Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in verteilten Systemen 2009 in Kassel (WowKiVS 2009)

Praxisfallbeispiel: Modernisierung einer Mainframe-Anwendung durch eine verteilte SOA

Carsten Kleiner, Arne Koschel

11 Pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Praxisfallbeispiel: Modernisierung einer Mainframe-Anwendung durch eine verteilte SOA

Carsten Kleiner, Arne Koschel

Fachhochschule Hannover
Fakultät IV, Abteilung Informatik
Ricklinger Stadtweg 120
30459 Hannover
{ckleiner,akoschel}@acm.org

Abstract: Auch heute noch sind Legacy-Anwendungen auf Basis von Mainframe-DBMS wie Adabas und zugehörigen Programmiersprachen wie Natural häufig produktiv. Sie sind jedoch oft nur schlecht mit neuen Unternehmensanwendungen integriert. Der vorliegende Beitrag zeigt in einem Fallbeispiel die Modernisierung einer solchen Anwendung unter Einsatz von Web Services als Basis ihrer Integration in eine verteilte, Service-orientierte Architektur (SOA).

Keywords: SOA, legacy system, Natural, Web Service, legacy integration, Adabas

1 Einführung

1.1 Motivation

Legacy-Anwendungen auf Basis von Mainframe-DBMS wie Adabas [Date04] und zugehörigen Programmiersprachen der 4. Generation wie Natural der Software AG, sind heute noch häufig im produktiven Einsatz, oft jedoch nur schlecht mit neuen Unternehmensanwendungen integriert. Die Einbeziehung dieser existierenden IT-Werte ist somit eine wichtige Aufgabe, die auch bereits seit etlichen Jahren die IT-Abteilungen von Unternehmen beschäftigt. Zunächst wurde üblicherweise eine punktuelle Anbindung der Altanwendung an neuere Software-Komponenten durch die sogenannte Enterprise Application Integration (EAI) verwendet (vgl. [Kel02], [CHKT05]).
Service-orientierte Architekturen (SOA) sind heutzutage ein Mittel der Wahl für eine solche Integration (vgl. [KBS05], [ST07]). Während damit die prinzipielle Vorgehensweise für die Integration festgelegt ist, gilt es in der Praxis doch noch zahlreiche konkrete Herausforderungen zu meistern, bis die Legacy-Integration lauffähig ist. Dabei handelt es sich nicht nur um rein technologische sondern auch um konzeptionelle Probleme. Der vorliegende Beitrag erhebt daher nicht den Anspruch in diesem Bereich Grundlagenforschung darzustellen, sondern zeigt vielmehr anhand eines Praxisfallbeispiels aus einer Forschungs- und Lehre-Kooperation (studentisches Projekt) mit der Software AG, wie diese Herausforderungen am Beispiel gelöst wurden.

1.2 Verwandte Arbeiten

Situationen, in denen eine existierende Mainframe-Anwendung zu modernisieren ist, ergeben sich meist eher in der betrieblichen Praxis in einem konkreten Anwendungsfall. Seltener

werden sie eher konzeptionell im akademischen Umfeld betrachtet. Nichtsdestotrotz gibt es einige wenige wissenschaftliche Veröffentlichungen, die das Vorgehen bei der Integration einer existierenden Mainframe-Anwendung in eine SOA untersuchen. Während sich [CFFT08] im Wesentlichen auf interaktive Funktionalitäten beschränkt und eine Wrapper-Variante ähnlich der in diesem Artikel verwendeten einsetzt, schlägt [Eng08] einen Bottom-Up-Ansatz vor. Dieser hat zwar den Vorteil, dass er nicht nur interaktive Komponenten betrachtet, ist aber aus Sicht der Prozessmodellierung eher ungünstig, da sich möglicherweise Prozessoptimierungen nicht erschließen.
Das Tutorial [Smi07] beschreibt Wege wie allgemein eine SOA im Unternehmen unter Berücksichtigung von Legacy-Anwendungen eingeführt werden kann. Es werden umfassende Ansätze vorgestellt, die in der in diesem Artikel beschriebenen Fallstudie aufgrund des fehlenden Umfelds sowie des mehr technischen Integrationsfokus nicht eingesetzt wurden. In [LMSO05] wird eine Technik vorgestellt, wie eine bestehende Legacy-Anwendung analysiert werden kann, um festzustellen, ob und ggfs. welche der Funktionalitäten sich für eine Integration in eine SOA eignen. Diese Technik musste im Fallbeispiel nicht verwendet werden, da hier eine vollständige Übernahme aller Funktionalitäten der Altanwendung von vornherein festgelegt war.
In [EAK06] wird schließlich eine Kategorisierung von Methoden zur Integration von Legacy-Systemen in eine SOA vorgestellt. Die verschiedenen Realisierungsoptionen, die in Kapitel 4.1 diskutiert werden, finden sich direkt auch in dieser Klassifizierung wieder. Aufgrund der einfachen Beschaffenheit in der Fallstudie konnten einige der dort vorgestellten Varianten („Direct Data Access“) direkt ausgeschlossen werden. Andere komplexere Verfahren (z.B. „Reengineering“) sind nicht erforderlich, da das schließlich verwendete Wrapping nicht auf Basis der kompletten Prozesse durchgeführt wird.

