte Funktionalität der Experience Base als einen großen Web Service anbieten kann, so ist es doch ratsam, diese weiter zu unterteilen und zu einzelnen Services zu gruppieren. Zukünftige Nutzer der Dienste können dann Teilaspekte der Funktionalität bei anderen Anbietern beziehen und zum Beispiel das Verwalten der Beispiele und Vorlagen auslagern.

Dabei ist darauf zu achten, dass ein Nutzen des Dienstes mit möglichst wenig Anfragen möglich ist. Durch die weiteren Protokollschichten und das Programmiermodell ist ein Service Request relativ teuer. Der vergleichsweise geringe Nutzdatenanteil einer Anfrage kann entsprechend durch größere Datenmengen kompensiert werden.

3.2 Proof-of-Concept Service Requestor

Die Implementierung des Service Requestors als Nutzer der Web Services soll zum einen die Funktionsweise und das Programmiermodell der Web Services illustrieren, zum anderen aber auch aufzeigen, wie eine Integration der Funktionalität der Experience Base in ein prozessunterstützendes Softwaresystem realisiert werden kann.

[10] K. Schneider, T. Schwinn, Maturing Experience Base Concepts at DaimlerChrysler, 2001

4. Entwurf

Für den Entwurf ist es notwendig, die vorhandene Architektur im Rahmen des Nötigen vorzustellen und Schnittstellen aufzuzeigen, um die Funktionalität verfügbar zu machen.

4.1 Plattform

Die vorliegende Experience Base von Tobias Buchloh ist als Applikation auf der Java 2 Enterprise Edition (J2EE) Plattform ausgeführt. Die eigentliche Funktionalität wird hierbei mit so genannten Enterprise JavaBeans (EJBs) beschrieben, diese werden dann in einem Application Server eingespielt (Deployment).

4.1.1 Application Server

Ein Application Server stellt eine Laufzeitumgebung für J2EE Applikationen bereit. Für die vorliegende Experience Base kommt JBoss 4.0.2 zum Einsatz, ein freier Application Server welcher von JBoss Inc. gepflegt und weiterentwickelt wird.[11]

Die EJB Spezifikationen von Sun bestimmen, welche Systeme ein Application Server anbieten muss und wie ein Deployment von Enterprise JavaBeans im Detail organisiert ist. So bietet ein Application Server verschiedene Schnittstellen an, um auf die Funktionalität in den EJBs zuzugreifen, verwaltet den Lebenszyklus der EJBs und stellt Dienste für die Persistenz spezieller EJBs zur Verfügung.

4.1.2 Enterprise JavaBeans

Enterprise JavaBeans (EJB) bilden die Komponenten für J2EE Applikationen. Im Rahmen dieser Arbeit werden die nötigen Konzepte der EJB Spezifikationen in Version 2.1 vorgestellt. Die Spezifikationen sind mittlerweile in Version 3.0 verfügbar.[12]

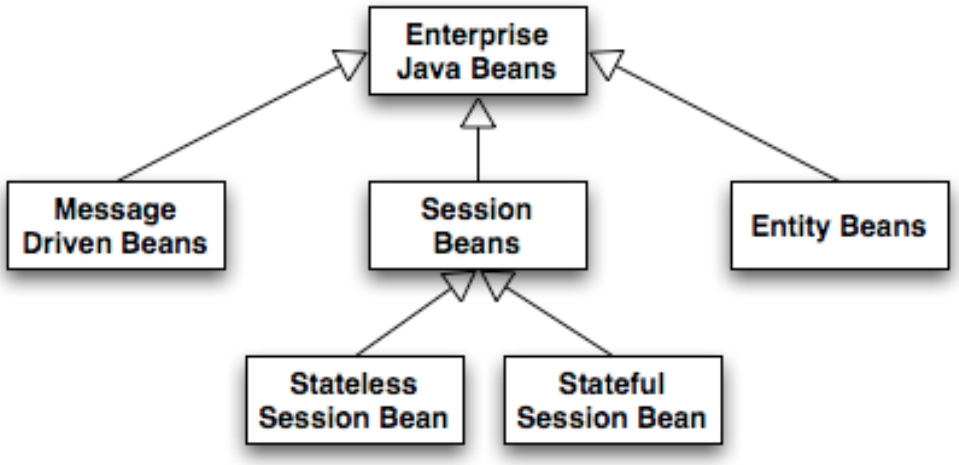

Abbildung 5: Hierarchie der Enterprise JavaBeans

Enterprise JavaBeans (EJB) kommen in drei verschiedenen Arten vor, SessionBeans, Message Driven Beans und Entity Beans. Allen gemein ist, dass sie für sich selbst

[11] http://www.jboss.com
[12] http://java.sun.com/products/ejb/docs.html

noch keinen Nutzen haben, sondern von einem Application Server nach Bedarf instanziiert und verwaltet werden.

Message Driven Beans werden für eine asynchrone Kommunikation mit angrenzenden Systemen verwendet. Entity Beans sind persistente Datenobjekte in einer Datenbank. Session Beans zergliedern sich nochmals in zustandsbehaftete und zustandslose Beans.
Eine Session Bean ist ein Objekt, welches eine Interaktion eines Akteurs mit dem System im Application Server darstellt. Für diese Arbeit sind nur zustandslose Session Beans (SLSB) als spezielle EJBs interessant, da die Spezifikationen nur für diese eine Schnittstelle als Web Service beschreiben.

SLSBs sind dabei konkret Klassendefinitionen, welche neben den Methoden mit der eigentlichen Funktionalität noch Methoden für ein Verwalten durch den Application Server definieren müssen. Dazu kommen Schnittstellenbeschreibungen in Form von Java Interfaces, um die Sichtbarkeit der Methoden für angrenzende Systeme zu beschreiben.

4.2 Architektur der bestehende Experience-Base

Die Applikation ist in die drei klassischen Schichten Persistenz, Präsentation und Serviceschicht einer 3-tier Applikation eingeteilt. Das Datenmodell der Persistenzschicht wird hierbei von der Geschäftslogik in der Serviceschicht losgelöst, sodass mehrere Komponenten dasselbe Datenmodell nutzen können, ohne unnötige Abhängigkeiten zu erzeugen. Die Präsentationsschicht nutzt dann die Serviceschicht um auf die Geschäftslogik zuzugreifen und die Daten grafisch aufzuarbeiten.

