Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Frank Hardenack

Webservice-basierte Kommunikation von Java-Anwendungen mit SAP

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Frank Hardenack

Webservice-basierte Kommunikation von Java-Anwendungen mit SAP

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bernd Kahlbrandt
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Abgegeben am 7. April 2009

Frank Hardenack

Thema der Bachelorarbeit
Webservice-basierte Kommunikation von Java-Anwendungen mit SAP
Stichworte
Webservice Java SAP Kommunikation RFC SAP-BP Business Partner
Kurzzusammenfassung
In dieser Bachelorarbeit wird ein Java-Frontend entwickelt, das über *Webservices* auf ein bestehendes SAP-System zugreift und die Kontaktdaten von Geschäftspartnern (Modul: *SAP-BP*) anzeigt sowie das Anlegen, Verändern und Löschen von Kontaktdaten ermöglicht. Die Firma erhofft sich durch die Umsetzung dieses Projekts einen Ansatz, um in Zukunft flexibler mit Java-Frontends auf SAP-Systeme zugreifen zu können. Ein Vergleich zur proprietären *RFC*-Technologie von SAP ist auch Teil dieser Arbeit. Um einen Vergleich vornehmen zu können, muss das Frontend ebenfalls für die *RFC*-Variante entwickelt werden. Für einen Vergleich ist es notwendig, geeignete Bewertungskriterien zu finden und zu nutzen.

Frank Hardenack

Title of the paper
Webservice-based Communication between Java-Applications and SAP
Keywords
Webservice Java SAP Communication RFC SAP-BP Business Partner
Abstract
A Java-frontend for *Webservice*-based communication with a SAP system shall be developed in this bachelor thesis. The frontend shall access contact information of business partners (module: *SAP-BP*) as well as allowing to create, change or delete entries for business partners. The company expects a basic approach for developing Java-frontends for SAP applications in future from realising this project. A comparison to the proprietary *RFC* technology from SAP is also part of this bachelor thesis. To draw a comparison it is necessary to develop the frontend for the *RFC*-variant and to find and use suitable assessment criteria.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Das Problem 1
1.2 Ein Lösungsansatz 1

2 Das Unternehmen C1WPS **2**
2.1 Wahl des Unternehmens 2
2.2 C1WPS 2

3 Aufgabenstellung **3**
3.1 Ausgangslage 3
3.2 Darstellung des bestehenden Systems 3
3.3 Lösungsansatz 3
3.4 Was spricht für diese Lösung 4

4 SAP **5**
4.1 Das Modulsystem von SAP 5
4.2 Eigene Erweiterungen 5
4.3 SAP BP 5
4.4 ABAP und ABAP-O 6
4.4.1 ABAP 6
4.4.2 ABAP-O 6

5 Webservice zwischen Java und SAP - Ein Beispiel **7**
5.1 Webservice-Funktionalität von SAP 7
5.2 Anlegen der Berechnungsfunktion 7
5.3 Erzeugung und Aktivierung des Webservices 7
5.3.1 Problem bei der Namensgebung 8
5.4 Konsumieren des Webservices 8
5.4.1 Probleme mit der Pfadangabe des WSDL-Dokuments 8
5.5 Ergebnis und Ausblick 9

6 Planung der Webservice-Lösung **11**
6.1 Frontend 11
6.2 Webservice 11
6.3 SAP-System 11
6.3.1 Relevante Funktionsbausteine in SAP BP 12

7 Entwurf der Webservice-Lösung **16**
7.1 Frontend 16
7.1.1 Softwaretechnische Struktur 16
7.1.2 Design 17
7.1.3 Anwendungsfälle 19
7.1.4 Selektion der angezeigten Daten 19
7.2 Webservice 20

7.2.1 Das Commit-Problem . . . 21
7.2.2 Die Lösung: Wrapper . . . 21
7.2.3 Das Problem mit den Live-Referenzen . . . 21
7.2.4 Die Lösung: Eigene Datenelemente . . . 21
7.2.5 Die einzelnen Services . . . 22
7.3 SAP-System . . . 22
7.3.1 Erweiterung um eigenen Webservice . . . 22
7.3.2 Verwendete Datentypen (SAP) . . . 23

8 Ausführung des Entwurfs der Webservice-Lösung 24
8.1 Frontend . . . 24
8.1.1 Hilfsklassen . . . 24
8.1.2 Eigene Datentypen . . . 25
8.1.3 Interfaces . . . 25
8.1.4 Swing und Threadsicherheit . . . 25
8.1.5 Dialogklassen . . . 25
8.1.6 Verwendung der Progress-Bar . . . 26
8.1.7 Fehlerbehandlung . . . 26
8.1.8 Export als .jar . . . 26
8.2 Webservice . . . 26
8.2.1 Verschachtelte Packages und wsimport . . . 26
8.2.2 Live-Referenzen und lesender Zugriff . . . 27
8.3 SAP-System . . . 27
8.3.1 Wrapper anlegen . . . 27
8.3.2 Eigene Datenelemente und Tabellentypen anlegen . . . 28

9 RFC in SAP 30
9.1 Varianten des RFC . . . 30
9.2 RFC mit einer externen Java-Anwendung . . . 31

10 RFC zwischen Java und SAP - Ein Beispiel 32
10.1 Anlegen eines RFC-fähigen Funktionsbausteins . . . 32
10.2 Nutzen des RFC in einer Java-Anwendung . . . 32
10.2.1 Anlegen eines Connection Pools und Erstellen der Verbindungen . . . 32
10.2.2 Aufrufen des entfernten Funktionsbausteins . . . 33
10.3 Ergebnis und Ausblick . . . 33

11 Planung der RFC-Lösung 34
11.1 Frontend . . . 34
11.2 RFC . . . 34
11.3 SAP-System . . . 34

12 Entwurf der RFC-Lösung 35
12.1 Frontend . . . 35
12.1.1 Verwendung der Hilfsklassen und eigenen Datentypen . . . 36
12.2 RFC . . . 36

12.3 SAP-System 36

13 Ausführung der RFC-Lösung 37
13.1 Frontend 37
13.1.1 Function Factory 37
13.1.2 Export als .jar 37
13.2 RFC 37

14 Bewertung / Fazit 38
14.1 Bewertung der Webservice-Lösung 39
14.2 Bewertung der RFC-Lösung 41
14.3 Gegenüberstellung der Lösungen 42
14.4 Entscheidung 43
14.5 Aufgetretene Probleme 44

Glossar 45

A Anwendungsfalldiagramm 49

B Anwendungsfälle 50
B.1 Selektieren 50
B.2 Verbinden 50
B.3 Anzeigen 51
B.4 Anlegen 52
B.5 Ändern 53
B.6 Löschen 55

C UML-Diagramme für das Frontend 57
C.1 Controller 57
C.2 Services (Webservice-Lösung) 58
C.3 Services (RFC-Lösung) 59

D Sequenzdiagramme für das Frontend (Webservices) 60
D.1 Webservice-Lösung 60
D.2 RFC-Lösung 62

E GUI des Frontends 65

F Diagramme zur Performanceanalyse 66

G Übersicht der erstellten Grafiken und Diagramme 68

Index 69

Literaturverzeichnis 71

Abbildungsverzeichnis

1 Die SAP-GUI 5
2 Angelegter Funktionsbaustein 8
3 Menü zum Anlegen eines Webservices 9
4 Transaktion WSCONFIG zur Freigabe von Webservices 10
5 Transaktion WSADMIN 10
6 Der BAPI-Explorer 12
7 Relevante Methoden für das Anlegen von BPs 13
8 Relevante Methoden für das Anzeigen von BPs 14
9 Relevante Methoden für das Ändern von BPs 15
10 Relevante Methoden für das Löschen von BPs 15
11 Paketdiagramm des Frontends 18
12 Detaillierte Darstellung der Service-Fassade (Webservice-Lösung) 19
13 Grafische Benutzeroberfläche des Frontends 20
14 Anlegen eines eigenen Datenelements 28
15 Anlegen eines eigenen Tabellentyps 29
16 Detaillierte Darstellung der Service-Fassade (RFC-Lösung) 35
17 Anwendungsfalldiagramm 49
18 UML-Diagramm des Controllers 57
19 UML-Diagramm der Services und der Servicefassade (WS) 58
20 UML-Diagramm der Services und der Servicefassade (RFC) 59
21 Sequenzdiagramm: Erfolgreiches Anlegen eines Business Partners (WS) 60
22 Sequenzdiagramm: Erfolgreiches Löschen eines Business Partners (WS) 61
23 Sequenzdiagramm: Erfolgreiches Anzeigen der Business Partner (WS) . 61
24 Sequenzdiagramm: Erfolgreiches Ändern der Business Partner (WS) . . 62
25 Sequenzdiagramm: Erfolgreiches Anzeigen der Business Partner (RFC) . 62
26 Sequenzdiagramm: Erfolgreiches Löschen eines Business Partners (RFC) 63
27 Sequenzdiagramm: Erfolgreiches Ändern der Business Partner (RFC) . . 63
28 Sequenzdiagramm: Erfolgreiches Anlegen eines Business Partners (RFC) 64
29 Grafische Benutzeroberfläche des Frontends mit Legende 65
30 Diagramm für die Laufzeitanalyse: Verbindungen 66
31 Diagramm für die Laufzeitanalyse: Aufrufe 66
32 Histogramm für die Laufzeitanalyse: Verbindungen 67
33 Histogramm für die Laufzeitanalyse: Aufrufe 67

Tabellenverzeichnis

1 Auflistung der einzelnen Services 22
2 Verwendete Datentypen (SAP) 23
3 Gegenüberstellung der Bewertungen 43
4 Auflistung der einzelnen Grafiken und Diagramme 68

Danksagungen

Im Rahmen dieser Bachelorarbeit möchte ich allen Menschen danken, die mich auf diesem Weg begleitet und unterstützt haben.
Zu allererst danke ich meinen Eltern, die es mir ermöglicht haben, mich voll auf mein Studium zu konzentrieren und die mir in allen Belangen den Rücken freigehalten haben.
Ein weiterer Dank geht an meinen besten Freund Niklas Bastian, der mich motiviert hat weiter zu schreiben, auch wenn eigentlich gar nichts mehr ging.
Natürlich danke ich auch Prof. Dr. Bernd Kahlbrandt, meinem Betreuer und Prüfer, sowie Prof. Dr.-Ing. Martin Hübner, meinem Zweitprüfer, für die Zeit und Unterstützung bei dieser Bachelorarbeit.
Auch Prof. Dr. Birgit Wendholt möchte ich meinen Dank aussprechen, da sie mich mit hilfreicher Literatur versorgt hat.
Meinen Korrekturlesern sei an dieser Stelle auch der Dank ausgesprochen, es ist schon erstaunlich, wo sich der Fehlerteufel überall einschleichen kann.
Zuletzt geht ein Dank an die Firma *C1WPS* und damit auch an Dr. Carola Lilienthal, ohne die ich nicht zu *C1WPS* gekommen wäre, an Jens Barthel und Tobias Rathjen, die mir immer während der Bachelorarbeit Rede und Antwort standen.

1 Einleitung

1.1 Das Problem

In der heutigen Welt werden immer mehr Tätigkeiten von Computern übernommen, um Arbeitsabläufe für den Menschen zu vereinfachen und zu beschleunigen. So verhält es sich auch mit Geschäftsprozessen, die durch die Software der SAP AG abgebildet werden. Der weitgehend automatisierte Ablauf einzelner Geschäftsprozesse, wie zum Beispiel Lohnbuchungen, Warenwirtschaft oder in diesem Fall die Verwaltung von Geschäftspartnern und Kontakten durch *SAP-BP*, bedarf aber noch immer einer gewissen Administration durch den Menschen. Eben diese Einbindung des Menschen erfordert eine Schnittstelle, die möglichst flexibel und unabhängig vom Ausgangssystem und von Programmiersprachen gestaltbar sein sollte.

1.2 Ein Lösungsansatz

Um die gewünschte Flexibilität zu erreichen, bietet es sich an, die Webservice-Funktionalität von SAP zu nutzen. Durch die Standardisierung von Webservices kann man diese mit standardisierten Programmen bzw. Programmiersprachen nutzen und zum Beispiel in bestehende Systeme integrieren. Dem gegenüber steht eine Lösung über den proprietären *Remote Function Call(RFC)*-Ansatz von SAP. Beide Lösungen führen zum Ziel, haben aber Vor- und Nachteile, die in dieser Arbeit untersucht werden.

2 Das Unternehmen C1WPS

Dieses Kapitel gibt einen kurzen Einblick in die Struktur der *C1WPS*. Ebenso wird auf die Wahl des Unternehmens eingegangen.

2.1 Wahl des Unternehmens

Durch die Vorlesung *„Architektur von Informationssystemen“* im Sommersemester 2008 bei Dr. Carola Lilienthal kam ich das erste Mal in Kontakt mit der *C1WPS*. Die spontane Frage an Frau Dr. Lilienthal ob die *C1WPS* auch Bachelorarbeiten betreut brachte den Kontakt zu Dr. Guido Gryczan. In einem Gespräch kamen Herr Dr. Gryczan und ich überein, dass es mehrere mögliche Themen bei der *C1WPS* gibt. Das in dieser Bachelorarbeit aufgegriffene Thema hat mich am meisten interessiert, da es sicherlich eine gute Referenz ist, schon einmal mit SAP gearbeitet zu haben. Ein wichtiger Aspekt dafür, die Bachelorarbeit bei der *C1WPS* zu schreiben, sind die universitären Wurzeln des Unternehmens, wodurch das Unternehmen unabhängig von Herstellern arbeiten und vor allem neutrale Bewertungen von Systemen und Technologien vornehmen kann.

2.2 C1WPS

Das Unternehmen *C1WPS* wurde 1999 als Tochter der *itelligence AG* gegründet. Die heutige Geschäftsführung setzt sich aus

- Dr. Guido Gryczan
- Ute Zimmermann
- Prof. Dr.-Ing. Heinz Züllighoven

zusammen. Es besteht eine enge Zusammenarbeit zwischen der *C1WPS* und dem Arbeitsbereich *Softwaretechnik* der Universität Hamburg. Seit 2004 ist die *C1WPS* Mitglied der *C1-Gruppe*, zu der ebenfalls 18 weitere Schwesterfirmen gehören. Die angebotenen Dienstleistungen der *C1WPS* erstrecken sich von kompletten Neuentwicklungen bzw. Weiterentwicklungen von Softwaresystemen und Sanierung von Altsystemen über die Umgestaltung von Softwaresystemen bis hin zu speziell angepassten Schulungen (C1WPS, 2008).

3 Aufgabenstellung

Die Aufgabenstellung dieser Bachelorarbeit ist, zu dem laufenden SAP-System inklusive *SAP-BP* ein webservicebasiertes Frontend zur Verwaltung von Geschäftspartnern zu entwickeln, um damit die Nutzbarkeit der Webservicefunktionalität von SAP für den weiteren Einsatz in der Praxis zu testen. Ein Vergleich zum proprietären *Remote Function Call*-Ansatz (RFC) von SAP ist vorgesehen und wird im Rahmen dieser Arbeit ebenfalls vorgenommen. Der Aspekt der Sicherheit einer Übertragung ist vernachlässigbar, da der Zugriff auf das SAP-System entweder aus dem lokalen Netzwerk oder über eine verschlüsselte VPN-Verbindung von außerhalb erfolgt. Eine Spezifizierung der zu nutzenden Programmiersprache für das Frontend gibt es nicht. Abschließend soll das Projekt nach geeigneten Kriterien bewertet werden.

3.1 Ausgangslage

Der Zugriff auf das bestehende SAP-System ist sowohl über das lokale Netzwerk, als auch über das Internet mit einer VPN-Verbindung möglich. Die Entwicklung von Benutzeroberflächen für ABAP-Anwendungen findet zurzeit mit *Dynpro* bzw. *Web Dynpro*, das nach dem *MVC*-Prinzip arbeitet, statt. Ziel ist es, sich von der Oberflächenentwicklung mit *(Web) Dynpro* zu lösen.