2 Die existierende Mainframe-Anwendung – SAG-Tours

SAG-Tours [SAG07] ist eine Natural-Anwendung in einer klassischen 1-Tier, Terminal-basierten Mainframe-Architektur (siehe Abb. 1). Sie erlaubt es fiktive Segelreisen zu buchen. Technisch kommen Terminal-Emulationen zum Einsatz, die via Telnet-Protokolle direkt mit einer Unix-Variation von Natural kommunizieren. Die Natural-Anwendung greift wiederum auf eine Adabas-Datenbank zu. Im Rahmen der Projektentwicklung kam auch von Adabas eine Unix-Variante zum Einsatz. Da Adabas und Natural aufgrund ihres Alters nicht mehr als allgemein bekannt angesehen werden können, werden sie im Folgenden exemplarisch etwas näher erläutert.

Abb. 1: 1-Tier Systemarchitektur der existierenden Mainframe-Altanwendung

Adabas

Adabas ist ursprünglich ein hoch-performantes Mainframe-DBMS, das intern auf sogenannten invertierten Listen basiert. Es wurde ca. 1971 erstmals produktiv installiert und seit dem

kontinuierlich weiterentwickelt. Konzeptuell basiert es auf einem NF²-ähnlichen Datenbankmodell (non first normal form). Dies bedeutet, dass Adabas Daten nicht nur als atomare Werte in der ersten Normalform zulässt, sondern zusätzlich sogenannte multiple Felder und Periodengruppen erlaubt. Adabas wird in der Regel in Verbindung mit Natural verwendet, bietet aber im Prinzip auch Schnittstellen für einige andere Programmiersprachen.

Exemplarisch wird an einer Anfrage erläutert, wie Adabas intern arbeitet. Es wird eine Datenbankabfrage ausgeführt, die alle Reisen mit Starthafen „Curacao“ ermittelt:

```
FIND CRUISE WITH START HARBOR= 'CURACAO'
```

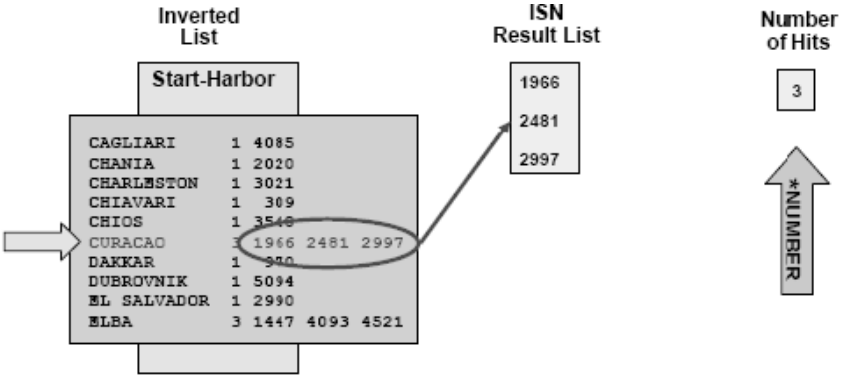

Abb. 2: Adabas Arbeitsweise intern – Phase 1

In Phase 1 werden die vom FIND-Statement angeforderten Datensätze anhand ihrer internen Satznummer (ISN) im Temporärbereich der Adabas-DB gespeichert.