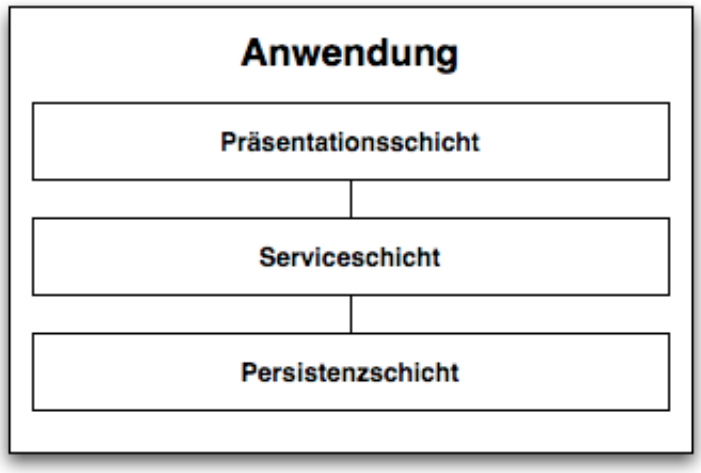

Abbildung 6: Typische Komponenten einer 3-tier Applikation

Eine Schnittstelle für Web Services kann in dieser Architektur als weitere Präsentationsschicht aufgefasst werden, wobei die Daten nicht einem Benutzer, sondern einer anderen Komponente in geeigneter Weise präsentiert werden.

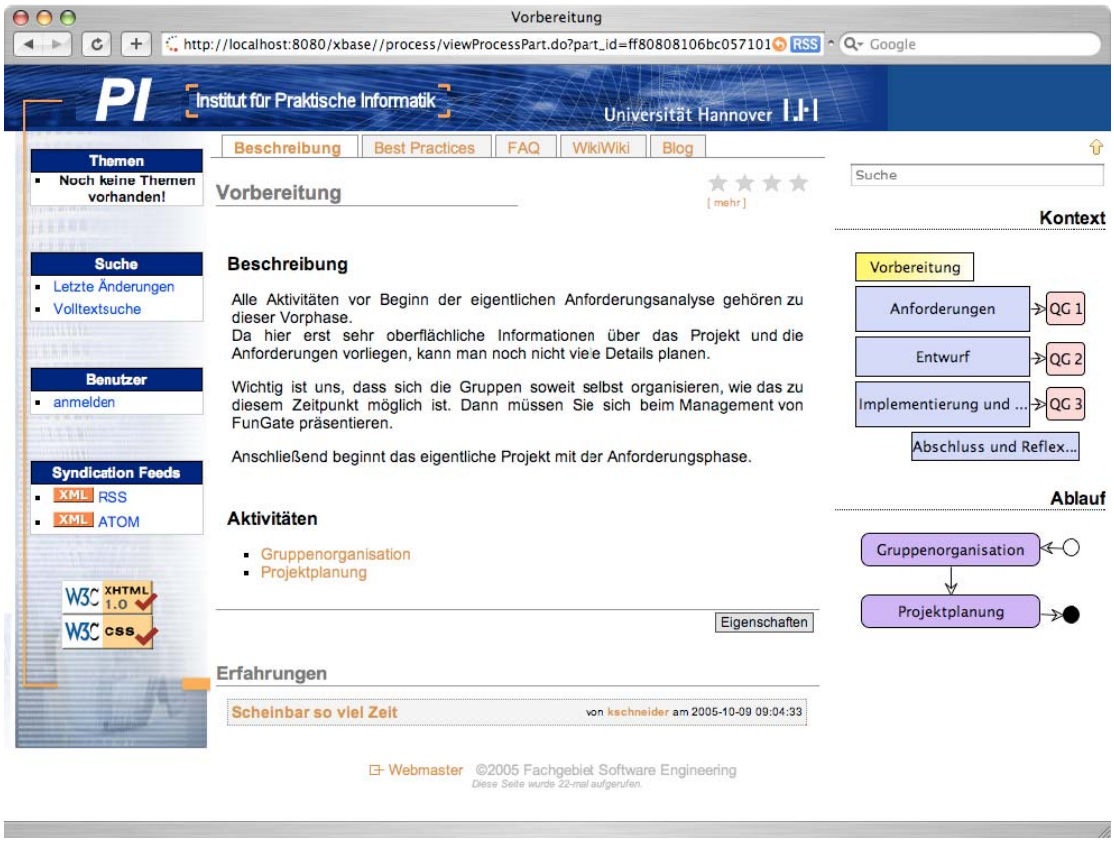

Abbildung 7: Weboberfläche der Experience Base

4.2.1 Bestehende Serviceschicht

Die Experience Base bietet verschiedene Dienste in der Serviceschicht, diese können über mehrere zustandslose Session Beans (SLSB) genutzt werden. Am Wichtigsten für die Aufgabenstellung ist der `ProcessPartService`, und die Session Beans für die Dienste des Feedbacksystems.

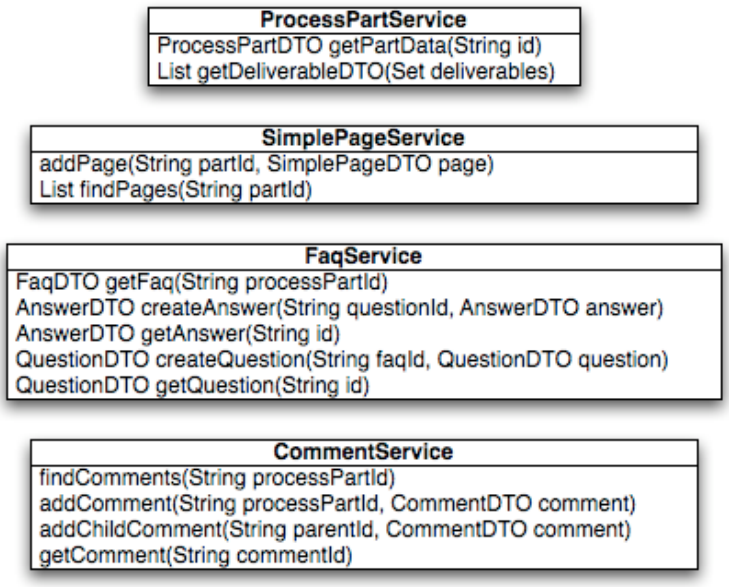

Abbildung 8: Ausschnitt aus bestehenden Session Beans für FeedbackServices

Mit dem Wissen um eine `ProzessPartId` kann man mit Hilfe der oben vorgestellten SessionBeans alle nötigen Datenobjekte beim System anfordern.

4.3 Datenmodell

Leider ist es nicht unmittelbar möglich, beliebige Objekte mit Web Services zu nutzen. Für spezielle Datentypen muss eine Serialisierung angegeben werden, sodass eine Web Service Implementierung diese Datentypen als WSDL für Nutzer des Dienstes beschreiben kann. Ein solcher Nutzer deserialisiert die Daten dann zu einem neuen Objekt.

Die betrachteten Web Service Implementierungen für die J2EE Plattform erlauben es, neben den nativen Datentypen auch Java Beans zu übermitteln. Eine Java Bean (nicht zu verwechseln mit einer Enterprise JavaBean) ist ein einfaches Datenobjekt. Es bietet ausschließlich Methoden an, um Attribute zu setzen und auszulesen. Für die Implementierung des Service Providers werden die Objekte aus der vorhandenen Experience Base in eine Hierarchie einfacher Java Beans überführt.