3.2 Darstellung des bestehenden Systems

In einem Firmensitz der *C1WPS*, auf dem Gelände des Informatikums der Universität Hamburg in Stellingen, läuft ein SAP-System inklusive *SAP-BP* (Version: SAP NetWeaver 2004s ABAP Server Trial Version). Das SAP-System läuft in einer virtuellen Maschine mit *VMware Virtual Server 1.0.x* unter *openSuse 10.2*. Eine Verbindung über einen VPN-Tunnel von außen ist nur mit einem aktiven Benutzeraccount vom Informatikum mit entsprechenden Rechten möglich und ermöglicht das Arbeiten von einem beliebigen Standort über das Internet.

3.3 Lösungsansatz

Der *SAP Application Server (SAP-AS)* bietet eine Webservice-Funktionalität an, die sich als Basis des *Internet Communication Frameworks (ICF)* von SAP bedient. Für eine *Webservice*-basierte Lösung wird auf diese Funktionalität zurückgegriffen. Das Frontend wird aufgrund meiner eigenen Vorkenntnisse durch das Studium in Java entwickelt werden. C# wäre bei entsprechenden Vorkenntnissen auch eine Alternative gewesen. Die angestrebte Lösung soll das

- Anzeigen,
- Anlegen,
- Ändern,
- Löschen

von Kontakten ermöglichen. Im hier dargestellten Szenario kennen sich der Service-Requester (Java-Anwendung) und der Service-Provider (SAP-System), was zwar dem Ansatz von *Webservices* über die Positionstransparenz widerspricht, hier aber Sinn macht (Vgl. Abschnitt 6.2, Seite 11).

3.4 Was spricht für diese Lösung

Der wichtigste Aspekt ist die dazu gewonnene Flexibilität (Vgl. Kapitel 14, Seite 38), da man im Bereich Frontend-Entwicklung nicht mehr nur auf eine Programmiersprache beschränkt ist, zumal die Entwicklung in ABAP/ABAP-O bezüglich der Komplexität der Sprache schwieriger ist als in anderen gängigen Sprachen wie beispielsweise Java oder C# (Vgl. Abschnitt 4.4.2, Seite 6). Diese Unabhängigkeit hat auch wirtschaftliche Vorteile, da man nicht mehr an vergleichsweise teuren ABAP-O-Entwicklern gebunden ist, sondern wie in diesem Fall, auf „günstigere“ Java-Entwickler zurückgreifen kann. Durch die Unabhängigkeit bezüglich der Programmiersprache kann eine *Webservice*-basierte Lösung wesentlich einfacher in eventuell bestehende Softwaresysteme integriert werden. Die Abhängigkeit von der *SAP-GUI*, die zur Ausführung von ABAP-Programmen zwingend erforderlich ist, wird gelöst. Damit geht neben dem Ansatz mit Webservices der alternative Ansatz mit *RFC* einher.

4 SAP

Die SAP AG ist einer der größten Softwarehersteller weltweit und beschäftigt heute ca. 52.000 Mitarbeiter. SAP ist auf dem Sektor *ERP*-Software tätig und bietet entsprechende Softwarelösungen für Unternehmen jeder Größe an (SAP AG, 2008), (Wikipedia/SAP, 2008).

4.1 Das Modulsystem von SAP

Die Software von SAP bedient sich eines Modulkonzepts, bei dem im Kern eine zentrale Datenbank steht. Die Zugriffe auf diese zentrale Datenbank koordiniert der *SAP Application Server (SAP-AS)*. Die einzelnen Anwendungsprogramme (von SAP oder Eigenentwicklungen) werden auf dem AS ausgeführt und nutzen die vom AS zur Verfügung gestellte Funktionalität zur Datenspeicherung.

4.2 Eigene Erweiterungen

Selbstentwickelte (ABAP-)Erweiterungen für ein SAP-System lassen sich in der *SAP-GUI* (Vgl. Abbildung 1, Seite 5) neben anderen SAP-Komponenten mit einbinden. Haben die eigenen Erweiterungen eine grafische Benutzeroberfläche, so wird diese nach Einbindung in die *SAP-GUI* als ein weiteres Fenster und somit als ein Teil der *SAP-GUI* dargestellt.

Abbildung 1: Die SAP-GUI

4.3 SAP BP

SAP-BP dient der zentralen Verwaltung von Geschäftspartnern und deren Kontaktdaten. Das Modul *Business Partner* ermöglicht eine redundanzfreie und konsistente Kontaktdatenhaltung. Die abgespeicherten Geschäftspartnerdaten lassen sich nach Rollen und

Kategorien gruppieren. Durch kategoriespezifische Einträge lassen sich Zusatzinformationen hinterlegen, sowie Beziehungen von Geschäftspartnern untereinander darstellen. Angelegte Geschäftspartner werden bereits beim Anlegen einer der drei Gruppen zugeordnet:

- 1 - Natürliche Personen

- 2 - Organisationen / Firmen

- 3 - Gruppen von nat. Personen oder Firmen

4.4 ABAP und ABAP-O

Die Sprachwelt der von SAP entwickelten Sprachen lässt sich grob in zwei Abschnitte teilen. Auch wenn es in beiden Abschnitten um die gleiche Sprache geht, so verfolgen die verschiedenen Entwicklungsstadien doch unterschiedliche Ansätze der Programmierung (Keller und Krüger, 2007, Seite 23-37), (Wikipedia/ABAP, 2008).

4.4.1 ABAP

Die Sprache ABAP ist eine von SAP entwickelte Programmiersprache, die ihre Wurzeln bereits in den 1970er Jahren hat. In den Anfängen stand ABAP noch für *Allgemeiner Berichts-Aufbereitungs Prozessor* und diente der Programmierung von kundenspezifischen Auswertungen von Datenbanken. Anfang der 1990er Jahre wurde SAP R/3 veröffentlicht und das neue ABAP/4 wurde als eine Programmiersprache der 4. Generation (4GL) vorgestellt.

4.4.2 ABAP-O

Ende der 1990er Jahre erfolgte die Vorstellung von ABAP-Objects (ABAP-O). ABAP-O stellt eine objektorientierte Erweiterung der bisher rein prozeduralen Sprache ABAP dar, in der viele Paradigmen der objektorientierten Programmierung umgesetzt wurden, wie z.B.:

- Kapselung
- Vererbung
- Polymorphie

Einzig Mehrfachvererbung und das Überladen von Methoden wurden nicht berücksichtigt. Durch die ständige Weiterentwicklung der Sprache ABAP/ABAP-O stehen mittlerweile eine Vielzahl von Schlüsselwörtern (über 700) zur Verfügung, von denen ein Teil einzig wegen der Abwärtskompatibilität übernommen wurde, aber in Neuentwicklungen nicht mehr verwendet werden soll. Dieser Aspekt und die Tatsache, dass die Erlernbarkeit durch verschiedene mögliche Notationen nicht so einfach ist, wie von SAP dargestellt, machen eine Entwicklung mit ABAP-O zeitaufwändig und komplex.

5 Webservice zwischen Java und SAP - Ein Beispiel

Bevor mit der Planung der Lösung begonnen wird, soll ein Versuch gemacht werden, um die *Webservice*-Funktionalität von SAP anhand eines kleinen Beispiels zu testen. Dieser Versuch dient einem Durchstich zur ersten Feststellung der technischen Machbarkeit (Royce, 1998), (Royce, 1970). Hierfür soll ein einfacher *Webservice* zur Berechnung der Wurzel einer Zahl implementiert werden. Das SAP-System wird der Service-Provider sein, eine kleine Java-Anwendung der Service-Requester.

5.1 Webservice-Funktionalität von SAP

Der *SAP-AS (NetWeaver)* kann sowohl als Service-Requester, als auch als Service-Provider agieren. In beiden Fällen bedient sich die *Webservice*-Funktionalität von SAP des *Internet Communication Frameworks (ICF)*. Es lassen sich mittels bereits bestehender Dialoge aus Funktionsbausteinen und Funktionsgruppen *Webservices* und die dazugehörigen WSDL-Dokumente erzeugen. Auch eine Publizierung über UDDI wird unterstützt.

5.2 Anlegen der Berechnungsfunktion

Es muss zuerst in der Transaktion SE80 *(Object Navigator)* des SAP-Systems ein neuer *RFC*-fähiger Funktionsbaustein in einer Funktionsgruppe angelegt werden. Für den Funktionsbaustein werden Import- und Exportparameter festgelegt. In diesem Fall reichen zwei Parameter:

Importparameter Typ: Float, Zahl aus der die Wurzel gezogen werden soll

Exportparameter Typ: Float, Ergebnis

Sind die Parameter angelegt, so wird die Funktion implementiert (Vgl. Abbildung 2, Seite 8) und im Anschluss im System aktiviert.

5.3 Erzeugung und Aktivierung des Webservices

Über das Kontextmenü des Funktionsbausteins lässt sich nun direkt ein *Webservice* anlegen. Man trägt im erscheinenden Menü (Vgl. Abbildung 3, Seite 9) die Servicedefinition, sowie den Endpunkt (Funktionsbaustein) ein. Wählt man bei der Servicedefinition den Namen des Funktionsbausteins, so kommt es beim späteren Konsumieren des WSDL-Dokuments zu Schwierigkeiten (Vgl. Abschnitt 5.3.1, Seite 8). Wichtig hierbei ist, dass die Checkbox „Mapping der Namen“ aktiviert ist, da nur so die Bezeichner des Endpunkts übernommen werden. Ist das Anlegen beendet, so muss der *Webservice* noch freigegeben werden. Hierfür stellt SAP die Transaktion *WSCONFIG* (Vgl. Abbildung 4, Seite 10) zur Verfügung. Nach Eingabe der Servicedefinition kann man Sicherheitsprofile, sowie Verbindungsdetails des *ICF* festlegen. Im Bereich der *ICF*-Details ist es wichtig bei dem Reiter „Logon Data“ den SAP-Benutzer einzutragen, unter dem der Service ausgeführt werden soll, da sonst bei jeder Nutzung des Services ein Login erfolgen müsste.

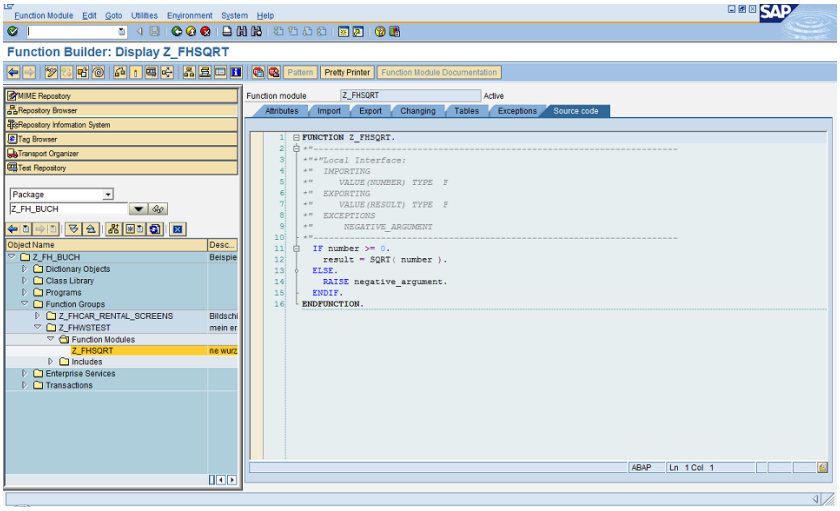


Abbildung 2: Angelegter Funktionsbaustein

5.3.1 Problem bei der Namensgebung

Wird beim Anlegen einer Servicedefinition für diese der gleiche Name verwendet wie für den Funktionsbaustein aus dem sie erstellt wird, so treten beim Konsumieren des WSDL-Dokuments mit *wsimport* Schwierigkeiten auf. Die Schwierigkeiten bestehen darin, dass bei den generierten Java-Klassen teilweise die Rückgabewerte von Methoden einfach unterschlagen werden und der Code damit fehlerbehaftet ist. Zusätzlich werden nicht alle benötigten Klassen generiert und die erzeugten Klassen sind somit unbrauchbar. Um dieses Problem zu vermeiden, sollte man bei der Wahl des Namens für die Servicedefinition auf keinen Fall den gleichen Namen wie für den Funktionsbaustein wählen. Für die späteren Anwendungen habe ich mich entschlossen, die Servicedefinitionen zur besseren Zuordnung nach den Funktionsbausteinen zu benennen, aus denen sie erzeugt wurden und an diesen Namen das Kürzel *_WS* anzuhängen.

5.4 Konsumieren des Webservices

Der in Abschnitt 5.3 erzeugte und freigegebene *Webservice* soll nun konsumiert werden. Dazu ruft man die Transaktion *WSADMIN* (Vgl. Abbildung 5, Seite 10) auf und wählt unter *SOAP Applications für RFC-Kompatible Funktionsbausteine* die Servicedefinition des erstellten *Webservices* aus. Über die Eigenschaften des Services kann man sich das dazugehörige WSDL-Dokument in einem Internetbrowser anzeigen lassen. Anhand des WSDL-Dokuments führt man nun den Befehl *wsimport* im Sourcefolder des Zielprojekts aus. Die generierten Artefakte stellen die Schnittstelle zum *Webservice* dar und können in der Java-Anwendung nun wie jede andere Klasse verwendet werden (Eberhart und Fischer, 2003).

5.4.1 Probleme mit der Pfadangabe des WSDL-Dokuments

Der *SAP-AS* stellt die jeweiligen WSDL-Dokumente zu den Servicedefinitionen unter einer *URL* zur Verfügung. An diese URL ist der durch ein „?“ abgetrennte Query-String

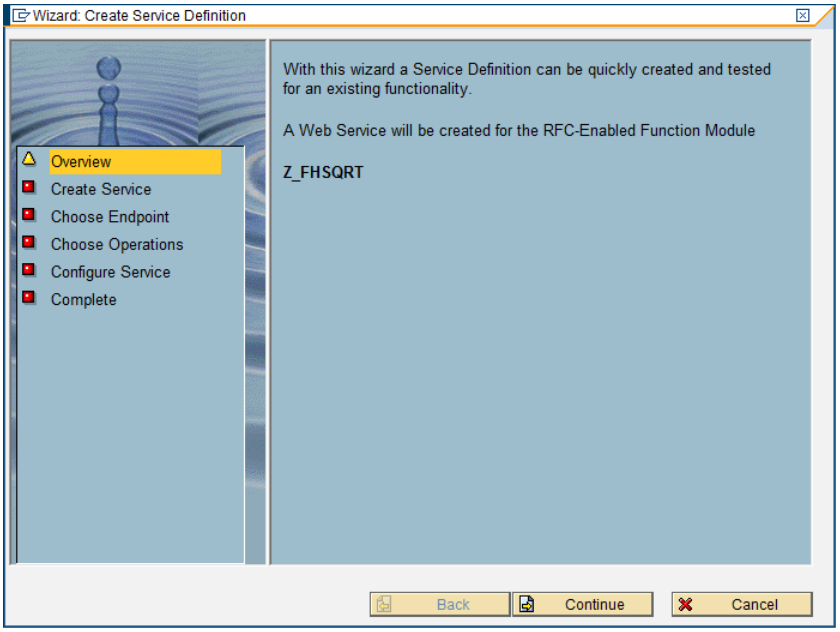


Abbildung 3: Menü zum Anlegen eines Webservices

angehängt, in dem mehrere durch ein „&“ getrennte Informationen mitgeführt werden. Eine Standardinformation ist die Angabe der WSDL-Version (z.B. 1.1), aber der *SAP-AS* führt noch weitere Informationen auf. Durch die zusätzlich angegebenen Informationen kommt es beim Konsumieren des WSDL-Dokuments mit dem Tool *wsimport* zu Fehlern. Eine Anpassung der *URL* durch das Weglassen von nicht benötigten Informationen (z.B. die SAP Client-Nummer) ist für die Verarbeitung des WSDL-Dokuments mit einem geeigneten Tool (in diesem Fall *wsimport*) notwendig.

5.5 Ergebnis und Ausblick

Die Kommunikation mit Hilfe von *Webservices* wurde realisiert und funktioniert einwandfrei. Anhand dieser Information kann mit der Planung der eigentlichen Lösung begonnen werden. Im Hinblick auf komplexere Datentypen, wie die Darstellung von internen Tabellen in SAP, ist mit Schwierigkeiten bei den *Webservices* zu rechnen. Ein Codelisting des behandelten Beispiels befindet sich auf der beigelegten CD (*Listings⇒Java⇒Beispiel_WS*).