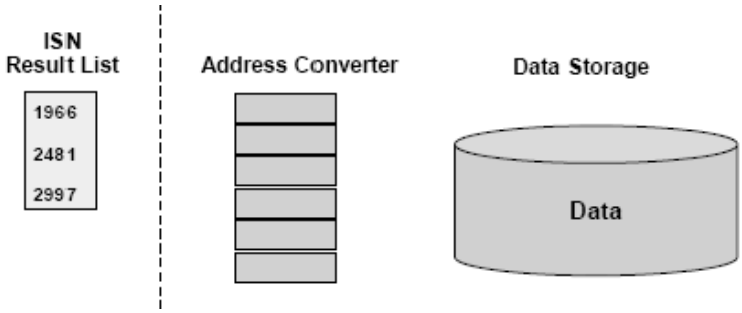

Abb. 3: Adabas Arbeitsweise intern – Phase 2

In Phase 2 wird eine Schleife durchlaufen, welche die Datensätze anhand der gespeicherten ISN ausliest und dem Programm zureicht.

Natural

Natural ist eine mittlerweile über 30 Jahre alte prozedurale Programmiersprache der 4. Generation, d.h. sie beinhaltet ein integriertes Datenbank-Zugriffskonzept. Ursprünglich wurde Natural für die Mainframerechner von IBM und Siemens entwickelt, steht aber mittlerweile auf sehr vielen Plattformen zur Verfügung. Der Sourcecode kann relativ problemlos zwischen einzelnen Plattformen ausgetauscht werden. Abhängigkeiten existieren jedoch teilweise zum verwendeten DBMS-Typ, z.B. Einschränkungen bei Verwendung von relationalen DBMS vs. dem Adabas-DBMS.
Ein einfaches Code-Beispiel zeigt nachfolgend, wie mittels Natural auf eine Adabas-Datenbank zugegriffen werden kann, die Artikel verwaltet. Im DEFINE DATA-Abschnitt wird zunächst eine lokale Datenstruktur zur Aufnahme der Artikeldaten als Sicht (VIEW) auf die Artikel-Tabelle deklariert. In der nachfolgenden READ-Anweisung wird sodann in einer impliziten Schleife die Artikel-Tabelle ausgelesen. Schön erkennbar ist der 4GL-konform verfügbare, inhärente Tabellendatentyp.

```
DEFINE DATA
LOCAL
      1 ARTIKELANSICHT VIEW OF ARTIKEL
           2 NAME (A30)
           2 ANZAHL (N7)
ENDDEFINE

READ ARTIKELANSICHT BY NAME
      DISPLAY NAME 5X ANZAHL
ENDREAD
END
```

Bei Ausführung erzeugt das Programm eine Liste aller Artikel der Tabelle (ARTIKEL) mit ihrem Artikelnamen (NAME) und deren Anzahl (ANZAHL) aus.

```
NAME                            ANZAHL
-----------------------------------
LEGO Pneumatik Kranwagen        10
LEGO Schneemobil                50
```

3 Geschäftsprozessmodell

Zunächst wurden aufgrund der gewählten Kombination aus Top-Down und Bottom-Up-Ansatz (vgl. auch Abschnitt 4.1) die Soll-Geschäftsprozesse identifiziert. Danach wurden basierend auf den vorhandenen Funktionalitäten der Altanwendung einzelne Dienste aus dem Bereich der Business Services[1] identifiziert. Diese Dienste entstanden dabei direkt aus den zugehörigen Natural-Programmen der Altanwendung bzw. wurden in einzelnen Fällen zusätzlich in Natural

[1] Aufgrund der eher einfachen vorliegenden Altanwendung in der Fallstudie wurden nur Dienste auf zwei Ebenen, nämlich Business Process Services sowie Business Services verwendet. Basic Services wurden nicht identifiziert. Das Vorgehen könnte aber auch analog auf Basic Services erweitert werden.