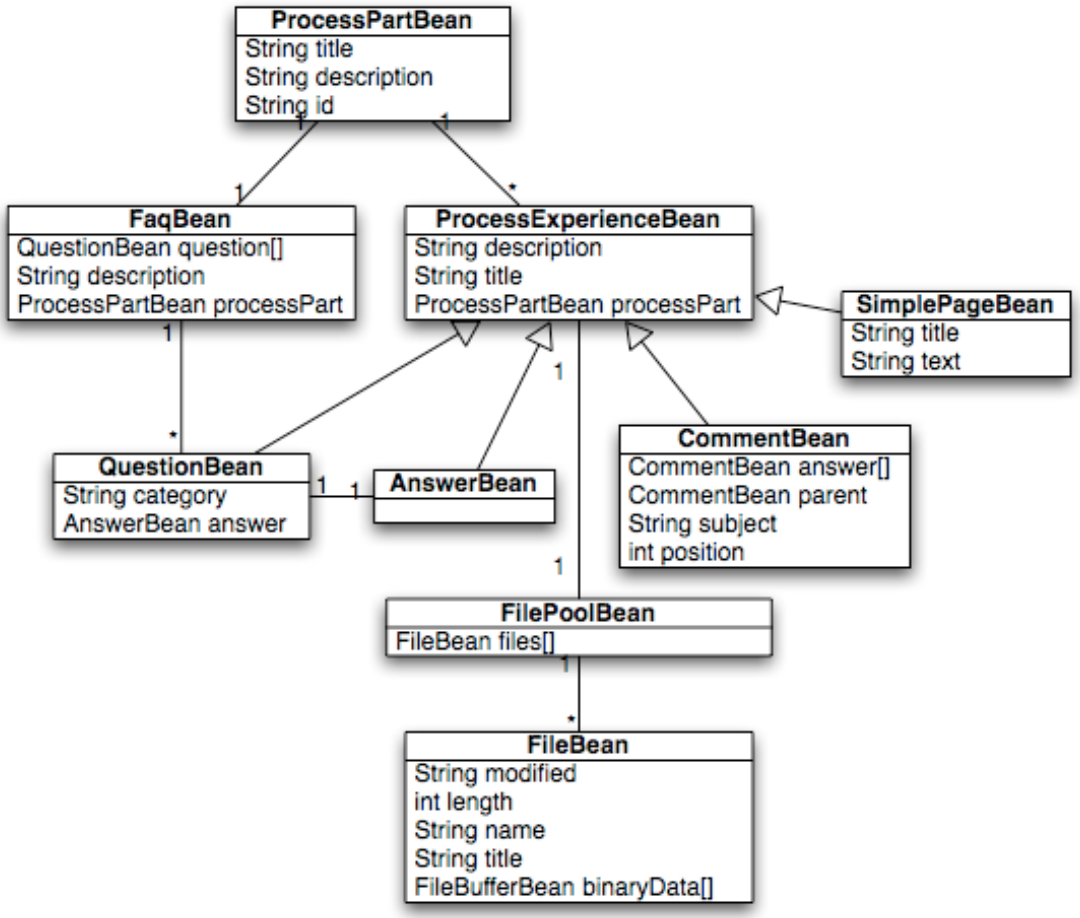

Abbildung 9: Datenmodell der Web Services

Das Datenmodell orientiert sich stark an dem bestehenden Modell der Experience Base. Für einen gegebenen Prozessschritt gibt es Prozesserfahrungen und eine Sammlung von häufig gestellten Fragen (FAQ) mit Antworten. Prozesserfahrungen sind Kommentare, Einträge in der FAQ oder beliebige freie Texte als SimplePages.

Neben den textuellen Erfahrungen kann eine Prozesserfahrung auch eine Menge an Dateien bereitstellen. Das `FilePoolBean` modelliert diesen Teil des Datenmodells.

Eine Beschreibung in der WSDL betrachtet allerdings nicht die Vererbungshierarchie, sodass für den Service Requestor eine flache Hierarchie entsteht. Dabei werden die Namensräume der Objekte vereint und Attribute der Elternklasse eventuell verdeckt. Dieses Problem wird für diese Arbeit einfach dadurch gelöst, dass entsprechende Attribute der Elternklasse mit einem Präfix versehen werden und nun auch für den Service Requestor zur Verfügung stehen.

4.4 Service Provider

Um eine Schnittstelle für Web Services in die oben beschriebene Architektur der bestehenden Experience Base zu integrieren, bietet es sich an, neue Stateless Session Beans (SLSB) als Fassade vor den vorhandenen SLSBs zu implementieren. Diese können dann die Schnittstelle neu zusammenstellen und als Web Services anbieten.

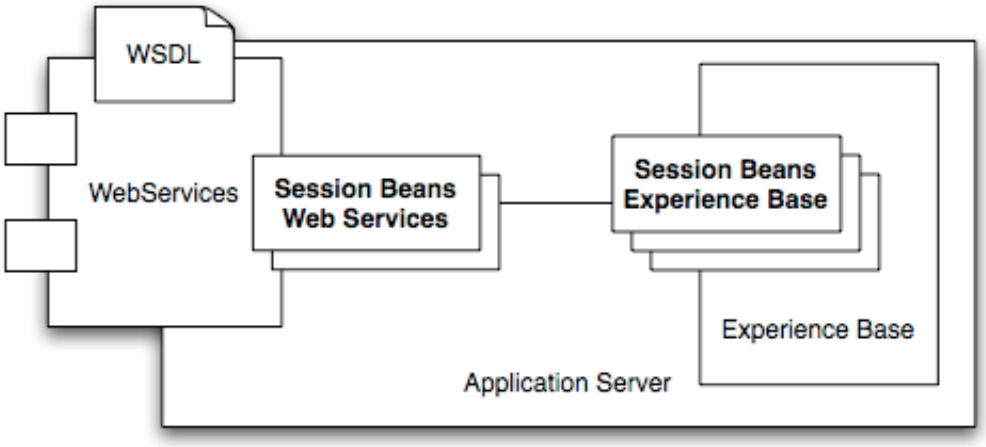

Abbildung 10: Neue SLSBs für die Web Services

Es ist darauf zu achten, dass die Dienste der neuen SLSBs eine geeignete Granularität haben. Wenn man viele kleine Operationen anbietet, kann Funktionalität gezielter abgerufen werden. Durch den vergleichsweise hohen Verwaltungsaufwand für die Kommunikation mit einem Web Service empfiehlt es sich, die Anzahl der Service Anfragen möglichst gering zu halten und möglichst viele Informationen auf einmal zu übertragen.