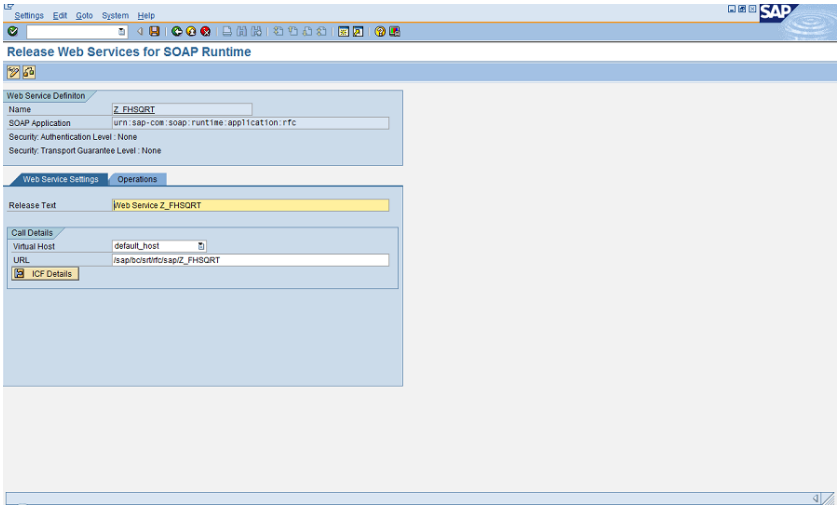


Abbildung 4: Transaktion WSCONFIG zur Freigabe von Webservices

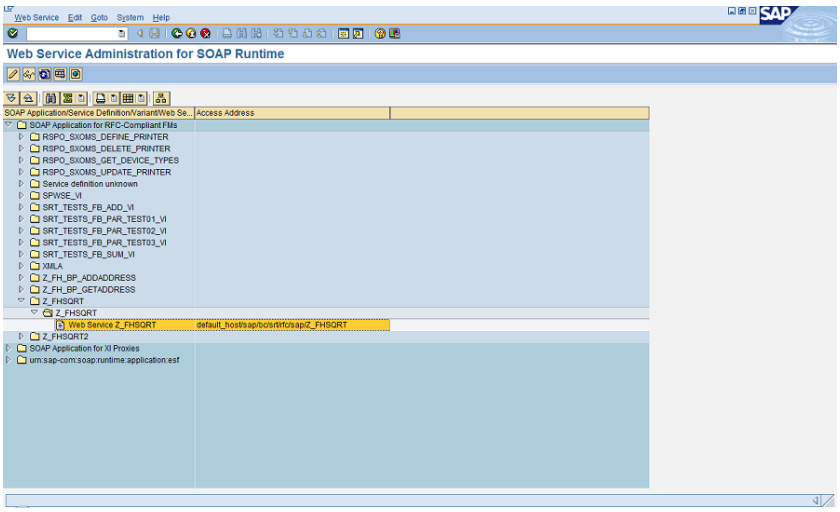


Abbildung 5: Transaktion WSADMIN

6 Planung der Webservice-Lösung

Nachdem der erste Durchstich im Bereich der *Webservices* unter SAP (Vgl. Kapitel 5, Seite 7) erfolgreich war, kann nun mit der Planung für die Lösung der eigentlichen Aufgabenstellung begonnen werden. Es ist nun bekannt, wie die *Webservice*-Schnittstelle von SAP zu nutzen ist und wie *Webservices* Java-seitig konsumiert werden können.

6.1 Frontend

Gemäß der Aufgabenstellung und meiner eigenen Erfahrungen soll das Frontend in Java entwickelt werden. Bei der grafischen Oberfläche fällt die Entscheidung auf eine *Swing*-Oberfläche, da es sich bei *Swing* um ein leichtgewichtiges Framework handelt. Das Rendering der einzelnen Komponenten findet direkt in Java statt, so dass die Komponenten auf allen Plattformen gleich aussehen und funktionieren. Das „Look and Feel“ von allen Komponenten kann zur Laufzeit geändert werden, um *Swing*-Applikationen besser in bestehende Systeme zu integrieren.
Die einzelnen Services sollen wie in Abschnitt 5.4 durch das Tool *wsimport* in das Frontend eingebunden werden. Durch das Aufrufen von *wsimport* auf das WSDL-Dokument des Webservices werden automatisch die Artefakte und ggf. Holder-Klassen für den Java-Endpunkt generiert.

6.2 Webservice

Für die vorliegende Aufgabenstellung ist das SAP-System der Service-Provider und das Java-Frontend der Service-Requester, da Services von einem SAP-System konsumiert werden sollen. Die *Webservices* sollen direkt aus bestehenden Funktionsbausteinen erstellt werden, wie es beim ersten Versuch zur technischen Machbarkeit in Kapitel 5 getestet wurde. Zur losen Kopplung sollen die in Abschnitt 3.3 dargestellten geplanten Funktionalitäten des Frontends nach Möglichkeit in einzelnen Services zur Verfügung stehen. Da die Services nur für die spezielle Verwendung mit dem hier zu entwickelnden Frontend zur Verfügung stehen sollen, ist eine Veröffentlichung in einer UDDI-Registry nicht erforderlich. Ebenso ist der Aspekt der Sicherheit bei der Übertragung zu vernachlässigen, da die Kommunikation entweder lokal im Firmennetz oder bei einer externen Verwendung über eine verschlüsselte VPN-Verbindung erfolgt.

6.3 SAP-System

Das laufende SAP-System der *C1WPS* braucht voraussichtlich nicht verändert zu werden, da das Modul *Business Partner* dort bereits installiert ist. Die Authentifikation am SAP-System, um auf *Business Partner* zuzugreifen, wird über einen SAP-seitig fest eingestellten Benutzer bei den *Webservices* realisiert, da sich die Anwender bereits beim Aufbau der VPN-Verbindung authentifizieren mussten.
Die Methoden, aus denen später die *Webservices* generiert werden sollen, können mittels des *BAPI-Explorers* (Vgl. Abbildung 6, Seite 12) aus der *BAPI* zu *SAP-BP* ermittelt.

Im *BAPI-Explorers* sind die Funktionsbausteine und Schnittstellen von *SAP-BP* aufgelistet und deren Verwendung inklusive der benötigten Parameter dokumentiert. Es wird auch auf den Speicherort der einzelnen Funktionsbausteine (Package, Funktionsgruppe) verwiesen.

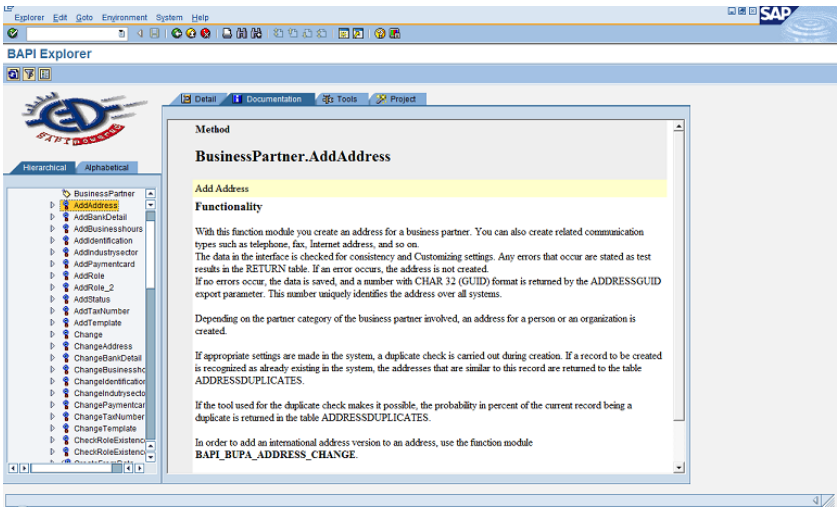


Abbildung 6: Der BAPI-Explorer

6.3.1 Relevante Funktionsbausteine in SAP BP

Über die *BAPI* wurden die relevanten Funktionsbausteine für die Funktionalität des Frontends ausfindig gemacht und auch im Modul *Business Partner* lokalisiert. Die entsprechenden Funktionsbausteine befinden sich im Package *BUPA* in der Funktionsgruppe *BUBA_3*. Für die gewünschten Anwendungen im Frontend haben sich folgende Funktionsbausteine als notwendig herausgestellt:

Anlegen: Zum Anlegen eines Business Partners greift man auf den Funktionsbaustein CREATE_FROM_DATA zurück, der einen neuen Business Partner im System anlegt und auch gleich Kontaktinformationen und Art des Kontakts (Person, Firma, Gruppe von Firmen oder Personen) einträgt (Vgl. Abbildung 7, Seite 13). Der Funktionsbaustein erhält als Parameter komplexe Datentypen, die eine Repräsentationen von SAP-Tabellen sind, und Strings. Als Rückgabewert werden die eindeutige Identifikationsnummer des neu angelegten Business Partners (im Fehlerfall „0“) und eine Tabelle mit allen aufgetretenen Fehlern geliefert.

Anzeigen: Das Anzeigen der Business Partner erfolgt über den Zugriff auf den Funktionsbaustein SEARCH. Alle vorhandenen Business Partner erhält man durch den Aufruf von SEARCH mit einer Wildcard („*“) für die eindeutige Identifikationsnummer. Der Rückgabewert dieses Bausteins ist zum einen eine Tabelle mit den eindeutigen Identifikationsnummern der Business Partner, auf die die Suche zutrifft und zum anderen eine Tabelle mit allen aufgetretenen Fehlern. Um die Business Partner anzuzeigen, müssen noch die Kontaktdaten zu den Identifikationsnummern aus dem Suchergebnis ausgelesen werden. Dazu bedient man sich der

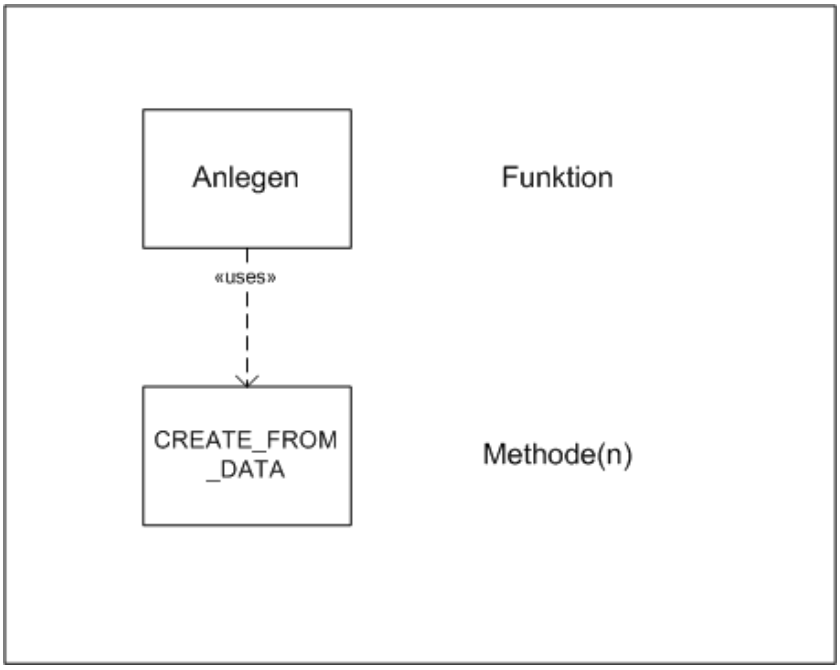


Abbildung 7: Relevante Methoden für das Anlegen von BPs

Funktionsbausteine ADDRESS_GETDETAIL und CENTRAL_GETDETAIL. Beide Funktionsbausteine erwarten als Aufrufparameter eine eindeutige Identifikationsnummer eines Business Partners und liefern Adressdetails bzw. kontaktspezifische Details in Abhängigkeit von der Art des Kontakts zurück (Vgl. Abbildung 8, Seite 14). Durch die Spezifizierung von mehreren Details kann man beim Aufruf des Funktionsbausteins SEARCH auch direkt nach Business Partnern suchen.

Ändern: Zur Änderung eines Business Partners greift man auf zwei Funktionsbausteine zurück (Vgl. Abbildung 9, Seite 15). CENTRAL_CHANGE ermöglicht das Ändern von kontaktspezifischen Details und erwartet als Eingabe die eindeutige Identifikationsnummer eines Business Partners. Weitere Parameter sind die geänderten Daten in Form von komplexen Datentypen. Als Rückgabeparameter erhält man eine Tabelle mit den eventuell aufgetretenen Fehlern. Mit ADDRESS_CHANGE kann man die Adressdaten (Anschrift, Kommunikationsdetails) eines Business Partners ändern. Anhand einer übergebenen Identifikationsnummer und den geänderten Daten in Form von komplexen Datentypen werden die Adressdaten geändert. Wie auch schon CENTRAL_CHANGE gibt ADDRESS_CHANGE eine Tabelle mit den aufgetretenen Fehlern zurück.

Löschen: Zum Löschen eines Business Partners wird aus der Funktionsgruppe BUD0_ARCH der Funktionsbaustein BUP_BUPA_MASS_DELETE verwendet (Vgl. Abbildung 10, Seite 15). Um den verwendeten Funktionsbaustein muss ein Wrapper geschrieben werden, da als Übergabeparameter ein Bereich von zu löschenden Business Partnern erwartet wird, aber im konkreten Fall immer nur je ein Business Partner gelöscht werden soll. Geplant ist, dass die eindeutige Identifikationsnummer des zu löschenden Business Partners der einzige Überga-

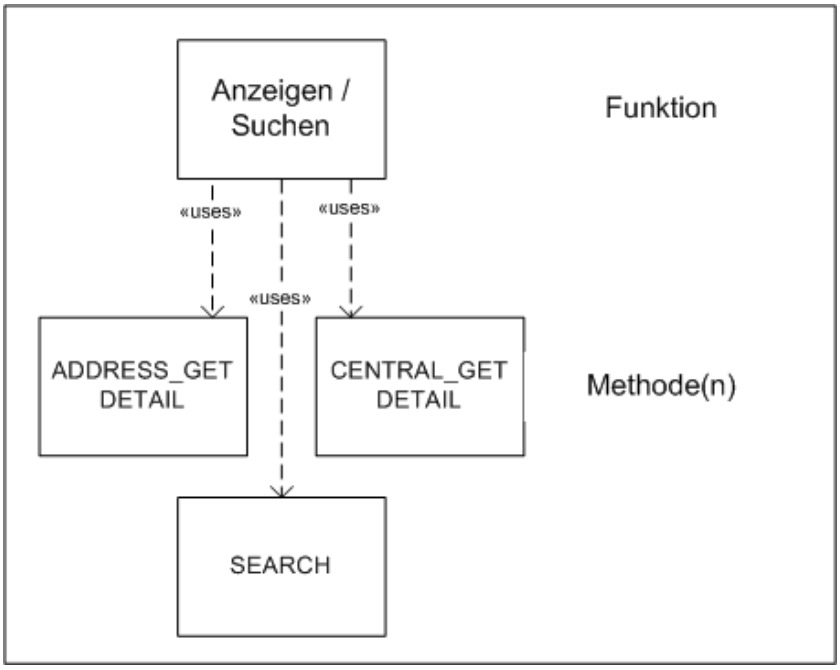


Abbildung 8: Relevante Methoden für das Anzeigen von BPs

beparameter ist. Rückgabeparameter ist zum einen wieder eine Tabelle mit allen möglichen aufgetretenen Fehlern und zum anderen eine Bestätigung über den erfolgreichen Löschvorgang. Ebenfalls erwartet der oben genannte Funktionsbaustein das Setzen von diversen Flags, die Restriktionen in Bezug auf das Löschen repräsentieren. Diese Restriktionen werden zur Vereinfachung fest im Code des Wrappers verankert.