implementiert, um eine einheitliche Realisierung aller Dienste der Altanwendung zu erreichen. So gab es in der Altanwendung ein Natural-Programm `DRINF-N0`, das alle verfügbaren Reisen zu einem Startdatum und Starthafen anzeigt; dieses wurde im Dienst **SRVC_FindCruises** gekapselt und so verfügbar gemacht.
Schließlich wurden die auf diese Weise gewonnenen Dienste zu den zuvor definierten Geschäftsprozessen kombiniert. So zeigt Abb. 4 einen Ausschnitt aus dem Geschäftsprozessmodell, in dem der Geschäftsprozess **BP_AddContract** zum Hinzufügen einer Reisebuchung aus Diensten wie **SRVC_FindCruises** und **SRVC_FindYachtWithSkipper** zusammengesetzt wird, die direkt den Funktionalitäten der Mainframe-Anwendung entsprechen.

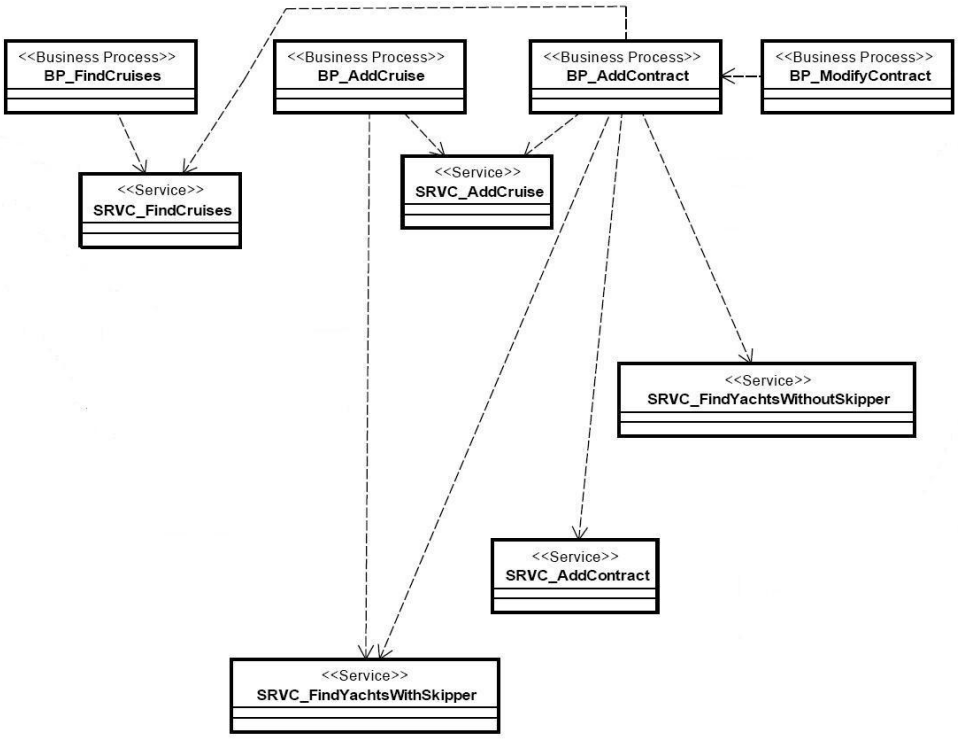

Abb. 4: Auszug aus dem Geschäftsprozessmodell der Fallstudie

Bei der Modellierung wurden die elementaren Dienste der Business-Schicht so kombiniert, das sie teilweise in verschiedenen Prozessen verwendet werden können. So besteht der Prozess (Abb. 4) des Hinzufügens einer Reisebuchung (**BP_AddContract**) aus den Bestandteilen des Suchens oder Anlegens einer Fahrtroute, des Findens der dafür zur Verfügung stehenden Schiffe sowie des Anlegens des eigentlichen Buchungssatzes. Wie in Abb. 4 angedeutet, können einige dieser elementaren Funktionen in anderen Prozessen ebenfalls verwendet werden.