Eine Alternative zum Bereitstellen neuer SLSBs für die Web Services wäre es, direkt ausgesuchte Methoden der bestehenden SLSB als Web Services verfügbar zu machen. Typischerweise besitzt die angebotene Funktionalität der bestehenden Serviceschicht aber keine geeignete Granularität und wurde für tradierte Komponentenschnittstellen entworfen.
Um nun eine Lösung aufzuzeigen, wie eine bestehende Serviceschicht zu benötigten Dienste zusammengestellt werden kann, wird der Ansatz mit den neuen Session Beans auf der bestehenden Serviceschicht gewählt.

4.4.1 Aufteilung auf Web Services

In der bestehenden Experience Base wird ein Prozess als Menge von abhängigen Prozessschritten modelliert. Zu jedem Prozessschritt kann der Anwender verschiedene

Arten von Rückmeldungen geben oder lesen. So kann man zum Beispiel beliebige Freitexterfahrungen hinterlegen und kommentieren, Antworten auf häufig gestellte Fragen lesen oder andere Freitexterfahrungen wie Best Practices einsehen. Zu jedem Prozessschritt kann eine Menge an Dateien hinterlegt werden, welche von den Prozesserfahrungen als Vorlagen oder Beispielen referenziert werden können.

Die Aufteilung der Funktionalitäten auf die Web Services orientiert sich an der Trennung zwischen den textuellen Erfahrungen einerseits und den Vorlagen oder Beispielen für die Dokumente eines Prozessschrittes andererseits.

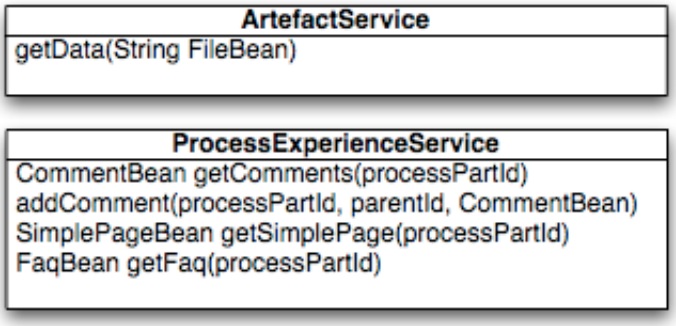

Abbildung 11: Die angebotenen Web Services

Web Services sind nur für zustandslose Interaktionen mit dem Service Provider definiert, daher muss eine Anfrage an den Dienst immer die behandelten Objekte eindeutig identifizieren.

Administrative Aspekte der Experience Base wie beispielsweise Löschen oder Bearbeiten von Benutzern oder Daten werden im Rahmen der Aufgabenstellung nicht modelliert. Es ist allerdings denkbar, dass sie in den dargestellten Entwurf integriert werden können.

4.4.1.1 ProcessExperienceService

Dieser Dienst bietet Möglichkeiten, um verschiedene Freitexterfahrungen zu einem gegebenen Prozessschritt zu hinterlegen oder einzusehen. Dies sind, wie oben dargestellt, Kommentare, die FAQ und simple Seiten statischen Inhalts. Die Methoden des Dienstes sollen kurz mit ihren Datenstrukturen vorgestellt werden.

Die Funktionalität für die Kommentare wird von den Methoden `getComments()` und `addComment()` bereitgestellt. Kommentare als Freitexterfahrung, können wieder kommentiert werden, was eventuell zu einem Diskussionsprozess führt. Um die Anzahl der Anfragen gering zu halten ist es nicht möglich einzelne Kommentare gezielt zu selektieren. Eine Anfrage bezüglich der Kommentare liefert immer den Baum aller Kommentare zu einem Prozessschritt. Es ist Aufgabe des Service Requestors diese geeignet zu verarbeiten und darzustellen.

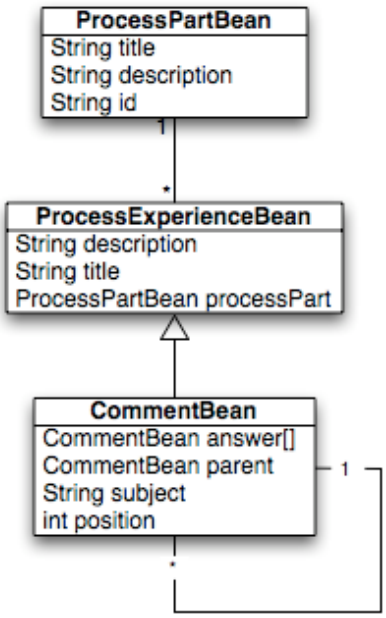

Abbildung 12: Datenstruktur für getComments

Die FAQ zu einem Prozessschritt kann ein Service Requestor mit `getFaq()` anfordern. Auch hier wird, wie bei den Kommentaren, die gesamte Hierarchie der Datenobjekte übertragen. Eine Selektion einzelner Fragen oder Antworten aus der FAQ ist nicht möglich.

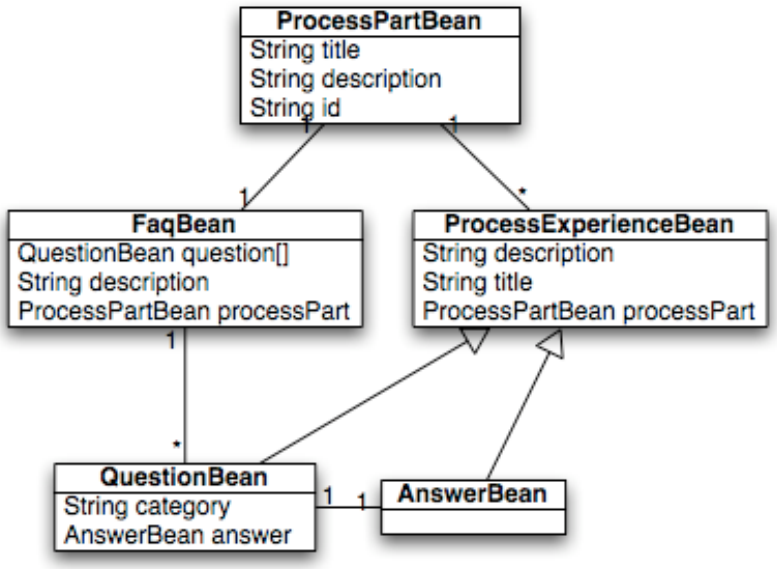

Abbildung 13: Datenstruktur für getFaq

Das Erstellen von Fragen und Antworten für die FAQ wird typischerweise von ausgezeichneten Anwendern übernommen und wird als administrative Aufgabe nicht als Service angeboten.

Mit `getSimplePage()` kann man weitere Freitexterfahrungen zu einem Prozessschritt abzurufen. Diese haben jedoch eher statischen Charakter und werden

wieder von ausgezeichneten Benutzern gepflegt. Typischerweise werden Best Practices zu einem Prozessschritt als SimplePage angeboten.