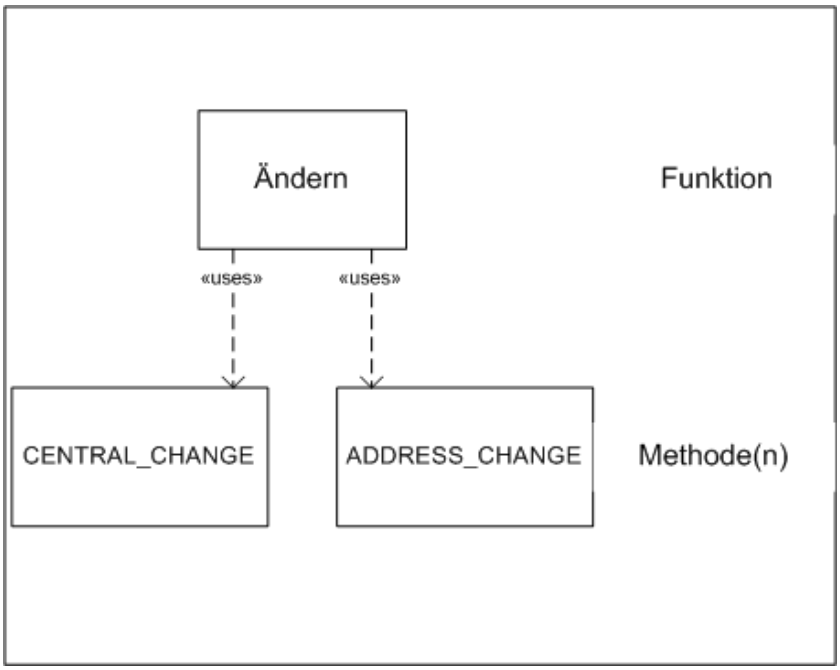


Abbildung 9: Relevante Methoden für das Ändern von BPs

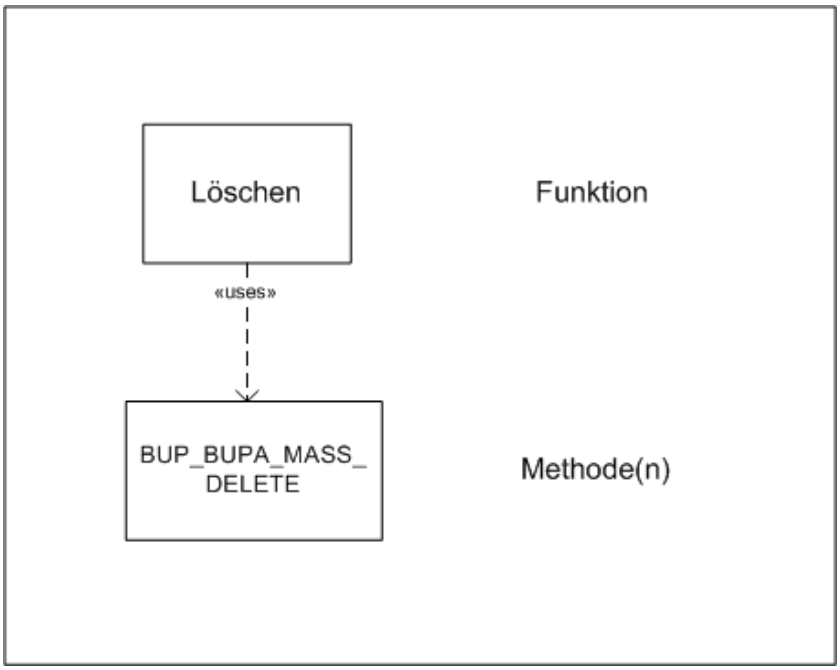


Abbildung 10: Relevante Methoden für das Löschen von BPs

7 Entwurf der Webservice-Lösung

Nach der grundsätzlichen Planung der Lösung soll in diesem Kapitel der konkrete Entwurf der Lösung erfolgen. Der Entwurf beinhaltet unter anderem Design- und Strukturentscheidungen für das Java-Frontend und die Beschreibung notwendiger Veränderungen am SAP-System und an den *Webservices*.

7.1 Frontend

Der Entwurf des Frontends macht den größten Teil dieses Kapitels aus, da hier grundlegende Entscheidungen zum Softwaredesign und zur Struktur getroffen werden müssen. Ebenso wird für das Frontend entschieden, welche Daten eines Business Partners dargestellt werden sollen und welche als „überflüssiger Ballast“ vernachlässigt werden können.

7.1.1 Softwaretechnische Struktur

Bei der grundsätzlichen Struktur des Frontends fällt die Wahl auf ein leicht abgewandeltes MVC-Pattern (Vgl. Abbildung 11, Seite 18), bei dem der Client für das Notify verantwortlich ist (Gamma u. a., 1996, Seite 287-300). Durch diesen Ansatz ergibt sich eine gute Erweiterbarkeit und Austauschbarkeit der einzelnen Komponenten. Gerade für den zweiten Teil dieser Bachelorarbeit, die Realisierung der Aufgabenstellung mit *SAP RFC*, ist eine Austauschbarkeit einzelner Komponenten (in diesem Fall: Austausch des Modells) vom Auftraggeber gewünscht. Im Standard-MVC unterrichtet das Modell den View mittels eines Observers über Änderungen der Daten. Das ist in dieser Form durch den Einsatz von *Webservices* zwischen dem Modell (das über *Webservices* angesprochene SAP-System) auf der einen und dem View und Controller auf der anderen Seite so nicht möglich. Eine Alternative ist das regelmäßige Aktualisieren der angezeigten Daten durch das Frontend. Das Aktualisieren könnte sowohl automatisiert mit festen Intervallen als auch manuell auf Anfrage erfolgen. Wichtig ist vor allem die Aktualisierung der Daten unmittelbar vor einem verändernden schreibenden Zugriff (Löschen oder Ändern eines Business Partners). Die Entscheidung ist in diesem Fall auf eine Aktualisierung direkt vor dem schreibenden Zugriff gefallen. Haben sich Daten im Vergleich zum lokalen Stand geändert und soll auf diese Daten schreibend zugegriffen werden, so muss der Benutzer den Vorgang bestätigen bzw. abbrechen. Die Kommunikation von der GUI mit den Services und umgekehrt läuft immer über den Controller ab.

Die *Webservices* sollen, um die in Abschnitt 6.2 erwähnte lose Kopplung bei den Services zu erhalten, jeweils so modular wie möglich gehalten werden. Dies bedeutet, dass es für jede Interaktionsart mit dem SAP-System einen eigenen *Webservice* geben soll. Eben diese granulare Aufteilung erlaubt eine möglichst hohe Modularität in Bezug auf die Services und wirkt einer nicht sinnvollen Bündelung von Services nach ihren Eingabeparametern entgegen. Nun kann es sein, dass eine Funktionalität des Frontends aus mehreren einzelnen Interaktionen mit dem SAP-System besteht (z.B. das Anzeigen der Business Partner). Für diesen Fall wird für diese Funktionalität eine Fassade vor die

einzelnen Services gesetzt und die einzelnen Services in Unterpaketen gruppiert (Vgl. Abbildung 12, Seite 19). Zuletzt wird vor die einzelnen Funktionalitäten, seien es nun einzelne Services oder mehrere, durch eine Fassade dargestellte Services, wiederum eine Fassade gesetzt, um eine klar definierte Schnittstelle zum Modell zu erhalten. Über die vorgeschalteten Fassaden ist der Zugriff auf die einzelnen Funktionalitäten des Frontends über klar definierte Schnittstellen einfacher möglich (Vgl. Abbildung 19, Anhang Seite 58). Außerdem können bei Bedarf Services ohne große Veränderungen ausgetauscht, entfernt oder hinzugefügt werden. Jeder einzelne Service wird in einem eigenen Paket eingebunden.

Die Kommunikation zwischen der grafischen Benutzeroberfläche und den Services (über die ServiceFassade) läuft über einen Controller ab. Für festgelegte Methodenbezeichnungen der Frontendfunktionalität muss der Controller das Interface IController implementieren. Auch für die GUI und die ServiceFassade sollen Interfaces entworfen werden. Zum Sammeln von Informationen über den Fortschritt einzelner Abläufe ist der Controller ein Observer auf die Servicefassade, welche sowohl Observable für den Controller als auch Observer für die untergeordneten Fassaden ist.
Für die Fehlerbehandlung werden aufgetretene Exceptions mittels *throws Exception* von den Services an den Controller durchgereicht, um dort verarbeitet zu werden.
Für die vier Funktionalitäten des Frontends ergeben sich die in Anhang D.1 ab Seite 60 aufgeführten Sequenzdiagramme.

7.1.2 Design

Beim Design der Oberfläche für das Frontend steht die Übersichtlichkeit und Bedienbarkeit im Vordergrund. Dazu ist es wichtig, dass sich die Bedienung intuitiv erschließt und die Oberfläche nicht zu verspielt oder unübersichtlich ist. Auch ein Überladen mit unnötigen Funktionen und Informationen ist einer guten Bedienbarkeit nicht förderlich. Für eine klare Struktur bei der Darstellung habe ich mich dafür entschieden, die einzelnen Kategorien von Business Partnern (natürliche Personen, Firmen und Gruppen) auch in einzelnen Reitern mit einem JTabbedPane und darin enthaltenen JTables darzustellen. Die vier einzelnen Funktionalitäten des Frontends werden über vier einzelne JButtons realisiert und bedienen sich, sofern benötigt, der Daten des aktuell selektierten Business Partners aus der JTabbedPane (Vgl. Abbildung 13, Seite 20 und Anhang E, Seite 65). Einmalig müssen durch einen Knopfdruck beim Beginn der Arbeit mit dem Frontend die Services initialisiert werden. Da einige Aktionen (z.B. das Initialisieren der Services oder das Anzeigen aller Business Partner) länger dauern können und dem Benutzer nicht das Gefühl gegeben werden soll, dass die Applikation nicht weiterläuft, wird eine JProgressBar verwendet, um den Fortschritt der ausgeführten Aktion auch sichtbar zu machen.
Für einen Benutzer ergeben sich somit folgende Möglichkeiten der Interaktion mit dem Frontend:

Anzeigen: Das Anzeigen der Business Partner erfolgt durch Knopfdruck auf den entsprechenden JButton in der GUI. Der Fortschritt wird mit der JProgressBar dargestellt und die einzelnen Reiter der JTabbedPane werden aktualisiert. Nach dem

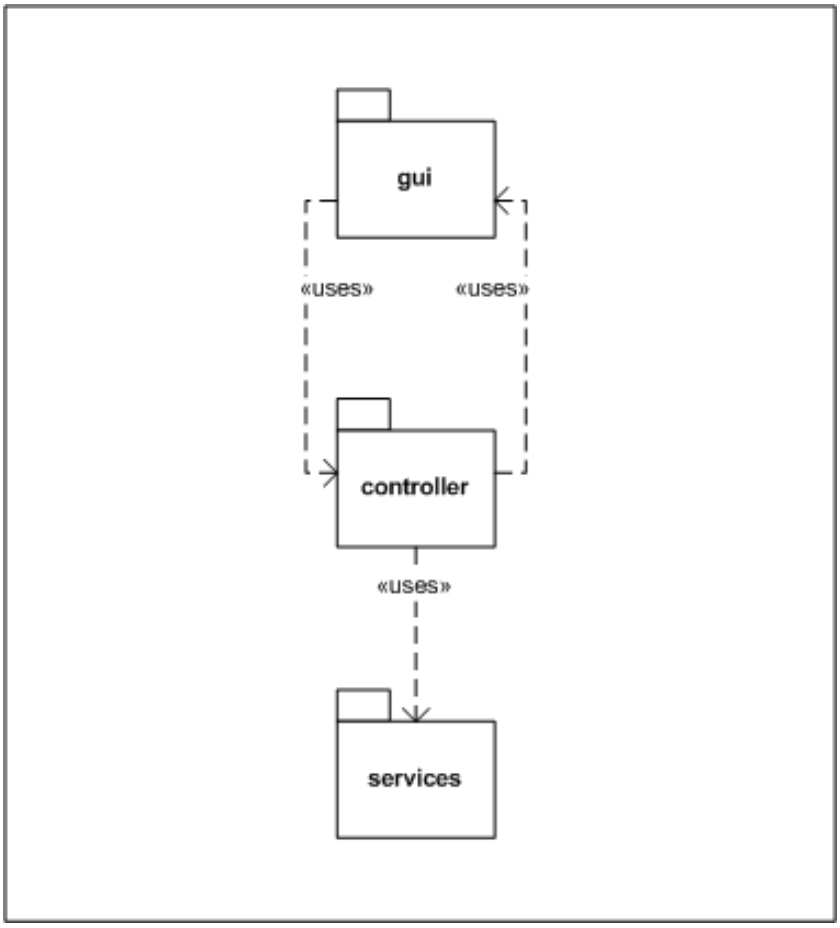


Abbildung 11: Paketdiagramm des Frontends

Starten des Frontends, wenn sich die Services initialisiert haben, wird das Anzeigen der Business Partner automatisch ausgeführt.

Löschen: Das Löschen geschieht auf Knopfdruck. Der zu löschende Business Partner wird ermittelt, indem der aktive Reiter der JTabbedPane und darin der selektierte Business Partner in der JTable ermittelt wird. Es erfolgt die Anzeige von Erfolg bzw. Misserfolg des Löschvorgangs.

Verbinden: Das Verbinden wird einmalig nach dem Starten des Frontends durch Knopfdruck ausgeführt. Hiermit werden die einzelnen Services initialisiert.

Ändern: Beim Änderungsvorgang, der ebenfalls auf Knopfdruck ausgeführt wird, wird der selektierte Business Partner genau wie beim Löschen ermittelt. Es öffnet sich eine Änderungsmaske, in der der Benutzer die gewünschten Änderungen vornehmen kann. Nach dem Bestätigen der Änderungen werden die lokal vorliegenden, alten Daten des Business Partners mit den serverseitig im SAP-System vorliegenden Daten des Business Partners verglichen. Ist hier bereits eine Inkonsistenz aufgetreten, so wird der Benutzer darüber informiert und die Änderung wird nicht durchgeführt. Erfolg bzw. Misserfolg werden in der GUI dargestellt.

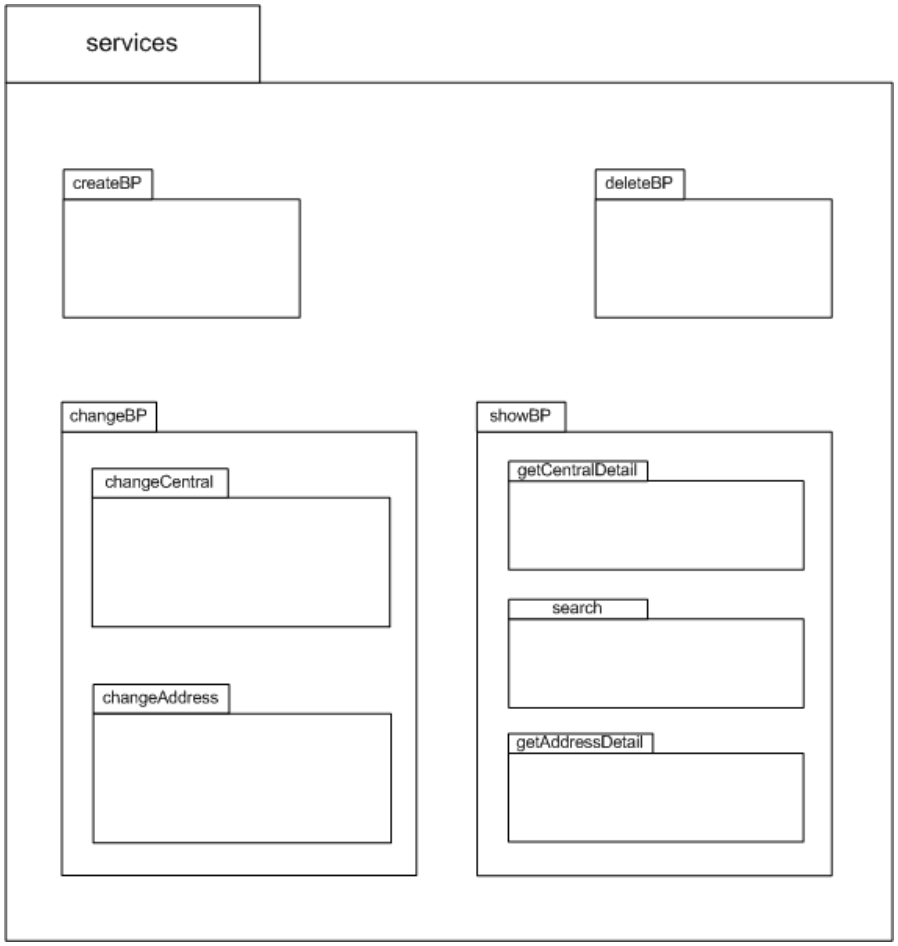


Abbildung 12: Detaillierte Darstellung der Service-Fassade (Webservice-Lösung)

Anlegen: Auch das Anlegen erfolgt auf Knopfdruck. Hierfür öffnet sich eine Maske zum Eintragen der Kontaktdaten. Nach dem Anlegen des Business Partners wird der Erfolg bzw. Misserfolg in der GUI dargestellt.