4 Integrationsarchitektur

4.1 Konzeption

Prinzipiell bieten sich bei der Integration einer Mainframe-Anwendung mehrere Vorgehensweisen an (vgl. auch Kapitel 1.2). In unserer Fallstudie wird letztendlich der Weg des „Business Logic Wrapping“ aus der Familie der invasiven Methoden (Klassifizierung nach [EAK06]) gewählt.
Der Ansatz einer Neuimplementierung (aus der Familie der invasiven Methoden nach [EAK06]) der vorhandenen Funktionalität ([SYON08]) in einer Programmiersprache, die sich besser für die Verwendung in einer SOA eignet (z.B. Java oder .NET), scheidet in vielen Fällen schon aufgrund des erforderlichen hohen Aufwands aus. Im vorliegenden Fall sollte daher ein Ansatz verwendet werden, der einen Weiterbetrieb der bestehenden Anwendung vorsieht. Außerdem sollte in dieser Fallstudie unter Beweis gestellt werden, dass eine Integration unter Beibehaltung der Altanwendung tatsächlich nur einen überschaubaren Aufwand erfordert.
Der Kooperationspartner Software AG bietet auch Werkzeuge an, mit denen man mithilfe von Screenscraping ganze Geschäftsprozesse aus der Altanwendung extrahieren und als Dienste (in diesem Fall meist Web Services) verfügbar machen kann. Diese Vorgehensweise wurde hier ebenfalls nicht eingesetzt, da sich auf diese Weise nur sehr wenige, grob granulare Dienste ergeben. Somit erhält man nur Dienste auf der Ebene der Geschäftsprozesse (vgl. [KBS05]) und keine Dienste auf den darunterliegenden Ebenen (Business Services, Basic Services). Mit diesem Vorgehen erreicht man zwar eine vermutlich noch schnellere Integration der Altanwendung; allerdings ist so keine erhöhte Flexibilität der Prozesse durch variable Kombination der Dienste der unteren Ebenen zu erreichen.

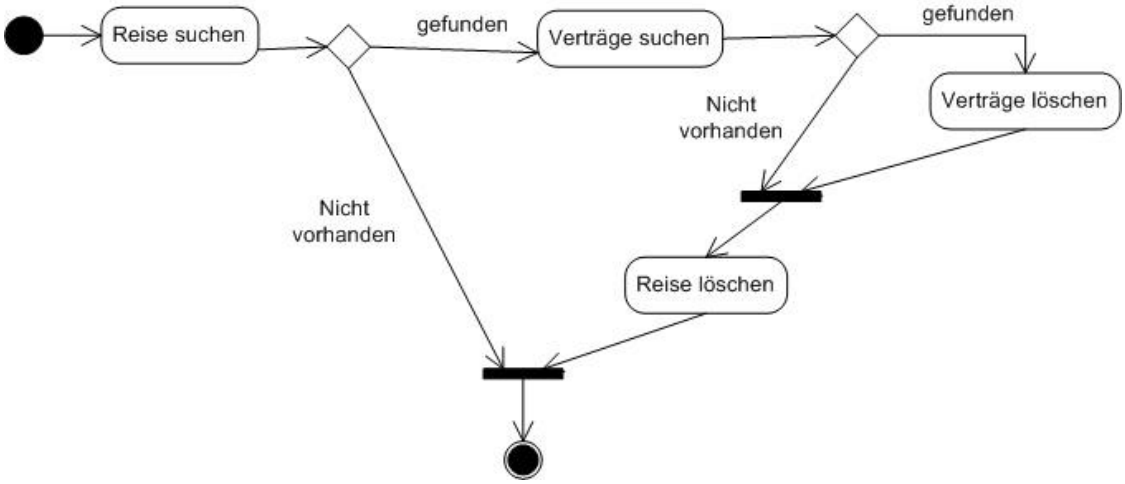

Abb. 5: Modellierung des Geschäftsprozesses zum Löschen einer Reise

Die Kapselung der Funktionalitäten der Altanwendung in Business Services findet sich auch in Abb.5 wieder, in der exemplarisch der modellierte Ablauf des Geschäftsprozesses zum Löschen einer Reise dargestellt ist. Man kann hier deutlich die elementaren Aufgaben erkennen (Business Objekt suchen bzw. löschen), die früher in einem Natural-Programm