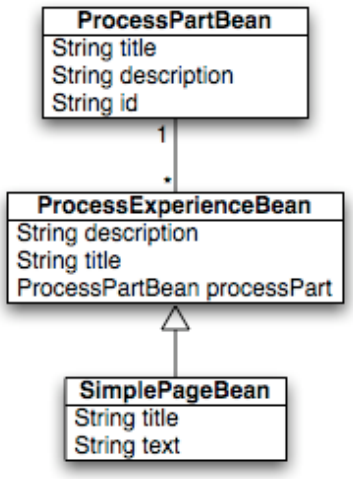

Abbildung 14: Datenstruktur für getSimplePage

4.4.1.2 ArtefactService

Dieser Service bietet Funktionalität, um Beispiele und Vorlagen anzufordern. Jede Prozesserfahrung kann mit einer Menge von Dateianhängen versehen werden. Dieser Service ermöglicht es, den tatsächlichen Inhalt für ein gegebenes `FileBean` einer ProcessExperience zu erhalten.

4.5 Service Requestor

Der Service Requestor soll zum einen das Nutzen der Web Services des Service Providers und zum anderen die direkte Integration der Funktionalität der Experience Base veranschaulichen.

4.5.1 Architektur

Der Service Requestor wird als MVC Applikation[13] ausgeführt. Der 3-Tier Architektur nicht unähnlich, wird das `Model` als Datenschicht von der Präsentation der Daten im `View` und den Operationen auf den Daten im `Controller` getrennt.

[13] http://java.sun.com/blueprints/patterns/MVC-detailed.html

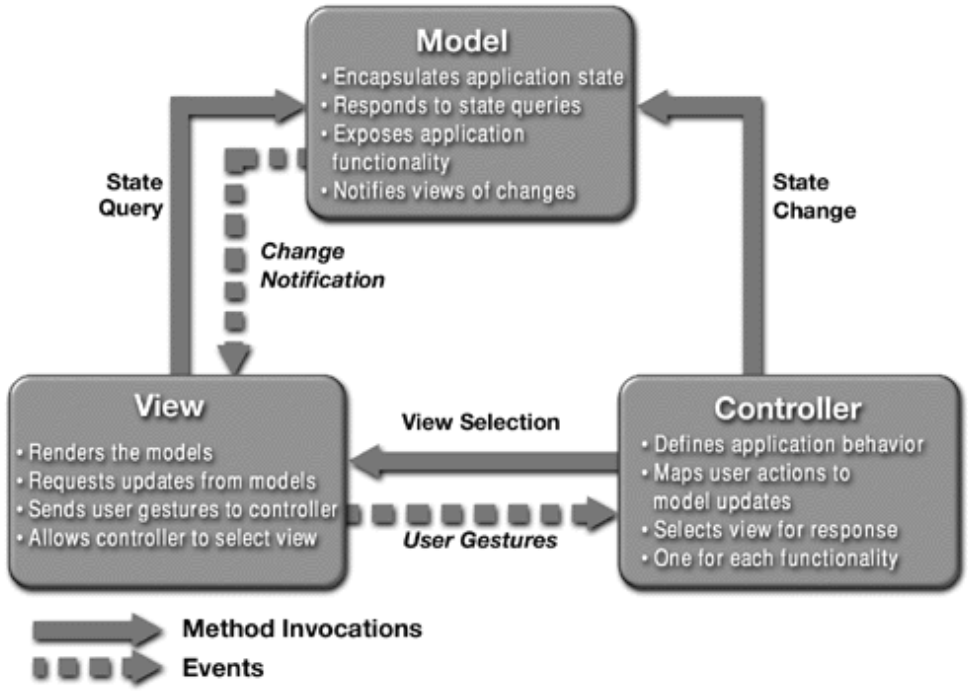

Abbildung 15: Die Model-View-Controller Architektur

Die verbreiteten Implementierungen grafischer Oberflächen für die J2EE Plattform vereinfachen die MVC Architektur ein wenig. Die Funktionalität des Controllers wird dabei typischerweise direkt in den Klassen des Views beschrieben.

In den Methoden des Controllers eines Service Requestor werden dann die Web Services des Service Providers genutzt.

4.5.2 Programmiermodell

Die formale Service Description in Form der WSDL zu einem Web Service kann von vielen Web Service Implementierungen genutzt werden, um Hilfsklassen für einen Service Requestor zu generieren.

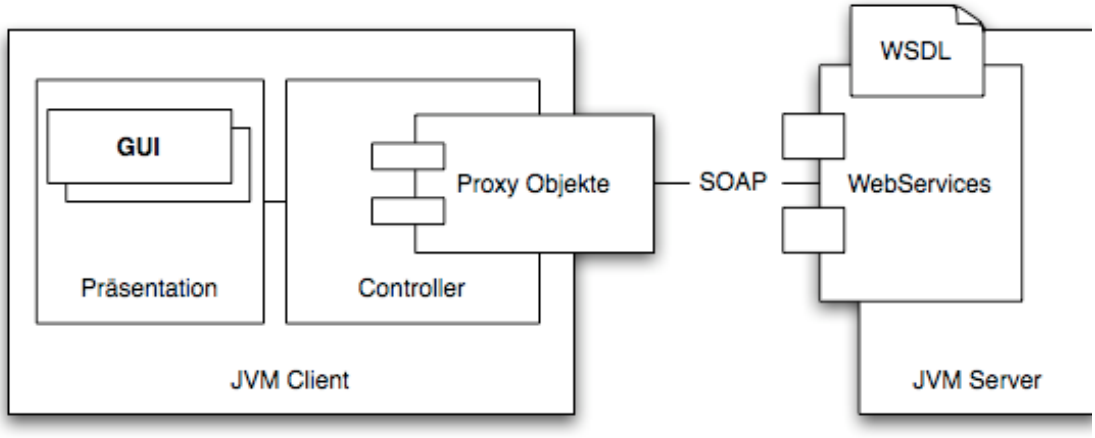

Abbildung 16: Integration der Web Service Stacks

So können Proxy Objekte erzeugt werden, welche die Methoden der Stateless Session Beans des Service Providers anbieten und Anfragen durch die Web Service Stacks

zum Application Server weiterleiten. Diese Proxy Objekte können nun in einer MVC Applikation aus dem Controller heraus genutzt werden.

4.5.3 Integration der Experience Base

Die einzelnen Masken der grafischen Oberfläche des Service Requestors werden Prozessschritten zugeordnet. Die Informationen aus der Experience Base zu einem Prozessschritt können schließlich in einer “ExperienceView” Komponente eingesehen werden.

Diese Komponente erlaubt das programmatische Bestimmen eines Prozessschrittes, für welchen Informationen angezeigt werden sollen. Der Einfachheit halber wird diese Komponente in einem eigenen Fenster organisiert, sodass auch bestehende Java Programme problemlos mit dieser Komponente nachgerüstet werden können. Selbstverständlich kann diese Komponente auch auf eine andere Art und Weise integriert werden.