7.1.3 Anwendungsfälle

Für die in 7.1.2 genannten Möglichkeiten der Interaktion von Anwendern mit dem Frontend bietet es sich an, Anwendungsfälle zu schreiben. Anhand der Anwendungsfälle erhält man einen Überblick über die Funktionalitäten des Frontends. In Anhang A wird mit einem Anwendungsfalldiagramm ein Überblick über die Anwendungsfälle geschaffen und in Anhang B werden die Anwendungsfälle detailliert dargestellt.

7.1.4 Selektion der angezeigten Daten

Für einen Business Partner sieht *SAP-BP* eine Vielzahl an optionalen Informationen vor, die für den Zweck dieser Bachelorarbeit teilweise zu sehr ins Detail gehen. So kann man dem Aufruf zum Anlegen eines neuen Business Partners sehr viele Parameter mitgeben. Würde man alle optionalen Parameter beachten, so hätte man einen Methodenaufruf

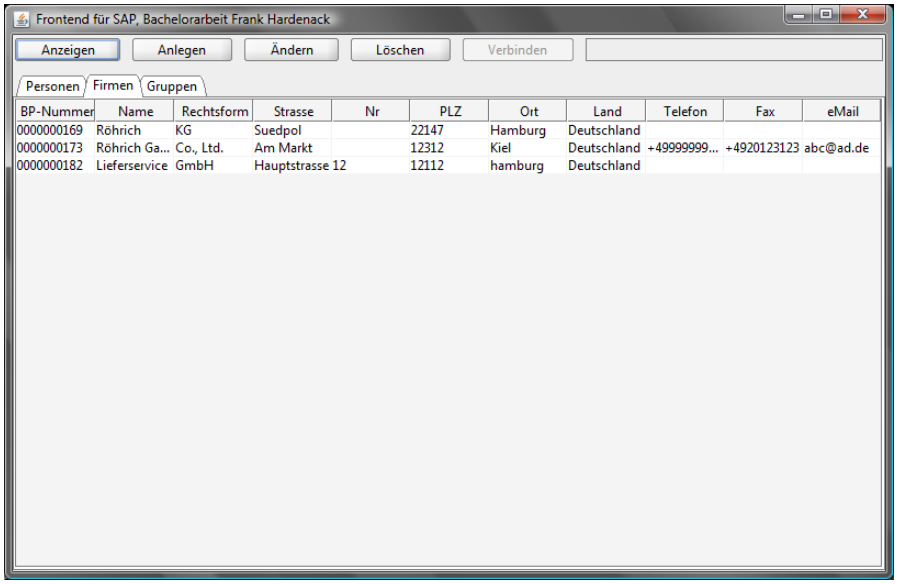


Abbildung 13: Grafische Benutzeroberfläche des Frontends

mit mehr als 40 Übergabeparametern. Um diese Komplexität zu verringern, beschränkt man die verwendeten Informationen nur auf eine Teilmenge an relevanten Kontaktdaten. Durch das Weglassen von nicht relevanten, optionalen Parametern erhält man eine übersichtlichere Struktur und kürzere Methodenaufrufe bei den Webservices. Die Parameter der Aufrufe sind oft spezielle Datentypen, die eine Repräsentation von internen Tabellen darstellen. Eine sinnvolle Beschränkung ist, sich nur an den gängigsten Kommunikationsmitteln zu orientieren und für einen Business Partner eine Postanschrift, sowie jeweils eine Telefonnummer, Faxnummer und eMail-Adresse zu hinterlegen. Bei den kategoriespezifischen Daten für Personen, Firmen und Gruppen bedeutet dies, dass für eine Person Vorname und Nachname, für eine Firma Name und Rechtsform und für eine Gruppe Gruppenname und Typ geführt werden.

7.2 Webservice

Die Struktur und Anzahl der benötigten Services ist durch die gewählten Funktionsbausteine aus Abschnitt 6.3.1 bereits vorgegeben. Ebenso wurde in Abschnitt 7.1.4 festgelegt, welche Daten zu einem Business Partner angezeigt werden sollen, um die *Webservices* durch die vielen möglichen Parameter nicht unnötig kompliziert zu gestalten.
Für die *Webservices* werden um die bestehenden Funktionsbausteine der *BAPI* zu *SAP-BP* Wrapper-Funktionsbausteine gebaut. Dies hat zum einen den Sinn, die optionalen Parameter für diverse nicht verwendete Kontaktdaten aus dem generierten *Webservice* herauszuhalten, zum anderen dient es bei den *Webservices* für den schreibenden Zugriff auf Business Partner der Lösung des Commit-Problems.

7.2.1 Das Commit-Problem

Erzeugt man aus dem Funktionsbaustein einer *BAPI* einen *Webservice* für einen schreibenden Zugriff auf ein SAP-System (Anlegen eines Business Partners oder Ändern von Adressdaten), so muss im Anschluss ein Commitbefehl ausgeführt werden, der serverseitig ein Commit für die Datenbanktransaktion durchführt. Der Commitbefehl kann als eine weitere Methode der Servicedefinition des aus der *BAPI* generierten Webservice hinzugefügt und so später einfach aufgerufen werden. Aufgrund einer bekannten Fehlfunktion wird die Speicherung der Daten nicht durchgeführt. Das hat zur Folge, dass die Transaktion sowohl fehlerlos als auch ergebnislos abgeschlossen wird (SAP AG und SAP Entwickler, 2008).

7.2.2 Die Lösung: Wrapper

Um das bestehende Commit-Problem für Schreibzugriffe zu lösen, werden Wrapperbausteine geschrieben, die den Zugriff auf den entsprechenden Funktionsbaustein der *BAPI* und das anschließende Commit in einem neuen Funktionsbaustein zusammenfassen, aus dem dann später ein *Webservice* generiert wird. Ein Testlauf hat ergeben, dass das nachträgliche Ausführen des Commits in einem Wrapper-Baustein das gewünschte Ergebnis liefert. Für die konkrete Aufgabenstellung bedeutet das, dass für alle schreibenden Zugriffe (Anlegen von Business Partnern, Ändern von Business Partnern) ein Wrapperbaustein geschrieben werden muss.

7.2.3 Das Problem mit den Live-Referenzen

Einige Datentypen (einige komplexen Datentypen, die eine Repräsentation einer Tabelle darstellen) werden bei *Webservice*-Aufrufen als Live-Referenz auf die originale Tabelle vorgehalten. Das führt beim Zugriff auf eine solche Live-Referenz zu Schwierigkeiten. Ein Auslesen bzw. Verändern von Daten in einer mittels Live-Referenz dargestellten Tabelle ist nicht möglich.

7.2.4 Die Lösung: Eigene Datenelemente

Um die aufgrund der Live-Referenz aufgetretenen Probleme beim Hinzufügen von Telefonnummern, Faxnummern oder eMail-Adressen bzw. beim Auslesen von Daten aus der Tabelle mit Rückgabewerten zu umgehen, werden hier eigene Datentypen benötigt, die bereits mit Daten versehen übergeben werden. Dadurch löst man sich von der Notwendigkeit einer Live-Referenz, da die einzufügenden Daten schon vor dem Aufruf feststehen und übermittelt werden können. Die Daten werden dann SAP-seitig aus den eigenen Datentypen ausgelesen und in die Live-Referenz der originalen Tabelle eingefügt (Vgl. Abschnitt 8.3.2, Seite 28). Für die Tabelle mit den Rückgabewerten gilt dies natürlich genau entgegengesetzt. Hier werden die Daten SAP-seitig aus der Live-Referenz in einen festen Exportparameter eingefügt und als ein Ergebnis des Aufrufs mittels Wertübergabe zurückgegeben.

7.2.5 Die einzelnen Services

Zur Erfüllung der Frontend-Funktionalität und nach Prüfung der SAP-seitig gegebenen Funktionsbausteine ergeben sich die in Tabelle 1, Seite 22 aufgeführten Services. Für jeden genannten Service wird eine eigene Servicedefinition und bei Bedarf ein eigener Wrapper erstellt.

Servicename	Beschreibung
createBP	Legt einen neuen Business Partner im System an.
changeAddress	Ändert die Adressdaten eines angegebenen Business Partners.
changeCentral	Ändert zentrale, adressunabhängige Daten eines angegebenen Business Partners.
getAddressDetail	Gibt die Adressdaten eines angegebenen Business Partners zurück.
getCentralDetail	Gibt die adressunabhängigen Daten eines Business Partners zurück.
searchBP	Gibt die eindeutigen Identifikationsnummern von Business Partnern zurück, auf die die Suchkriterien zutreffen. Wird auch zum Anzeigen von Business Partnern verwendet, was der Suche nach einer eindeutigen Identifikationsnummer mit einer Wildcard („*“) gleichkommt.
delBP	Löscht die Daten eines anhand der übergebenen Identifikationsnummer ermittelten Business Partners.

Tabelle 1: Auflistung der einzelnen Services

7.3 SAP-System

Es wurde bereits festgestellt, dass am SAP-System keine grundlegenden Veränderungen vorgenommen werden müssen. Jedoch soll ein eigenes Package für die Servicedefinitionen der erstellten *Webservices* und für die Funktionsbausteine der zu schreibenden Wrapper angelegt werden. Das Package wird gemäß der geltenden Namenskonventionen unter dem Namen *Z_FH_BACHELOR* (Konvention: Z, da ein eigenes Package und kein SAP-Package, Namenskürzel und Packagename) angelegt.

7.3.1 Erweiterung um eigenen Webservice

Das Anlegen, Freigeben und Publizieren der Servicedefinitionen und Webservices aus den Funktionsbausteinen (seien es nun direkt Funktionsbausteine der *BAPI* zu Business Partner oder selbstgeschriebene Wrapper) erfolgt gemäß dem Beispiel aus Kapitel 5. Die Servicedefinitionen zu den einzelnen *Webservices* werden alle im Package *Z_FH_BACHELOR* unter dem Punkt *Service Definitions* abgelegt.

7.3.2 Verwendete Datentypen (SAP)

Die im Abschnitt 7.1.4 eingeführte Beschränkung auf die Kerndaten eines Business Partners zur Verringerung der Komplexität bedeutet, dass bei weitem nicht alle Parameter verwendet werden. Somit wird auch nur ein Teil der möglichen Datentypen verwendet (Vgl. Tabelle 2, Seite 23). Zusätzlich zu den aufgeführten Datentypen kommen beim Ändern von Business Partnern noch spezielle Datentypen zur Signalisierung von Änderungen hinzu. Dafür existiert für alle Datentypen außer für die Rückgabewerte ein gleichnamiger Typ mit der Erweiterung „x“ im Namen. Diese Datentypen sind von der Grundstruktur her gleich aufgebaut wie ihre Originale, nur wird hier bei einer Änderung ausschließlich ein „X“ in das geänderte Feld eingetragen, um dem SAP-System die Änderung zu signalisieren.

Datentyp	Beschreibung
Bapibus1006Address	Komplexer Datentyp, der eine Postanschrift repräsentiert. (Firmen)Namen werden darin noch nicht berücksichtigt.
Bapibus1006Central	Komplexer Datentyp, der kategorieunabhängige Daten aufnimmt (z.B. Archivierungsflags).
Bapibus1006CentralPerson	Komplexer Datentyp, der kategoriebezogene Daten zu einer natürlichen Person aufnimmt.
Bapibus1006CentralOrgan	Komplexer Datentyp, der kategoriebezogene Daten zu einer Firma/Organisation aufnimmt.
Bapibus1006CentralGroup	Komplexer Datentyp, der kategoriebezogene Daten zu einer Gruppe von natürlichen Personen oder Firmen/Organisationen aufnimmt.
ZFhTableOfInts	Eigener komplexer Datentyp zur Aufnahme von 0 bis n Business Partner Identifikationsnummern (10 stellig).
ZFhTableOfMails	Eigener komplexer Datentyp zur Aufnahme von 0 bis n eMail-Adressen.
ZFhTableOfNumbers	Eigener komplexer Datentyp zur Aufnahme von 0 bis n Telefon- oder Faxnummern.
ZFhTableOfBapiret	Eigener komplexer Datentyp zur Aufnahme von 0 bis n Rückgabewerten aus Aufrufen von Funktionsbausteinen.
Bapiret2	Komplexer Datentyp zur Darstellung **eines** Rückgabewerts.

Tabelle 2: Verwendete Datentypen (SAP)

8 Ausführung des Entwurfs der Webservice-Lösung

In diesem Kapitel geht es um die Umsetzung der im Entwurf getroffenen Entscheidungen und um die Realisierung des Zugriffs über *Webservices* auf das SAP System sowie die Programmierung des Frontends.

8.1 Frontend

Für das Frontend sind die GUI, der Controller und die Servicefassaden zu implementieren. Die Servicefassaden greifen auf die aus den WSDL-Dokumenten zu importierenden Java-Klassen zu (Vgl. Abschnitt 8.2, Seite 26 und Anhang C, Seite 57).

8.1.1 Hilfsklassen

Um die Kommunikation der Klassen untereinander zu vereinfachen und die Erweiterbarkeit zu erleichtern, werden Hilfsklassen geschrieben. Die Hilfsklassen werden im Package *util* abgelegt. Nachfolgend werden die Hilfsklassen kurz erläutert.

BPComparator vergleicht zwei übergebene Business Partner auf eventuell aufgetretene Datenänderungen.

BusinessPartner ist eine Klasse zur Repräsentation eines Business Partners. Intern werden die Kontaktdaten in den eigens dafür implementierten eigenen Datentypen gehalten, die vom Aufbau her den SAP-Datentypen nachempfunden sind, aber nur über die in Abschnitt 7.1.4 beschlossenen Informationen verfügen. Es gibt für jeden Datentyp reguläre Getter- und Settermethoden. Für jeden aus dem SAP System ausgelesenen Business Partner wird eine Instanz dieser Hilfsklasse angelegt.

CountryCodes dient der Abbildung von den in SAP verwendeten zwei bis drei Buchstaben langen Ländercodes auf die ausgeschriebenen Ländernamen und umgekehrt. Weitere Länder brauchen nur hier eingepflegt zu werden und können dann im gesamten System verwendet werden.

GroupTypes dient der Abbildung von den in SAP verwendeten Nummerncodes für Gruppentypen auf die ausgeschriebenen Gruppenbezeichnungen und umgekehrt. Werden Gruppen hinzugefügt, so brauchen sie nur in dieser Hilfsklasse nachgetragen zu werden.

LegalForms dient der Abbildung von den in SAP verwendeten Nummerncodes für die Rechtsform einer Firma auf die ausgeschriebene Rechtsform (z.B. KG, GmbH) und umgekehrt.

Für die Hilfsklassen CountryCodes, GroupTypes und LegalForms wird zudem eine Methode implementiert, die die Langformen der gemappten Daten als String-Array zurückgibt, um beispielsweise eine JComboBox damit zu füllen.

8.1.2 Eigene Datentypen

Um die Wiederverwendbarkeit einzelner Komponenten im Speziellen bei der *RFC*-Lösung zu gewährleisten, werden für das Frontend eigene Datentypen implementiert, die eine Abbildung der SAP-seitigen Datentypen (Vgl. Abschnitt 7.3.2, Seite 23) darstellen, deren Inhalt allerdings auf die in Abschnitt 7.1.4 aufgeführten Daten beschränkt wurde. Damit wird auf eine Verwendung der aus den Informationen der WSDL-Dokumente generierten Datentypen verzichtet.

8.1.3 Interfaces

Um die Austauschbarkeit der einzelnen Komponenten zu vereinfachen, werden alle Methoden, die von außen (sprich von anderen Klassen) zugreifbar sein sollen, durch das Implementieren eines Interfaces vorgeschrieben. Dadurch ergeben sich Interfaces für die GUI, den Controller und die obersten Servicefassade (die ja die Schnittstelle zu den Services und somit zum Modell darstellt). Ein Codelisting der Interfaces findet sich auf der beigelegten CD (*Listings⇒Java⇒Frontend_Interfaces*).