implementiert worden sind. Diese Programme werden nunmehr in Dienste verpackt, so dass der Ablauf des Geschäftsprozesses alleine durch adäquate Kombination der Business Services erreicht werden kann. Im Unterschied zu realen Szenarien, in denen diese Kombination der Dienste durch automatische Generierung aus den modellierten Diagrammen automatisiert vorgenommen werden würde, wurde dies im Projekt manuell erledigt. Dies hatte einerseits den Grund, dass hier nur wenige Prozesse statisch modelliert wurden, so dass der Aufwand überschaubar war. Andererseits sollte der Overhead für die Einarbeitung in ein weiteres umfassendes Werkzeug eingespart werden. In der Praxis ist eine Verwendung solcher Werkzeuge selbst bei mittelgroßen Projekten jedoch zu empfehlen.

4.2 Architektur

In Abb. 6 ist die entwickelte Architektur für die Integration der Legacy-Systeme dargestellt. Die Integration findet in der Anwendungsschicht statt. Die existierenden Legacy-Komponenten Adabas und Natural bleiben unverändert und werden mittels eines RPC-Servers genutzt. Dieser RPC-Server stellt also einen Teil des in Abschnitt 4.1 beschriebenen Wrappers um die Natural-Funktionalitäten dar und wird von der für die Integration erforderlichen Komponente EntireX-Broker angesprochen. Die interne Kommunikation dieser beiden Komponenten wird hierbei mittels XML Mapping (XMM) durchgeführt.
Auch der EntireX-Broker gehört technisch gesehen noch zum Wrapper, denn der EntireX-Broker wiederum kann von außen direkt über Web Services auf Basis von SOAP angesprochen werden. Daher bilden diese Komponenten zusammen den Web Service Wrapper.

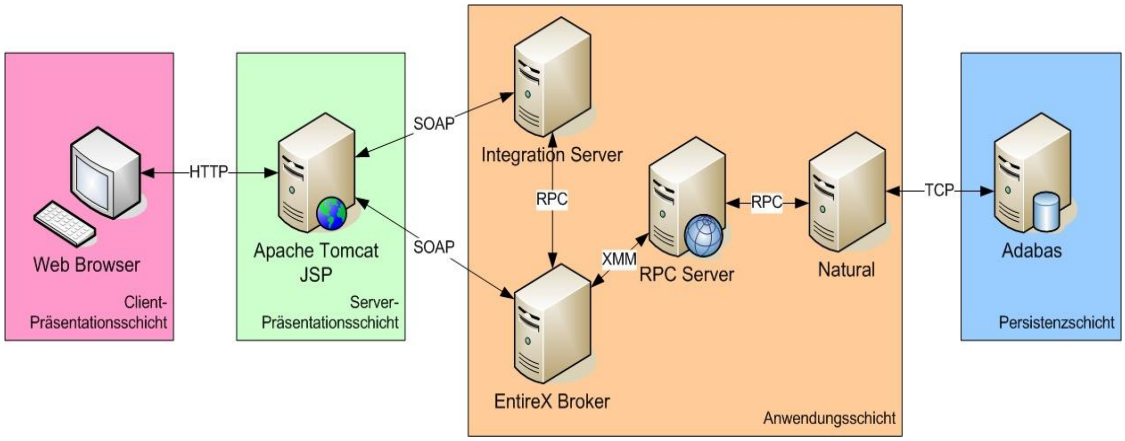

Abb. 6: Software-Architektur zur Integration der existierenden Anwendung

Alternativ sprechen die Web Services die weitere Komponente „Integration Server“ an. In diesem Falle kontaktiert der Integration Server über RPC den zuvor erwähnten Broker und gehört ebenfalls zur Wrapper-Schicht. Im konkreten Fallbeispiel wurden die Web Services der Anwendungsschicht direkt aus JSP-Seiten der Server-Präsentationsschicht angesprochen, um ein einfaches Frontend zu erhalten. Die konkret definierten Geschäftsprozesse werden also hier direkt in der Präsentationsschicht implementiert. In realistischen Szenarien ist natürlich eher ein externes Ansprechen der entsprechenden Web Services durch eine Prozessmaschine zu

erwarten, in der die Geschäftsprozesse in einer geeigneten Sprache (z.B. BPEL oder BPMN) definiert werden können. Auf diese Weise sind die Legacy-Komponenten vollständig als Dienste gekapselt und können so in eine SOA-Gesamtlandschaft eingebettet werden.