Abbildung 17: Die Maske der Experience View

5. Realisierung

Für die konkrete Realisierung des Entwurfs müssen Entscheidungen getroffen werden, um die Konzepte des Entwurfes umzusetzen. Dabei gilt es neben einer Implementierung eines Web Service Stacks auch ein System zum Übersetzen und Einrichten der Anwendung zu wählen.

Für ein Deployment von Enterprise JavaBeans in einem Application Server sind neben dem übersetzten Quellcode weitere Artefakte nötig, welche Details für angrenzende Systeme festlegen und Eigenschaften des Deployments selber bestimmen. Die Generierung dieser Artefakte, das Übersetzen des Quellcodes und das Einspielen des fertigen Archivs in den Application Server werden typischerweise zur Übersetzungszeit von einem Build System erledigt.

5.1 Build System

Es haben sich mehrere, teilweise überschneidende, Lösungen für die notwendigen Aktivitäten zur Übersetzungszeit entwickelt. Wenn man auch ganz auf ein solches Build System verzichten könnte, so diktiert doch die Bequemlichkeit, möglichst viel zu automatisieren und generieren zu lassen.
Ein Build System sollte dabei mit so wenigen Informationen wie nötig auskommen, um ein Deployment zu erzeugen. Durch die reduzierte Redundanz ist eine Konsistenz der Daten gewährleistet und Änderungen können zentral vorgenommen werden.

5.1.2 Ant

Ant ist ein Build System des Apache Project.[14] Der Build Prozess wird in einer XML Datei als Menge abhängiger Teilziele beschrieben. So werden typischerweise zumindest das Säubern des Verzeichnisbaumes von Kompilat, das Übersetzen der Quelldateien und das Erstellen eines Archivs als Ziel definiert.
Ant kann über eigene Klassen erweitert werden, sodass es mittlerweile alle wesentlichen Aspekte eines J2EE Deployments abdeckt.

5.1.3 XDoclet

*"XDoclet is an open source code generation engine. It enables **Attribute-Oriented Programming** for java. In short, this means that you can add more significance to your code by adding meta data (attributes) to your java sources. This is done in special JavaDoc tags."* [15]

XDoclet wird hier als Erweiterung zu Ant genutzt. Es ist es ein Interpreter für Vorlagen, aus welchen in Abhängigkeit von Annotationen im Quellcode Artefakte erzeugt werden können. Entstanden ist XDoclet aus einer Erweiterung zu Javadoc, einem System zum Erstellen von Dokumentation zu Java Quellcode und die Funktionsweise ist dementsprechend ähnlich.

Notwendig wurde eine solche Erweiterung, da für ein Deployment der Enterprise JavaBeans auf der J2EE Plattform viele Metainformationen in Form von Deployment Deskriptoren und Artefakten für verwendete Systeme erzeugt werden müssen. Mit XDoclet können diese Informationen direkt im Quellcode in Form von Kommentaren

[14] http://ant.apache.org/
[15] http://xdoclet.sourceforge.net/xdoclet/index.html

gepflegt und dann während des Build Prozesses die notwendigen Artefakte automatisch erzeugt werden.

5.1.4 Maven

Maven ist ein weiteres Build System aus dem Apache Project, welches neben der Funktionalität von Ant noch weitergehende Konzepte wie die Abhängigkeiten zwischen verschiedenen EJBs und Bibliotheken auflöst.
Es versucht einen einheitlichen Build Prozess für die J2EE Plattform zu etablieren und das Problem mit vielen nur wenig unterschiedlichen Ant Build Prozessen zu lösen.

Viele der Vorteile von Maven kommen bei einem kleinen Projekt jedoch noch nicht zum Tragen. Wenn man auch Ant komplett mit Maven abbilden kann, so rechtfertigt ein eventueller Nutzen nicht das aufwendigere Einarbeiten, zumal Ant als etablierteres System einen Einstieg in das Beschreiben des Build Prozesses erleichtert.

5.2 Web Service Implementierungen

Eine Web Service Implementierung stellt zumindest einen Web Service Stack und optimalerweise noch eine Menge an Werkzeugen bereit, um mit Web Services zu arbeiten. Es gibt mittlerweile mehrere konkurrierende Web Service Implementierungen.

Die Implementierung der Experience Base ist bereits recht spezifisch für die JBoss 4.0.2 Plattform geschrieben, sodass die Wahl einer Web Service Implementierung bereits eingeschränkt ist. Die verwendete Implementierung sollte allerdings frei von potentiellen lizenzrechtlichen Problemen und möglichst kostenlos sein. Im Rahmen dieser Arbeit sollen drei Implementierungen kurz vorgestellt und gegeneinander abgewogen werden.

5.2.1 AXIS

AXIS vom Apache Projekt[16] ist wohl die am weitesten verbreitete Implementierung. Sie bietet eine ausgezeichnete Dokumentation und eine lebendige User Community. Ursprünglich von IBM entwickelt ist AXIS mittlerweile frei verfügbar. Die enthaltenen Werkzeuge bieten die Möglichkeit das Erstellen der Artefakte in den Build Prozess zu integrieren, sodass sich das Entwickeln von Web Services auf das Beschreiben der Interfaces und SessionBeans beschränkt.

AXIS, als eigentliche Web Service Implementierung, wird als eigenständige Applikation im Application Server eingerichtet. Stateless SessionBeans werden anschließend zusammen mit einer Beschreibung ihrer Java Schnittstelle für Web Services mit Hilfe eines “Web Service Deployment Descriptors” (WSDD) registriert. Zur Laufzeit werden Anfragen an den Dienst dann von AXIS an die entsprechende SessionBean delegiert.

5.2.2 JBossWS

JBossWS ist die aktuelle Implementierung eines Web Service Stacks für den JBoss Application Server und ersetzt das ältere auf AXIS basierende JBoss.NET. Der

[16] http://ws.apache.org/axis/

Vorteil liegt in einer guten Integration in die Laufzeitumgebung des JBoss Application Servers. Allerdings werden kaum Werkzeuge bereitgestellt um Artefakte der Web Services erzeugen zu lassen.
Bei dem Deployment eines EJB Archivs wird eine Konfigurationsdatei namens `webservices.xml` im Archiv gesucht. Diese kann genutzt werden, um eine Menge von enthaltenen Stateless SessionBeans mit ihren Java Schnittstellen anzugeben, welche als Web Service verfügbar sein soll.

5.2.3 JWSDP

Das "Java Web Services Developer Pack"[17] (JWSDP) ist die Referenzimplementierung für Web Services von SUN. Sie integriert sich hervorragend in die "Sun Java System Application Server Platform Edition", den Referenz Application Server für die J2EE Plattform von SUN. Eine Integration in den JBoss Application Server wird allerdings nicht unterstützt. Es bietet aber eine exzellente Möglichkeit Artefakte für Web Services während des Build Prozesses zu erstellen.