8.1.4 Swing und Threadsicherheit

Dadurch, dass *Swing* nicht threadsicher ist und alle Aktionen von *Swing* in dem *Event Dispatcher Thread (EDT)* ausgeführt werden, kann es zu Wechselwirkungen mit anderen Threads und eventuell auch zu Blockaden von Swingkomponenten kommen. Ein gutes Beispiel dafür ist die Verwendung einer Progress-Bar zur Anzeige des Fortschritts bei rechenintensiven, lange andauernden Operationen. Die Initialisierung der einzelnen Services ist ein rechenintensiver Ablauf, der die Progress-Bar blockierte bis die Initialisierung der Services abgeschlossen war. Durch diese Blockade wurde kein Fortschritt angezeigt, was den Grundgedanken des sichtbaren Fortschritts untergräbt. Die Progress-Bar wurde in eine eigene, von JDialog abgeleitete Klasse ausgelagert und wird bei Bedarf in einem eigenen Thread ausgeführt. Wird nun eine Progress-Bar benötigt, so findet die Initialisierung der Progress-Bar in einem eigenen Thread statt (Der genauere Aufbau und die Benutzung der Progress-Bar wird in Abschnitt 8.1.6 beschrieben).

8.1.5 Dialogklassen

Das Anlegen und Ändern von Business Partnern sowie die Darstellung von Fehlern und der Progress-Bar sind Interaktionen vom Benutzer mit dem Frontend, die voraussetzen, dass der Rest der Applikation für den Zeitraum der Darstellung und Bearbeitung der vorliegenden Aktion in den Hintergrund tritt. Dafür wurden von JDialog abgeleitete Klassen für die jeweilige Verwendung geschrieben, da diese durch die Angabe eines *Modality Types*, in diesem Fall *Application Modal*, für ihre Laufzeit die aufrufende Anwendung und deren Funktionen für Benutzereingaben sperren. So kann sichergestellt werden, dass der Benutzer keine unerwarteten Eingaben bzw. Aktionen tätigen kann, während einer der oben genannten Vorgänge aktiv ist.

8.1.6 Verwendung der Progress-Bar

Die Progress-Bar wird in eine eigene, von JDialog abgeleitete Klasse ausgelagert. Die Klasse wird anfangs in der GUI des Frontends einmalig instanziiert und kann über verfügbare Methoden zurückgesetzt und erneut benutzt werden. Wird eine Progress-Bar benötigt, so wird sie vom Controller initialisiert und sichtbar gesetzt. Die Initialisierung findet in einem eigenen Thread statt, ebenso wie der rechenintensive Vorgang, für den eine Progress-Bar angezeigt werden soll, in einen eigenen Thread ausgelagert wird. Bekommt der Controller als Observer auf die Servicefassade die Mitteilung, dass sich beim Observable etwas geändert hat, so wird mit *SwingUtilities.invokeLater()* die Aktualisierung der Progress-Bar in den *EDT* von *Swing* mit eingereiht.

8.1.7 Fehlerbehandlung

Ein wichtiger Punkt bei der Programmierung ist die Fehlerbehandlung. Die Entscheidung fällt in diesem Fall darauf, dass die ServiceFassaden Exceptions werfen können, die zum einen von den Services selbst kommen, zum anderen aber auch selbst ausgelöst werden können. Die auftretenden Exceptions werden bis in den Controller durchgereicht und dort verarbeitet. Es wird also im Controller die Darstellung der Exceptions vorgenommen (über das Setzen der Exception in der GUI) und darüber entschieden, wie weiter verfahren werden soll. Die Darstellung der aufgetretenen Fehler erfolgt über eigene Dialoge, wie in Abschnitt 8.1.5 beschrieben.

8.1.8 Export als .jar

Um das entwickelte Frontend auf jedem beliebigen PC einsetzen zu können, wird es als ausführbares .jar-Archiv exportiert. Dank der Plattformunabhängigkeit von Java kann das Frontend nun unter allen gängigen Betriebssystemen eingesetzt werden. Eine komfortable Möglichkeit ein .jar-Archiv zu erstellen ist die Nutzung des Eclipse-Plugins *FatJar*. Mit wenigen Befehlen erhält man ein lauffähiges .jar-Archiv. Ein ausführbares .jar-Archiv findet sich auf der beigelegten CD (*Jars⇒SAPBacWS.jar*).

8.2 Webservice

Um für das Frontend verwertbare Java-Klassen zu erhalten, werden die WSDL-Dokumente, die Teil der Servicedefinitionen der erstellten *Webservices* sind, mit *wsimport* konsumiert. Beim Konsumieren der WSDL-Dokumente trat ein unerwarteter Fehler auf.

8.2.1 Verschachtelte Packages und wsimport

Um eine bessere Struktur bei den einzelnen Services zu erreichen, sollten diese alle eigene Packages unterhalb des Packages *Services* erhalten (Vgl. Abbildung 12, Seite 19). Durch den Parameter „-p“ bei der Nutzung von wsimport kann man einen Packagenamen angeben (*wsimport -keep -p <Paketname> <Pfad zum WSDL-Dokument>*). Bei dem Versuch die Services nun jeweils in eigene Packages unterhalb von *Services*

zu importieren, traten im späteren Verlauf Fehler bei der Initialisierung der Services auf. Eine andere Möglichkeit verschachtelte Packages für die generierten Artefakte aus den WSDL-Dokumenten der Servicedefinitionen zu nutzen ist die separate Angabe eines Verzeichnisses für den Sourcefolder mit dem Parameter „-d“ und die anschließende Angabe des Packages wie gewohnt durch Punktnotation (*wsimport -d <Source-Directory> -keep -p <Paketname> <Pfad zum WSDL-Dokument>*).

8.2.2 Live-Referenzen und lesender Zugriff

Entgegen der Annahme, dass die Live-Referenzen nur bei schreibenden Zugriffen ein Problem darstellen war das Auslesen der eindeutigen Identifikationsnummern beim Anzeigen der Business Partner aus einem Datentyp mit einer Live-Referenz nicht möglich. Die Liste mit den Identifikationsnummern blieb trotz vorhandener Business Partner immer leer. Eine Lösung für dieses Problem ist das Anlegen von einem eigenen Datenelement und einem eigenen Tabellentyp, um die Liste mit den Nummern nicht als Live-Referenz sondern als Rückgabeparameter zu erhalten (Vgl. Abschnitt 8.3.2, Seite 28). Damit werden für alle Parameter der *Webservice*-Aufrufe, die eine Live-Referenz benutzen, eigene Datenelemente und Tabellentypen angelegt. Live-Referenzen sind SAP-seitig bei den Funktionsbausteinen als *Tables*-Parameter dargestellt.

8.3 SAP-System

Bei der Ausführung des Entwurfs geht es auf der Seite des SAP-Systems darum, die Wrapper und eigenen Datentypen anzulegen, um zum einen das Commit-Problem (Vgl. Abschnitt 7.2.1) und das Live-Referenz-Problem (Vgl. Abschnitt 7.2.3 und 8.2.2) zu umgehen und zum anderen die Komplexität der *Webservice*-Aufrufe zu verringern. Zur Verringerung der Komplexität der Aufrufe werden optionale, nicht benötigte Parameter der *BAPI*-Funktionsbausteine in den Wrappern nicht als Parameter eingetragen. Aus den geschriebenen Wrapper-Funktionsbausteinen werden die Servicedefinitionen und die WSDL-Dukumente generiert, die zur Generierung der Java Klassen für den Service benötigt werden (Vgl. Abschnitt 8.2).

8.3.1 Wrapper anlegen

Bereits in Abschnitt 7.2.1 wurde festgestellt, dass für die Schreibzugriffe auf SAP über *BAPI*-Funktionsbausteine passende Wrapper geschrieben werden müssen. Hierfür wird jeweils ein RFC-fähiger Funktionsbaustein im Package *Z_FH_BACHELOR* angelegt. Im Wrapper-Baustein wird der Funktionsbaustein der *BAPI* aufgerufen und die gewünschten Aufrufparameter werden durchgereicht. Im Anschluss wird der Funktionsbaustein *BAPI_TRANSACTION_COMMIT* aufgerufen, um die Änderungen zu speichern und die Transaktion abzuschliessen. Durch das Commit steht ein weiterer optionaler Rückgabeparameter zur Verfügung, der angibt, ob das Commit erfolgreich war. Auch für rein lesende Zugriffe werden Wrapper angelegt, um nicht benötigte Parameter herauszufiltern. Im Fall eines lesenden Zugriffs fällt das Commit selbstverständlich weg. Das direkte Ausführen von Funktionsbausteinen im SAP-System ermöglicht die Überprüfung der

Funktion eines Wrappers. Ein Codelisting der Wrapper findet sich auf der beigelegten CD (*Listings⇒Abap*).

8.3.2 Eigene Datenelemente und Tabellentypen anlegen

Nachdem in Abschnitt 7.2.3 das Problem mit den Live-Referenzen erkannt wurde, werden nun die eigenen Datenelemente und Tabellentypen angelegt, um dieses Problem zu umgehen. Hierfür wird zuerst ein neues Datenelement angelegt (Vgl. Abbildung 14, Seite 28). Beim Anlegen eines Datenelements kann man den Ausgangsdatentyp und weitere Eigenschaften wie zum Beispiel die maximale Länge angeben. Aus diesem Datentyp lässt sich nun ein eigener Tabellentyp bauen, der einer Menge von 0 bis n Datenelementen des angegebenen Zeilentyps entspricht (Vgl. Abbildung 15, Seite 29). Dieser Vorgang wird für alle benötigten eigenen Datenelemente und Tabellentypen wiederholt.

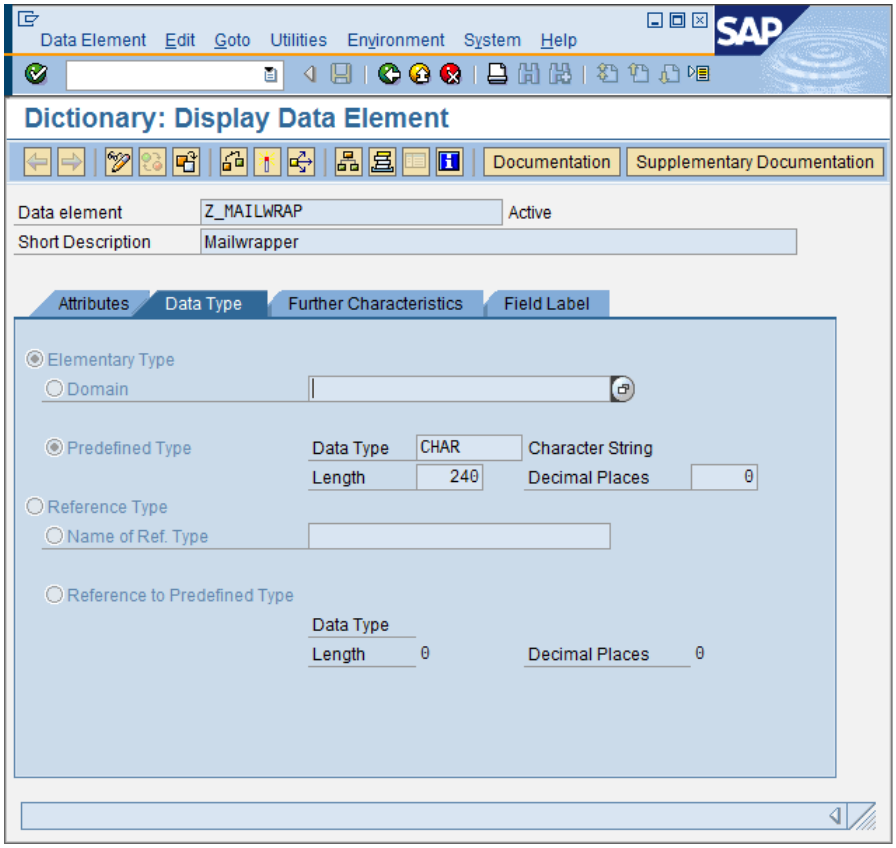


Abbildung 14: Anlegen eines eigenen Datenelements

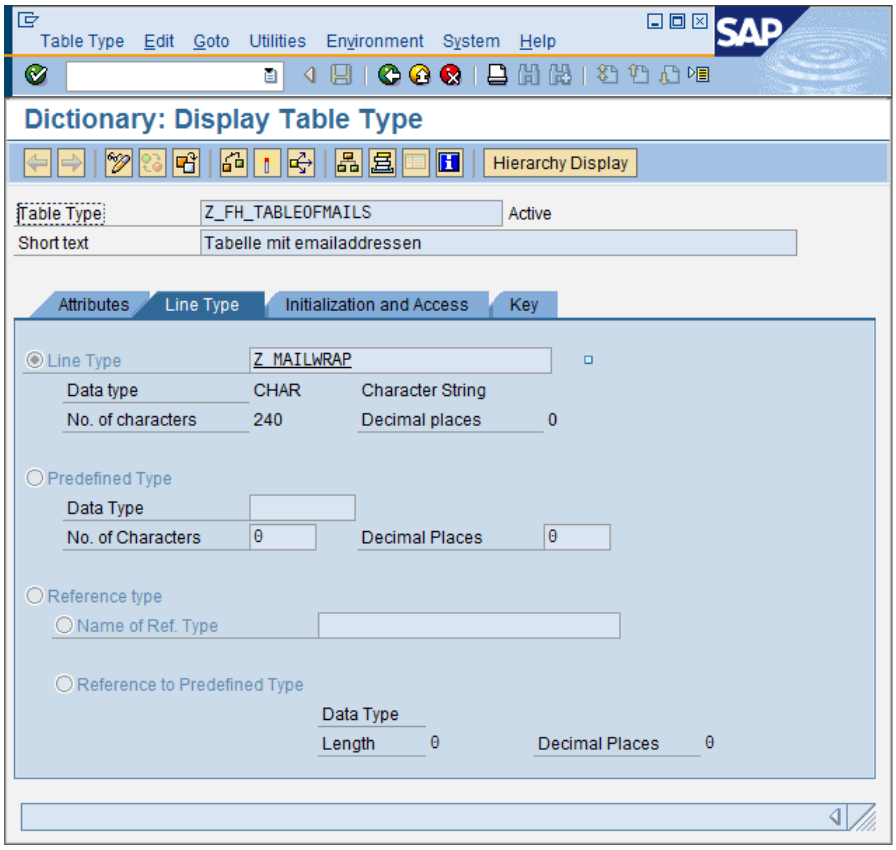


Abbildung 15: Anlegen eines eigenen Tabellentyps

9 RFC in SAP

Bei *Remote Function Call (RFC)* handelt es sich um eine proprietären Ansatz von SAP auf Basis des *Remote Procedure Call (RPC)*. *RFC* ermöglicht das Aufrufen von Funktionen auf entfernten Systemen. Die Kommunikation funktioniert sowohl zwischen zwei *SAP-AS* als auch zwischen einem *SAP-AS* und einem beliebigen anderen System, das den *RFC* unterstützt. Hierfür werden von SAP diverse Connectoren zur Verfügung gestellt und stehen ausschließlich SAP-Kunden zum Download bereit. Die Aufrufschnittstelle für *RFC* in SAP ist zweigeteilt. Zum einen gibt es eine Schnittstelle für ABAP-Programme und zum anderen eine Schnittstelle für nicht-ABAP-Programme. In beiden Fällen findet die Übertragung über das *Common Programming Interface for Communication (CPI-C)* oder über *TCP/IP* statt. (Keller und Krüger, 2007, Seite 921-948), (Wikipedia/RFC, 2008)

9.1 Varianten des RFC

Bei *RFC* stehen vier unterschiedliche Varianten zur Verfügung, die in ihren jeweiligen Eigenschaften nachfolgend kurz beschrieben werden.

Synchroner RFC

Der *synchrone RFC* führt die Funktionsaufrufe auf Basis der synchronen Kommunikation durch. Dies bedeutet, dass alle am Aufruf beteiligten Systeme (aufrufendes System und Zielsystem) beim Aufruf verfügbar sein müssen. Ist das Zielsystem beim Aufruf nicht verfügbar, so schlägt der Aufruf fehl und wird vom aufrufenden System nicht wiederholt. Die Art der Ausführung des Aufrufs ist „At most Once“.