4.3 Implementierung

Anhand von Ablauferklärungen und kleinen Code-Beispielen aus der integrierten (neuen) Anwendung wird die Umsetzung der Integrationsarchitektur näher erläutert.
„Bottom Up“ betrachtet, beginnt die Entwicklung auf Basis der bestehenden Adabas-Datenbank. Hierzu bietet die SAG eine integrierte Entwicklungsumgebung, die den Import von Adabas-Datenstrukturen (im folgenden Beispiel: `YACHT-V`) in entsprechende „`DEFINE DATA`“-Blöcke für Natural-Programme (genauer: Natural-Sub-Programme) leicht ermöglicht. Für die Sub- Programme werden entsprechende Ein-/Ausgabe-Parameter festgelegt, die sodann im eigenen Code genutzt werden können. In nachfolgenden Code-Stück werden alle Yachten mit dem Yacht-Branch „`B`“ gefunden. Die Daten werden zuerst in die Ansicht `YACHT-VIEW` gelesen und danach werden die Suchergebnisse in der Struktur `YACHT-INFO` gespeichert. Die gefüllte Struktur `YACHT-INFO` wird vom Sub-Programm zurückgegeben.

```
DEFINE DATA
PARAMETER USING YACHT-PD
LOCAL USING YACHT-V
   1 ZAEHLER (N5.0) INIT <0>
   1 GERNZE (N5.0) INIT <0>
END-DEFINE

FIND ALL YACHT-VIEW WITH YACHT-VIEW.YACHT-BRANCH = "B"
     . . .
MOVE YACHT-VIEW.YACHT-NAME TO YACHT-INFO.YACHT-NAME(ZAEHLER)
     . . .
```

Die Natural-Sub-Programme werden im nächsten Schritt im Entire-X-Broker als mittels RPC aufrufbare Programme registriert, für die eine Schnittstellenbeschreibung zur Verfügung steht. Über Werkzeuge der Entwicklungsumgebung, kann aus dieser Schnittstellenbeschreibung eine Web Services-Beschreibung (WSDL) generiert werden.
Danach erfolgt eine normale Client-Entwicklung für Web Services. In unserem Beispiel werden entsprechende Axis-Werkzeuge zur Generierung von Java-Aufruf- und Holder-Klassen aus obiger WSDL eingesetzt. Innerhalb von JSPs bzw. Servlets, werden diese Java-Klassen von der neuen Web-Anwendung genutzt.

5 Fazit und Ausblick

Die Integration von Legacy-Anwendungen in dienstorientierte Architekturen ist inzwischen weder unmöglich noch sehr aufwändig. Am Fallbeispiel von Legacy-Anwendungen auf Basis von Adabas und Natural konnte in diesem Projekt eine Integration in die Web Service Welt erreicht werden. Es sind zwar gegenüber der bestehenden Architektur (vgl. Abb. 1) weitere Komponenten in die Architektur aufzunehmen (vgl. Abb. 6). Die Tatsache, dass dieses aber im Rahmen eines Projekts mit Bachelor-Studierenden ohne größere Probleme möglich war, zeigt, dass eine solche Integration keine unverhältnismäßigen hohen Kenntnisse bzw. Aufwand mehr

erfordert. Außerdem wurde aufgezeigt, welche Vorgehensweise bei der Integration der Altanwendung verfolgt wurde. Diese Methodik sollte auch bei anderen ähnlich strukturierten Altsystemen anwendbar sein. Das gleiche gilt für die technische Realisierung der Integration.
Ob eine solche Integration in einem heterogeneren Systemumfeld möglich wäre (z.B. ohne die vom selben Hersteller angebotenen Integrationskomponenten einzusetzen) bzw. welchen Aufwand sie ansonsten erfordert hätte, könnte in einem Folgeprojekt evaluiert werden. Ebenso könnten Alternativen zum kombinierten Top-Down-/Bottom-Up-Vorgehen im Detail angewendet werden. Ein Vergleich des dann erhaltenen Zielsystems mit dem in Kapitel 4 beschriebenen könnte Aufschlüsse über die Eignung geben.