5.3 Service Provider

Die Web Service Schnittstelle der Experience Base wird von mehreren Stateless SessionBean (SLSB) implementiert. In den Methoden dieser Beans werden die SessionBeans aus der Experience Base genutzt und als Web Services neu zusammengestellt..

Das Anbieten der ausgezeichneten Methoden der SessionBeans als Web Service wird von der gewählten Web Services Implementierung realisiert. Für den Service Provider kommt als Web Service Implementierung ausschließlich JBossWS in Frage, da es die einzige aktuelle, unterstützte Implementierung für die JBoss 4.x Plattform ist.[18]

Wenn diese Implementierung auch alle relevanten Technologien für Web Services umsetzt, so fehlen doch Möglichkeiten, um zur Übersetzungszeit benötigte Artefakte zu generieren. Diese müssten aufwendig von Hand erstellt und gepflegt werden.
In der Praxis hat es sich daher etabliert, zur Übersetzungszeit `wscompile` aus der JWSDP Referenzimplementierung zur Generierung benötigter Artefakte zu nutzen.

5.3.1 Build Prozess

Der Build Prozess wird für Ant in einer Datei namens `build.xml` beschrieben. In dieser wird ein Projekt als Menge abhängiger Teilziele beschrieben.

[17] http://java.sun.com/webservices/jwsdp/
[18] http://wiki.jboss.org/wiki/Wiki.jsp?page=WSDeployJBossWS

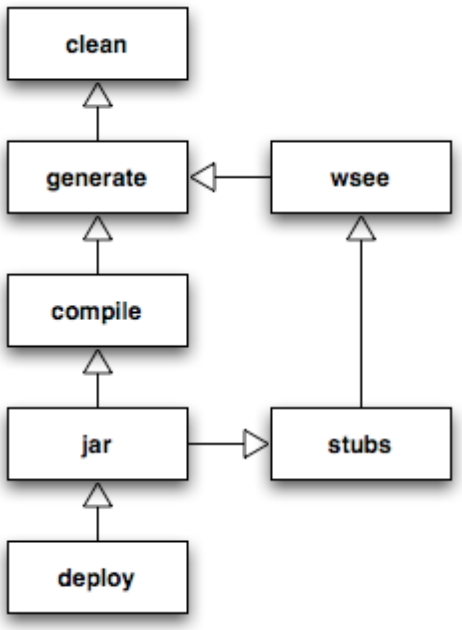

Abbildung 18: Abhängigkeiten der Teilziele für Ant

Einige Ziele sind selbsterklärend. So säubert `clean` den Verzeichnisbaum von Dateien, welche im Laufe des Build Prozesses erstellt werden, `compile` übersetzt den Java Quellcode, `jar` erstellt ein Archiv mit dem Kompilat und den Artefakten, welches dann mit `deploy` in den Application Server eingespielt wird.
In den anderen Zielen werden weitere Artefakte für ein Deployment auf der J2EE erzeugt.

5.3.2 Artefakte für die Plattform

Um die benötigten Artefakte für ein Deployment zu erstellen, stehen Erweiterungen des Ant Buid Systems aus den Web Service Implementierungen bereit. JBossWS als verwendete Implementation bietet hier leider noch keine eigenen Erweiterungen, so dass Elemente aus XDoclet mit Werkzeugen der Referenzimplementation JWSDP kombiniert werden müssen.

Im `generate` Ziel wird für die gegebenen SLSBs eine Schnittstellenbeschreibungen für ein Nutzen durch angrenzende Systeme im Application Server erzeugt. Damit stehen die SLSBs für andere Komponenten der J2EE Plattform zur Verfügung.

Um eine Nutzung als Web Services zu ermöglichen müssen noch weitere Artefakte generiert werden. Das `wsee` Ziel generiert eine Beschreibung der Abbildung der verwendeten Parameter der Schnittstelle auf XML Datentypen und die Service Description als WSDL Datei.

Das `stubs` Ziel erzeugt dann die nötigen Serialisierer und Verbindungen für die Web Services.

5.3.3 Weitere Services

Will man weitere Dienste der Experience Base als Web Servies anbieten, so muss man lediglich eine SLSB im Verzeichnis der vorhandenen SLSBs beschreiben. Der Name der SLSB muss auf *EJB.java enden, um vom Build System als Web Service

ausgeprägt zu werden und die SLSB muss noch in der Datei `wscompile-config` für das `stubs` Ziel mit ihren Schnittstellen eingetragen werden.

5.4 Service Requestor

Der Service Requestor kann losgelöst von der Realisierung des Service Providers betrachtet werden, einzige Abhängigkeit stellt die Service Description in Form der WSDL Datei dar.
Der Service Requestor als prozessunterstützende Applikation bildet prototypisch das Erstellen von Testplänen an. Durch die Integration der Funktionalität der Experience Base steht dann für die einzelnen Fenster die im Entwurf vorgestellte ExperienceView Komponente bereit.

5.4.1 Build Prozess

Der Build Prozess des Service Requestors wird ebenso wie der des Service Providers mit Ant und XDoclet abgebildet. Mit dem Wissen um die Adresse der WSDL Datei können Proxy Objekte erstellt werden, welche die Methoden der SLSB des Service Providers anbieten und in der Laufzeitumgebung des Service Requestors verfügbar machen.

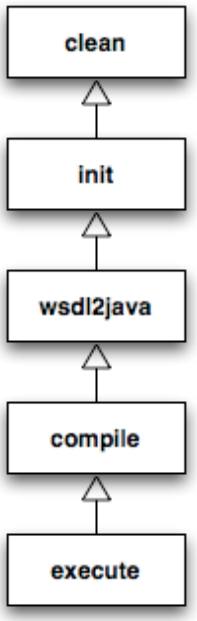

Abbildung 19: Abhängigkeiten im Build Prozess des Service Requestors

Um die Interoperabilität sicherzustellen, aber auch um eine alternative Web Service Implementierung vorzustellen kommt für den Service Provider AXIS zum Einsatz. Es bietet ebenso wie JWSDP die Möglichkeit, die Proxy Objekte zu generieren und wird im Ziel `wsdl2java` des Build Prozesses bemüht.
Darüber hinaus können automatisch Testfälle erzeugt werden, welche in den Build Prozess integriert werden können, um Aspekte wie Verfügbarkeit und Korrektheit der Service Description zu testen.

Am Ende des Build Prozesses steht dem Benutzer dann eine Applikation als Menge von einzelnen Masken zur Verfügung, für welche jeweils im Hilfsmenu Erfahrungen abgerufen werden können.