Asynchroner RFC

Der *asynchrone RFC* ist, anders als der Name es vermuten lässt, nur bedingt eine Art der asynchronen Kommunikation, da nicht alle Kriterien der asynchronen Kommunikation erfüllt sind. So muss das Zielsystem zur Zeit des entfernten Funktionsaufrufs verfügbar sein, um den Aufruf entgegen zu nehmen. Als asynchron lässt sich in diesem Zusammenhang nur das Arbeitsverhalten des aufrufenden Systems bezeichnen, da nicht explizit auf die Antwort des Zielsystems gewartet werden muss. Auch hier ist die Ausführungseigenschaft „At most Once“.

Transaktionaler RFC

Der *transaktionale RFC* stellt eine echte asynchrone Kommunikation dar. Das entfernte System muss zur Zeit des Aufrufs nicht zwingend verfügbar sein. Die Ausführungseigenschaft ist in diesem Fall „Exactly Once“.

Queued RFC

Der *queued RFC* ist eine erweiterte Form des *transaktionalen RFC*, bei der die Reihenfolge der Aufrufe erhalten bleibt. Die Ausführungseigenschaft ist in diesem Fall „Exactly Once in Order“.

9.2 RFC mit einer externen Java-Anwendung

Um eine Java-Anwendung über *RFC* an ein SAP-System anzubinden, benötigt man den von SAP zur Verfügung gestellten *Java Connector (JCo)*, den man sich als SAP-Kunde unter *http://service.sap.com/connectors* nach einer Anmeldung und Angabe des Entwicklerschlüssels herunterladen kann. In diesem Fall wird der *JCo* als eigenständige Komponente verwendet und kommuniziert nicht direkt mit dem *SAP-AS*, sondern mit der SAP-seitig bereitgestellten Schnittstelle für Fremdsysteme. Verbindungen von externen Anwendungen zu einem SAP-System sind auf zwei unterschiedliche Arten möglich.

- direkte Verbindung
- Connection Pools

Die *direkte Verbindung* kann nach dem Öffnen beliebig lange geöffnet bleiben und muss explizit geschlossen werden, während bei der Verwendung von *Connection Pools* der Pool die Verwaltung der Verbindungen sowie deren Aufbau vornimmt. Wird eine Verbindung benötigt, so muss sie nur noch beim *Connection Pool* angefordert und nach Abschluss der Arbeit wieder freigegeben werden. Eine Kombination beider Lösungen ist möglich aber nur bedingt ratsam. Die Verwendung eines *Connection Pools* ist empfehlenswert, da für die verwalteten Verbindungen nur anfangs eine einmalige Anmeldung erfolgen muss und anschließend die Verwaltung beim *Connection Pool* liegt.

10 RFC zwischen Java und SAP - Ein Beispiel

Bevor mit der Planung für die *RFC*-Lösung begonnen wird, soll an einem kleinen Beispiel die Funktionsweise von *RFC* in SAP getestet werden. Dieses erste Beispiel dient auch der detaillierten Beschreibung der Vorgehensweise beim Verwenden von *RFC* in SAP, sowie einem ersten Durchstich zur Feststellung der technischen Machbarkeit (Royce, 1998), (Royce, 1970). In diesem Beispiel werden *Connection Pools* verwendet, um die Verbindung einfacher verwalten zu können.

10.1 Anlegen eines RFC-fähigen Funktionsbausteins

Der erste Schritt ist das Anlegen eines *RFC*-fähigen Funktionsbausteins. Hierfür kann der bereits erstellte Funktionsbaustein zur Wurzelberechnung aus Abschnitt 5.2 auf Seite 7 genutzt werden.

10.2 Nutzen des RFC in einer Java-Anwendung

Um den *RFC* in einer Java-Anwendung nutzen zu können, bedarf es einiger Vorbereitungen. Zuerst muss man sich von der SAP Webseite den *JCo* herunterladen und installieren. Bei einem Windows-Betriebssystem fügt man dazu den Pfad zum entpackten *JCo* der Umgebungsvariablen *PATH* hinzu. Um die Funktionalität des *JCo* nun auch in einem Java-Projekt nutzen zu können, bindet man den als .jar-Datei vorliegenden *JCo* als externes Archiv in das Projekt mit ein (Eclipse: *Project Properties* $\Rightarrow$ *Build Path* $\Rightarrow$ *Add external Archives*). Nun können die Klassen des *JCo* im eigenen Java-Projekt verwendet werden.

10.2.1 Anlegen eines Connection Pools und Erstellen der Verbindungen

Zur Erzeugung eines *Connection Pools* muss zuerst eine *Property-Datei* angelegt werden. Bei der Vergabe eines Namens empfiehlt sich ein selbsterklärender Name. Die vorgegebene Bezeichnungen der Felder einer *Property-Datei* für den *RFC* stehen als statische Strings in der Klasse *DestinationDataProvider* durch den eingebundenen *JCo* zur Verfügung. Für die Erzeugung einer *Property-Datei* zur Nutzung von *Connection Pools* sind nicht alle Felder relevant. Für die angestrebte Nutzung reicht das Angeben folgender Eigenschaften aus:

JCO_ASHOST: Adresse des *SAP-AS*

JCO_SYSNO: SAP Systemnummer

JCO_CLIENT: Klientennummer

JCO_USER: Benutzer

JCO_PASSWD: Passwort

JCO_LANG: Anmeldesprache

JCO_POOL_CAPACITY: Anzahl maximaler ruhender Verbindungen (Idle-Connections) des Pools

JCO_PEAK_LIMIT: Anzahl maximaler gleichzeitiger Verbindungen des Pools zum Ziel

Es stehen neben den genannten Eigenschaften noch diverse weitere Eigenschaften zur Verfügung, um beispielsweise eine sichere Übertragung zu gewährleisten oder eine Authentifizierung durch ein X.509-Zertifikat vorzunehmen. Durch die Verwendung einer VPN-Verbindung mit Benutzer und Passwort sind diese Eigenschaften vernachlässigbar. Hat man alle oben genannten Eigenschaften mit konkreten Werten versehen, so schreibt man den Inhalt des Properties-Objekts in eine Datei oder einen Stream. Diese Erzeugung ist, sofern man in eine Datei schreibt, in der Regel einmalig und muss nur wiederholt werden, wenn sich an den festgeschriebenen Eigenschaften etwas geändert hat.
Ist das Eintragen der Eigenschaften beendet, wird der *Connection Pool* durch das Aufrufen der Methode *getDestination(String name)* der Klasse *JCoDestinationManager* erzeugt.

10.2.2 Aufrufen des entfernten Funktionsbausteins

Ist der *Connection Pool* erzeugt und die Verbindung hergestellt, so kann man über die Methode *getRepository()*, aufgerufen auf das Objekt der Verbindung zum Zielsystem, eine Instanz der Klasse *JCoRepository* mit Metadaten zum Zielsystem anfordern. Die Metadaten enthalten unter anderem eine Auflistung aller remotefähigen Funktionsbausteine des Zielsystems. Mittels der Funktion *getFunction(String funktionsname)* kann unter Angabe des Funktionsbausteinnamens im SAP-System ein remotefähiger Funktionsbaustein angesprochen werden. Man erhält beim Aufruf von *getFunction(String funktionsname)* eine Instanz der Klasse *JCoFunction* mit den Parameterlisten zum aufgerufenen Funktionsbaustein oder im Fehlerfall *null* zurück.
Der nächste Schritt ist das Setzen der entsprechenden Parameter in den Parameterlisten des Funktionsobjekts zur Vorbereitung der Ausführung. Sind die Parameter gesetzt, so wird der Aufruf des entfernten Funktionsbausteins durchgeführt. Zurückgegebene Ergebnisse können aus den Parameterlisten ausgelesen und weiterverarbeitet werden.

10.3 Ergebnis und Ausblick

Die Realisierung eines ersten *RFC* auf einen entfernten Funktionsbaustein eines SAP-Systems funktionierte trotz mangelhafter Dokumentation zum *JCo* einwandfrei und die Implementierung auf Java-Seite war verhältnismäßig einfach durchzuführen. Das Arbeiten mit komplexen Strukturen wie zum Beispiel Adressdaten (Vgl. Tabelle 2, Seite 23) kann gegebenenfalls zu Schwierigkeiten führen. Ein Codelisting des behandelten Beispiels befindet sich auf der beigelegten CD (*Listings⇒Java⇒Beispiel_RFC*).

11 Planung der RFC-Lösung

Nach einem ersten erfolgreichen *RFC*-Aufruf (Vgl. Kapitel 10, Seite 32) kann nun mit der Planung der *RFC*-Lösung begonnen werden. In vielen Punkten der Planung der *RFC*-Lösung kann auf die Planung der *Webservice*-Lösung (Kapitel 6) und dort getroffene Entscheidungen, speziell in Bezug auf das Frontend, zurückgegriffen werden.

11.1 Frontend

Für die Entwicklung des Frontends für die *RFC*-Lösung sollen wesentliche Teile des Frontends für die *Webservice*-Lösung wiederverwendet werden (GUI und Controller). Die Entwicklung wird aus diesem Grund und durch die notwendige Nutzung des von SAP zur Verfügung gestellten *JCo* zum Ansprechen *RFC*-fähiger Funktionsbausteine ebenfalls in Java stattfinden.

11.2 RFC

Da ein Vergleich zwischen der *Webservice*- und der *RFC*-Lösung angestrebt wird, sollten die Voraussetzungen möglichst identisch sein. Um dies zu erreichen wird *synchroner, zustandsloser RFC* verwendet, da die *Webservice*-Kommunikation ebenfalls synchron und zustandslos arbeitet. Auch die bereits entworfenen Wrapper sollen aus diesem Grund und zur Aufwandsreduzierung wiederverwendet werden. Um nicht selbst die Verwaltung der offenen Verbindungen vornehmen zu müssen, werden *Connection Pools* verwendet.

11.3 SAP-System

Der Planungsaufwand für SAP-seitige Veränderungen ist für die angestrebte *RFC*-Lösung minimal, da vollständig auf die Ergebnisse der Planung der *Webservice*-Lösung (Vgl. Abschnitt 6.3, Seite 11) zurückgegriffen wird. So sind die für die einzelnen Frontend-Funktionalitäten zu verwendenden Funktionsbausteine bereits bekannt und auch eine Einschränkung der zu verwendenden Daten inklusive dem Entwurf von Wrappern um die Funktionsbausteine wurde bereits durchgeführt. Der Aspekt der Sicherheit bei der Übertragung ist auch bei dieser Lösung zu vernachlässigen, da die Übertragung ebenfalls im lokalen Firmennetzwerk oder über eine VPN-Verbindung stattfindet.

12 Entwurf der RFC-Lösung

Da in der Planung bereits festgelegt wurde, einen Großteil der *Webservice*-Lösung zu übernehmen, sollen in diesem Kapitel die vorzunehmenden Veränderungen und Anpassungen näher beschrieben werden. Wesentliche Entscheidungen zur Struktur und zum Design des Frontends wurden bereits in Abschnitt 7.1 auf Seite 16 getroffen und werden hier nicht erneut explizit dargestellt.

12.1 Frontend

In Abschnitt 7.1 wurden bereits Entscheidungen über die softwaretechnische Struktur, das Design der Oberfläche und die anzuzeigenden Daten getroffen. Ebenso wurden die Anwendungsfälle zu den Anforderungen entworfen (Vgl. auch Anhang A und B, ab Seite 49). Durch die Wiederverwendung von GUI und Controller der *Webservice*-Lösung muss für das *RFC*-Frontend nur die Servicefassade, die die eigentliche Kommunikation und damit die entfernten Funktionsaufrufe auf dem SAP-System vornimmt, geändert werden.

Um die Wiederverwendbarkeit von einzelnen Komponenten des Frontends zu vereinfachen, wurden bereits in Abschnitt 8.1.3 auf Seite 25 Interfaces definiert. Die hier zu entwerfende neue Servicefassade muss also das Interface *IServiceFassade* implementieren. Die einzelnen Funktionalitäten des Frontends sollen, wie auch schon in der Servicefassade bei der *Webservice*-Lösung, möglichst getrennt voneinander vorliegen, um eine lose Kopplung zu erreichen. Komplexere Funktionalitäten, die aus mehreren entfernten Funktionsaufrufen bestehen, werden durch eine untergeordnete Fassade und somit über eine klar definierte Schnittstelle zugreifbar gemacht, um die Wartbarkeit zu verbessern (Vgl. Abbildung 16, Seite 35). Für die einzelnen Funktionalitäten ergeben sich für die *RFC*-Lösung die in Anhang D.2 ab Seite 62 aufgeführten Sequenzdiagramme.

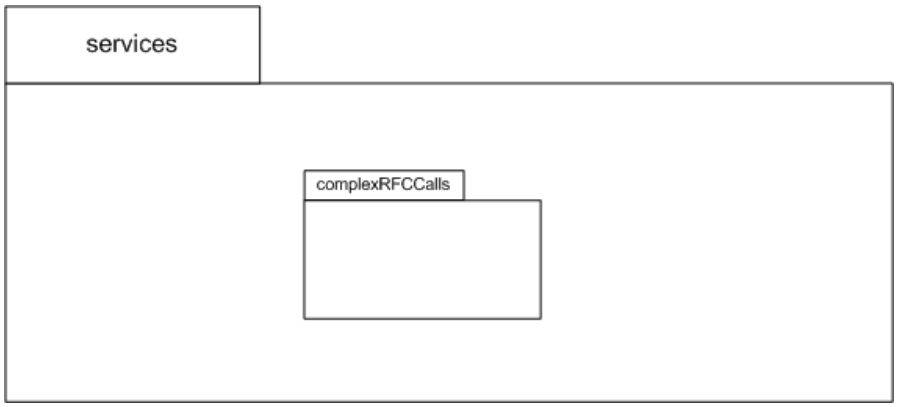


Abbildung 16: Detaillierte Darstellung der Service-Fassade (RFC-Lösung)

12.1.1 Verwendung der Hilfsklassen und eigenen Datentypen

Die in Abschnitt 8.1.1 beschriebenen Hilfsklassen aus der *Webservice*-Lösung finden in der *RFC*-Lösung erneut Verwendung. Auch die eigenen Datentypen werden im neuen Frontend wiederverwendet.

12.2 RFC

Durch die Vorarbeit in Abschnitt 6.3.1 auf Seite 12 sind die einzelnen Funktionsbausteine für den *RFC* bereits bekannt. Die Benutzung eines *Connection Pools* wie in dem Minimalbeispiel in Kapitel 10 kann durch eine Factory-Klasse ergänzt werden, um die Verwaltung des Pools und der einzelnen *JCoFunction*-Objekte zentral im Frontend vornehmen zu können.

12.3 SAP-System

Da die Wrapper um die benötigten Funktionsbausteine von *SAP-BP* in Abschnitt 8.3.1 zur Realisierung der *Webservice*-Lösung bereits angelegt wurden, braucht für die *RFC*-Lösung am SAP-System nichts weiter verändert zu werden. Die bestehenden Wrapper (*RFC*-fähige Funktionsbausteine) können direkt für die *RFC*-Lösung wiederverwendet werden.

13 Ausführung der RFC-Lösung

In diesem Kapitel wird auf die Umsetzung der in Kapitel 12 getroffenen Entscheidungen eingegangen. Es muss die Servicefassade für den *RFC*-Zugriff implementiert werden und aus den bereits entwickelten Komponenten GUI und Controller zusammen mit der neuen Servicefassade (Im weiteren Verlauf: RFCFassade) die *RFC*-Lösung erstellt werden. Auch bei der Ausführung des Entwurfs der *RFC*-Lösung gibt es große Überschneidungen mit der Ausführung in Kapitel 8. Dies führt dazu, dass in diesem Kapitel nicht auf alle Punkte der Entwicklung des Frontends erneut eingegangen wird.

13.1 Frontend

Nach den in Abschnitt 10.2 beschriebenen Vorbereitungen bei der Erstellung des Java-Projekts muss nur die RFCFassade zum Ausführen des *RFC*-Aufrufs entwickelt werden, da die anderen Komponenten aus der *Webservice*-Lösung wiederverwendet werden. Bei der Implementierung der RFCFassade wird das Interface für die Serviceschicht (Vgl. Abschnitt 8.1.3, Seite 25) verwendet. Für die Verwaltung der *RFC*-Funktionsobjekte wird eine eigene Klasse implementiert. (Vgl. Diagramme in Anhang C, Seite 57).