6 Danksagung

Wir möchten uns bei den Studierenden des SAG-Tours Projektteams für Ihre produktive und motivierte Mitarbeit in diesem Projekt bedanken.

7 Referenzen

[ATZ08] Francesca Arcelli, Christian Tosi, and Marco Zanoni. Can design pattern detection be useful for legacy system migration towards SOA? In SDSOA'08: Proceedings of the 2nd international workshop on Systems development in SOA environments, pages 63–68, New York, NY, USA, 2008. ACM.

[CFFT08] Gerardo Canfora, Anna Fasolino, Gianni Frattolillo, P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. Journal of Systems and Software, 81(4):463–480, 2008.

[CHKT05] Stefan Conrad, Wilhelm Hasselbring, Arne Koschel, Roland Tritsch. Enterprise Application Integration: Grundlagen - Konzepte - Entwurfsmuster – Praxisbeispiele. Spektrum Akademischer Verlag, 2005

[Date04] C.J. Date, An Introduction to Database Systems, 8th ed., Pearson, 2004

[DEF+08] Jürgen Dunkel, Andreas Eberhart, Stefan Fischer, Carsten Kleiner, Arne Koschel: Systemarchitekturen für Verteilte Anwendungen, Hanser-Verlag, 2008.

[DJA07] Asit Dan, Robert Johnson, and Ali Arsanjani. Information as a service: Modeling and realization. In SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments, page 2,Washington, DC, USA, 2007. IEEE Computer Society.

[EAK06] Abdelkarim Erradi, Sriram Anand, and Naveen N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In IEEE SCC, pages 257–260. IEEE Computer Society, 2006.

[Eng08] Sandra Englet. Wiederverwendung von Legacy Systemen durch einen bottom up Ansatz bei der Entwicklung einer SOA. In Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, and Christian Scheideler, editors, GI Jahrestagung (1), volume 133 of LNI, pages 96–100. GI, 2008.

[KBS05] Dirk Krafzig, Karl Banke, and Dirk Slama. Enterprise SOA: Service Oriented Architecture Best Practices. Prentice Hall International, 2005.

[Kel02] Wolfgang Keller. Enterprise Application Integration. Erfahrungen aus der Praxis. Dpunkt-Verlag, 2002.

[KK09] Carsten Kleiner, Arne Koschel: Legacy vs. Cutting Edge Technology in Capstone Projects: What works better? Erscheint in: Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE09), 2009.

[LMS06] Grace Lewis, Edwin Morris, and Dennis Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In CSMR ’06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 15–23,Washington, DC, USA, 2006. IEEE Computer Society.

[LMSO05] Grace Lewis, Edwin Morris, Dennis Smith, and Liam O’Brien. Serviceoriented migration and reuse technique (smart). In STEP ’05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pages 222–229, Washington, DC, USA, 2005. IEEE Computer Society.

[PH07] Mike Papazoglou and Willem Heuvel. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal, 16(3):389–415, 2007.

[PZL08] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big web services: making the right architectural decision. In WWW’08: Proceeding of the 17th international conference onWorldWide Web, pages 805–814, New York, NY, USA, 2008. ACM.

[Smi07] Dennis Smith. Migration of legacy assets to service-oriented architecture environments. In ICSE COMPANION ’07: Companion to the proceedings of the 29th International Conference on Software Engineering, pages 174-175,Washington, DC, USA, 2007. IEEE Computer Society.

[Soft07] Software AG, SAGtours: SOA Integration Project, Application Document, Darmstadt, 2007.

[ST07] Gernot Starke, Stefan Tilkov (Hrsg.). SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen. Dpunkt-Verlag, 2007.

[SYON08] Toshio Suganuma, Toshiaki Yasue, Tamiya Onodera, and Toshio Nakatani. Performance pitfalls in large-scale java applications translated from cobol. In OOPSLA Companion ’08: Companion to the 23rd ACM SIGPLAN conference on Object oriented programming systems languages and applications, pages 685–696, New York, NY, USA, 2008. ACM.