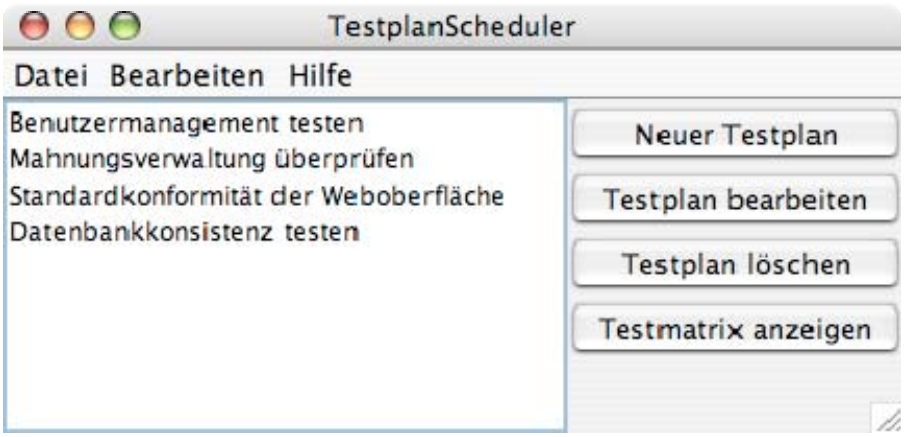

Abbildung 20: Übersichtsmaske des Service Requestors

6. Reflektion

Das Entwickeln für die J2EE Plattform unterscheidet sich wesentlich von herkömmlichen Ansätzen um Software Systeme zu entwickeln. Die recht aufwändige Beschreibung des Deployments und die abstrakten Konzepte erschweren einen Einstieg in die Plattform. Wenn es auch ausreichend Dokumentation gibt und viele Beispiele vorgestellt werden, erschließen sich weitergehende Konzepte wie die Abhängigkeit und Sichtbarkeit von Komponenten erst spät.

Die Anbindung der neuen SLSB an die bestehende Serviceschicht hat sich als schwierig gestaltet. Probleme des Deployments der eigentlichen Experience Base und Sichtbarkeitsproblemen der Enterprise JavaBeans im Application Server mit eher verwirrenden Fehlermeldungen haben die Entwicklung erschwert. Als Alternative zur Erstellung neuer SLSB über der bestehende Serviceschicht bietet sich das Verwenden der vorhandenen SLSB an. Wie auch im Entwurf dargestellt bieten diese allerdings oftmals keine geeignete Granularität, so dass diese Arbeit auch aufzeigen wollte, wie man die Funktion neu zusammenstellen kann.

Wenn die wesentlichen Technologien um Web Services auch verbreitet und gut verstanden sind, so ist es dennoch überaus wichtig, dass diese ausreichend in die Plattform integriert sind. Das ist für die Kombination aus AXIS und JBoss leider nicht gegeben und wurde erst recht spät im praktischen Teil dieser Arbeit festgestellt.

Das Anbieten von Funktionalität als Web Service macht nicht immer Sinn. Bei entsprechenden Integritätsbedingungen der Daten und verteiltem Deployment ist es aufwendig, die Daten konsistent zu halten. Abhängige Datenbankzugriffe aus mehreren Komponenten führen zu teuren Zugriffen und langsamem Laufzeitverhalten.

7. Zusammenfassung / Ausblick

Die Organisation der Experience Base als Menge von Web Services erlaubt eine starke Integration dieser in andere Softwaresysteme. Hierdurch entsteht für ein solches Softwaresystem zum einen ein verkürzter Feedback Cycle zwischen Anwendern und Entwicklern, Änderungswünsche und Erweiterungen können schneller und verlustfreier kommuniziert werden. Zum anderen bietet es den Anwendern eine Plattform, um Erfahrungen zu einzelnen Prozessschritten zu hinterlegen und verfügbar zu machen und die Abarbeitung des unterstützten Prozesses besser zu verstehen.

Softwaresysteme mit einer integrierten Experience Base garantieren damit eher eine Tauglichkeit, Prozesse auch in einer sich verändernden Domäne abzubilden. Sie erleichtern auch neuen Anwendern einen Einstieg durch vorhandenes, an einen Prozessschritt gekoppeltes Expertenwissen. Aufwendige Schulungen zur Bedienung eines solchen Softwaresystems können damit knapper ausfallen.

Die Technologien um Web Services als solche haben das Potential die Informationsverarbeitung vor allem zwischen Unternehmen neu zu organisieren. Klassische Korrespondenz könnte in Bereichen, die ausreichend formalisiert werden können, von prozessunterstützenden Softwaresystemen abgebildet und in die Ausführung eines konkreten Prozesses eingebunden werden. Sich verändernde Prozesse können durch eine neue Orchestrierung von Service Providern ohne Änderungen am Quellcode der Komponenten umgesetzt werden.

Die einzelnen Implementationen der Web Services für sich sind ausgereift, variieren aber stark bei den Aspekten der Integration in bestehende Softwaresysteme und bieten unterschiedliche Hilfswerkzeuge, um den Vorgang zu automatisieren. Es ist für den Einstieg wichtig, eine etablierte Implementation für die jeweilige Plattform zu wählen, um auf Beispiele und Erfahrungen anderer Benutzer zurückgreifen zu können.

8. Literaturverzeichnis

Sun Microsystems: The J2EE 1.4 Tutorial. (Auflage vom 5. Dezember 2005). Internet: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/ (Zugriff: 28. März 2005)

Sun Microsystems: Java Web Services Tutorial. (Auflage vom 14. Juni 2005). Internet: http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html (Zugriff: 28. März 2005)

V. R. Basili, G. Caldiera, H. D. Rombach: The Experience Factory, University Maryland, 1994

Heiko W. Rupp: JBoss – Server Handbuch für J2EE-Entwickler und Administratoren. (1. Auflage 2005: dpunkt.verlag)

K. Schneider, T. Schwinn: Maturing Experience Base Concepts at DaimlerChrysler (2001).

Tobias Buchloh: Erstellung eines Baukastens für Experience Bases. Internet: http://www.se.uni-hannover.de/documents/studthesis/MSc/Tobias_Buchloh-Erstellung_eines_Baukastens_fuer_Experience_Bases.pdf (Zugriff: 20.November 2005)

Heather Kreger, IBM Software Group: Web Services Conceptual Architecture. (Mai 2001). Internet: http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf (Zugriff: 27. November 2005)

Jim Knutson, Heather Kreger, IBM Software Group: Web Services for J2EE (Version 1.0 Final Release vom 21. September 2001)

Kurt Sinem: Ereignisgesteuerte Prozessketten. (2003). Internet: http://www.informatik.uni-ulm.de/dbis/01/lehre/ss03/gs/p_ablauf/EPK.pdf (Zugriff: 23. Februar 2005)

Daniel Lübke, Jorge Marx Gómez und Kurt Schneider: Serviceorientierte Architekturen und Prozessmanagment (GITO-Verlag 2005).