13.1.1 Function Factory

Für die Erstellung der *JCoFunction*-Objekte für den *RFC*-Aufruf wird eine Factory-Klasse implementiert, die die Instanziierung der einzelnen *RFC*-Funktionsobjekte vornimmt. Auf die *JCoFunction*-Objekte der jeweiligen Funktion kann über Getter zugegriffen werden. Für jeden verwendeten Funktionsbaustein des SAP-Systems (die entworfenen Wrapper) gibt es jeweils ein entsprechendes *JCoFunction*-Objekt. Neben der Verwaltung und Instanziierung der Funktionsobjekte übernimmt die Function Factory auch die Initialisierung der Kommunikation und des *Connection Pools* sowie den Aufbau der Verbindung zum SAP-System. Um zur Laufzeit immer mit der gleichen Verbindung zu arbeiten, wird die Function Factory wegen mehrfacher Verwendung als Singleton implementiert. Um weitere Funktionsbausteine im Frontend zur Verwendung mit aufzunehmen, braucht nur noch ein passender Getter in der Function Factory eingefügt zu werden.

13.1.2 Export als .jar

Wie schon bei der *Webservice*-Lösung (Vgl. Abschnitt 8.1.8, Seite 26) wird das fertige Frontend der *RFC*-Lösung ebenfalls mit dem Eclipse-Plugin *FatJar* als ausführbares .jar-Archiv exportiert. Ein ausführbares .jar-Archiv findet sich auf der beigelegten CD (*Jars⇒SAPBacRFC.jar*).

13.2 RFC

Um einen *RFC* auf einen Funktionsbaustein auszuführen, kann man sich nun der Function Factory bedienen. Über den entsprechenden Getter kann das benötigte *JCoFunction*-Objekt erzeugt werden. Nach dem Setzen der Importparameter kann der entfernte Aufruf durchgeführt und im Anschluss die Liste der Exportparameter ausgelesen werden.

14 Bewertung / Fazit

Für eine aussagekräftige Bewertung beider Einzellösungen ist es wichtig, geeignete Bewertungskriterien zu finden. Dafür muss festgelegt werden, welche Faktoren beider Lösungen überhaupt bewertbar und damit auch miteinander vergleichbar sind. Nach längerem Abwägen habe ich mich entschlossen, die beiden Lösungen nach folgenden Kriterien zu bewerten:

- Flexibilität
- Entwicklungsaufwand
- Wartbarkeit / Codequalität
- Erweiterbarkeit / Anpassbarkeit
- Performance
- Sicherheit

Die oben genannten Kriterien gehen zu gleichen Teilen in die Bewertung ein und werden nachfolgend näher betrachtet.

Flexibilität: Die hier zu bewertende Flexibilität zielt auf eine möglichst flexible Verwendung der einzelnen Lösungen ab. Ausschlaggebend ist die mögliche Verwendung unterschiedlicher Programmiersprachen und damit die oftmals leichtere Integration von Aufrufen auf ein SAP-System in bestehende Softwarearchitekturen.

Entwicklungsaufwand: Die Betrachtung des Entwicklungsaufwands bezieht sich sowohl auf den reinen Entwicklungsaufwand für die Anwendung und das SAP-System als auch auf die zu treffenden Vorbereitungen zur Entwicklung.

Wartbarkeit und Codequalität: Bei der Wartbarkeit geht es im Wesentlichen um die Komplexität des Codes, wie ihn der Entwickler zur Wartung vor sich hat, oder bei Neuentwicklungen selbst schreibt. Es wird ein Blick auf die Codequalität geworfen, die sich durch Verwendung spezieller Datentypen, Anzahl der Codezeilen und die Komplexität von Aufrufen definiert.

Erweiterbarkeit und Anpassbarkeit: Für die zu bewertende Erweiterbarkeit ist zu beachten, mit welchem Aufwand ein neuer Service einer bestehenden Anwendung hinzugefügt werden kann. Dabei ist sowohl der Aufwand am SAP-System als auch der Aufwand bei der Entwicklung der eigentlichen Anwendung zu beachten. Neben dem Hinzufügen eines neuen Services soll auch die Anpassung eines bestehenden Services betrachtet werden.

Performance: Zur Bewertung der Performance sollte ursprünglich das Tool *JMeter* zum Einsatz kommen, was aber keinen Sinn macht, da eine Performanceprüfung mit *JMeter* für den proprietären *RFC*-Ansatz von SAP nicht möglich ist. Um aber vergleichbare Ergebnisse zu erhalten, sollen beide Lösungen auf gleiche Weise

einer Performanceprüfung unterzogen werden. Die Entscheidung ist darauf gefallen, selbst einen kleinen Performancetest mittels einer Stoppuhrfunktion in Java zu implementieren. Die Performance soll zum einen für den Verbindungsaufbau und zum anderen für den Aufruf eines entfernten Funktionsbausteins gemessen werden. Für ein aussagekräftiges Ergebnis wird mit beiden Technologien (*Webservices* und *RFC*) auf den gleichen Funktionsbaustein im SAP-System zugegriffen und jeweils die benötigte Zeit für die Initialisierung und die Ausführung eines Aufrufs gemessen. Da einzelne Messergebnisse keine hohe Aussagekraft haben, wird der Mittelwert aus jeweils 1000 Messungen ermittelt. Diagramme zur Performanceprüfung sind in Anhang F ab Seite 66 dargestellt.

Sicherheit: Der Aspekt der Sicherheit wurde in dieser Bachelorarbeit bisher vernachlässigt, soll aber zur Bewertung trotzdem betrachtet werden. Hierfür wird betrachtet, welche Möglichkeiten der sicheren Übertragung für die beiden Lösungen zur Verfügung stehen und wie die Authentifikation gesichert werden kann.

Die Nachfolgende Bewertung geschieht auf Basis der in Punkt 14.3 näher erläuterten Bewertungsskala.

14.1 Bewertung der Webservice-Lösung

Bei der Bewertung der *Webservice*-Lösung wird, sofern für ein vergleichbares Ergebnis (Lines of Code (LoC), Datentypen, Codequalität, Performance) notwendig, das Minimalbeispiel aus Kapitel 5 auf Seite 7 verwendet.

Flexibilität (++)

Die Verwendung von *Webservices* ist dank der Standartisierung der Schnittstelle durch die W3C weitgehend sprachunabhängig. Es kann zur Nutzung des *Webservices* jede Programmiersprache mit einer *Webservice*-Funktionalität verwendet werden, was die Einsatzmöglichkeiten wie zum Beispiel die Integration in bestehende Systeme denkbar flexibel macht.
Die Möglichkeit der Verwendung einer *UDDI-Registry* erlaubt das Konsumieren eines *Webservices* ohne das Wissen über die eigentliche Position des Service-Providers

Entwicklungsaufwand (+)

Zur Verwendung von *Webservices* sind weder SAP-seitig noch auf Seiten der konsumierenden Anwendung nennenswerte Vorbereitungen zu treffen. Der reine Entwicklungsaufwand hingegen ist jedoch höher als bei der *RFC*-Lösung. SAP-seitig muss gegebenenfalls ein *RFC*-fähiger Funktionsbaustein angelegt werden, aus dem in mehreren Schritten eine Servicedefinition erstellt und diese dann freigegeben werden muss. Der Weg zu einem verwendbaren Service ist damit wesentlich länger als bei der Verwendung von *RFC*, da dort das Erstellen eines *RFC*-fähigen Funktionsbausteins der einzige notwendige Schritt ist. Auf der Seite der konsumierenden Anwendung wird das Einlesen des WSDL-Dokuments oftmals durch geeignete Tools (*wsimport*) unterstützt, was den

Aufwand für den Entwickler denkbar gering hält, da nur noch der Aufruf eines Services selbst implementiert werden muss. Leider ist die Dokumentation zur Verwendung von *Webservices* mit SAP sehr dürftig, dadurch entsteht bei der ersten Nutzung ein hoher Initialaufwand, um das Zusammenspiel der nötigen SAP-Komponenten und die vielfältigen Einstellungsmöglichkeiten zu überblicken.

Wartbarkeit (-)

Da für jeden Datentyp der Import- und Exportparameter eine eigene Klasse generiert wird, ergibt sich bei größeren Services oftmals eine unübersichtliche Ansammlung von Klassen. Hinzu kommt, dass für die Datentypen eines *Webservices* **immer** die Klassen für die Datentypen generiert werden. Benötigen mehrere *Webservices* einen Parameter des gleichen Typs, so muss man die Klasse für diesen Datentyp mehrfach generieren lassen, da diese Datentypen trotz identischen Aufbaus nicht kompatibel sind. Durch die Vielzahl von generierten Klassen und die mögliche Redundanz von Datentyp-Klassen ergibt sich ein massiver „Overhead“ bei den Lines of Code (*Webservice*-Minimalbeispiel: ca. 600 LoC, *RFC*-Minimalbeispiel: ca. 150 LoC). Bezogen auf die entwickelten Frontends in dieser Bachelorarbeit bedeutet dies einen Größenunterschied bei den Lines of Code um den Faktor 5 (*Webservice*-Frontend: ca. 25.700 LoC, *RFC*-Frontend: ca. 5.700 LoC). Es ist zu erkennen, dass die Codekomplexität bei der *Webservice*-Lösung deutlich höher ist als bei der *RFC*-Lösung

Erweiterbarkeit (-)

Kommt ein neuer Service hinzu, so müssen die Artefakte aus dem WSDL-Dokument erzeugt werden, um den Service verwenden zu können. Dies hat zur Folge, dass wieder eine Reihe von Klassen für die verwendeten Datentypen generiert werden, selbst wenn identische Datentyp-Klassen für einen anderen Service bereits existieren. Leider sind diese Klassen dann nicht kompatibel.
Verändert sich ein bestehender Service, so muss das entsprechende WSDL-Dokument erneut eingelesen und die Artefakte zur Verwendung erneut erzeugt werden. Ebenso müssen gegebenenfalls manuelle Anpassungen im Quelltext vorgenommen werden. Das Erzeugen der Artefakte ist zwar durch die Verwendung von unterstützenden Tools (*wsimport*) denkbar einfach, aber der Overhead durch doppelt generierte Klassen für die Datentypen ist beträchtlich (Vgl. Lines of Code).

Performance (++)

Zur Ermittlung der Performance wird der selbst geschriebene Performance-Tester (Vgl. beigelegte CD *PerfTest⇒SAPBacPerfTest*) verwendet, da ein Test mit vergleichbaren Ergebnissen mit keinem mir bekannten Tool möglich war. Die generierten Artefakte beinhalten alle für die Kommunikation relevanten Informationen, so dass vor dem Aufruf keine Verbindung initialisiert werden muss. Die Laufzeit ist hier nicht messbar und wird mit 0 Millisekunden (*ms*) bewertet. Für die mittlere Laufzeit aus 1.000 Aufrufen des Minimalbeispiels werden 121 ms gemessen, was etwa 30% schneller ist als die durchschnittlich 159 ms pro Aufruf bei der *RFC*-Lösung (Vgl. Abbildungen 30 bis 33 ab Seite 66).

Sicherheit (+)

Für eine sichere Übertragung lassen sich auf dem *SAP-AS* verschiedene Sicherheitsprofile auswählen. Diese unterstützen neben HTTPS auch XML-Signaturen und -Verschlüsselung sowie Schutz durch Zeitstempel und Security Token.

14.2 Bewertung der RFC-Lösung

Bei der Bewertung der *RFC*-Lösung wird, wie auch schon bei der Bewertung der *Webservice*-Lösung, in einigen Punkten das Minimalbeispiel aus Kapitel 5 auf Seite 7 zum Erlangen vergleichbarer Ergebnisse verwendet.

Flexibilität (o)

Durch die zwingend notwendige Verwendung des *JCo* von SAP ist man zurzeit auf die Programmiersprachen *Java* und *C#/.Net* beschränkt. Es gibt keine Informationen darüber, ob der *JCo* in Zukunft auch für weitere Programmiersprachen (Ich habe in diesem Zusammenhang beispielsweise an *Ruby* gedacht) zur Verfügung stehen wird.
Der *RFC*-Ansatz glänzt jedoch durch verschiedene angebotene Kommunikationsarten (aRFC, sRFC, tRFC, qRFC) sowie die Wahl zwischen zustandsloser und zustandsbehafteter Kommunikation und damit auch mit vielfältigeren Einsatzmöglichkeiten.

Entwicklungsaufwand (o)

Bei der Entwicklung mit *RFC* ist bei den Vorbereitungen zu beachten, dass man den *JCo* bei SAP nur mit einem gültigen Entwicklerschlüssel herunterladen kann. Dazu kommt, dass zwar Beispiele zur Verwendung des *JCo* mitgeliefert werden, diese jedoch nur mangelhaft dokumentiert und beschrieben sind. Durch die mangelhafte Dokumentation der Beispiele kommt es hier ebenso wie bei der *Webservice*-Lösung zu einem hohen Initialaufwand, um die Funktionsweise, das Setzen der Parameter und das Auslesen der verschiedenen Typen von Rückgabewerten zu erarbeiten und zu verstehen.
Bei der Implementierung ist, anders als bei der Nutzung von *Webservices*, genaues Wissen über den Ort des SAP-Systems (IP oder URL) sowie die Eigenschaften von Aufrufparametern (z.B. Länge) nötig. Die Verwendung von *Connection Pools* bei Nutzung mehrerer *RFC*-fähiger Funktionsbausteine erleichtert die Entwickung durch die Verwaltung der einzelnen Verbindungen.

Wartbarkeit (++)

Im Gegensatz zur Verwendung komplexer eigener Datentypen wie bei der *Webservice*-Lösung verwendet der *RFC*-Ansatz ausschließlich Strings. Dies und der duch die fehlenden Datentyp-Klassen sehr schlanke Code mit gerade einmal 150 LoC für das Minimalbeispiel (*Webservice*-Lösung: 600 LoC) und 5.700 LoC für das *RFC*-Frontend (*Webservice*-Frontend: 25.700 LoC) tragen zusammen mit der Verwendung von *Connection Pools* zu einfachem und übersichtlichem Code mit niedriger Komplexität bei. Auch die Komplexität einzelner Aufrufe ist aufgrund der Struktur mit dem einzelnen Setzen

der Importparameter sowie dem einzelnen Auslesen der Exportparameter geringer als bei der *Webservice*-Lösung, wo der Aufruf mit allen Parametern des Services sehr lang werden kann.

Erweiterbarkeit (++)

Soll ein neuer Service eingefügt werden, so braucht nur ein neuer *RFC*-fähiger Funktionsbaustein mit der gewünschten Funktionalität im SAP-System angelegt zu werden. Dieser kann direkt im Anschluss mittels *RFC* angesprochen werden.
Ändert sich ein Service, so braucht man neben den Änderungen am Funktionsbaustein nur die Implementierung in der konsumierenden Anwendung anzupassen. Ein aufwändiger Import von neuen Informationen wie bei der *Webservice*-Lösung ist nicht notwendig.

Performance (-)

Beim Testen der Performance der *RFC*-Lösung kam, wie auch schon bei der *Webservice*-Lösung, der selbst geschriebene Performance-Tester zum Einsatz. Es ergab sich im Mittel für 1.000 Aufrufe des Minimalbeispiels eine Laufzeit von 159 Millisekunden (*ms*), womit der *RFC*-Aufruf ca. 30% langsamer war als der *Webservice*-Aufruf. Außerdem ist der Verbindungsaufbau zur Initialisierung des *Connection Pools* und der einzelnen Verbindungen mit durchschnittlich 158 ms im Vergleich zu der mit 0 ms bewerteten Verbindungszeit der *Webservice*-Lösung deutlich langsamer. Aufgrund der fehlenden Initialisierung bei einem *Webservice*-Aufruf ist dieser Punkt allerdings nur schwer vergleichbar (Vgl. Abbildungen 30 bis 33 ab Seite 66).

Sicherheit (+)

Die Übertragung vom *JCo* zum SAP-System kann über *Secure Network Communication (SNC)* gesichert werden. Dafür wird die SAP Kryptografie-Bibliothek benötigt, die wie der *JCo* bei SAP nach Angabe eines Entwicklerschlüssels heruntergeladen werden kann. *SNC* unterstützt neben Ticket-Authentifizierung auch Zertifikate (X.509).

14.3 Gegenüberstellung der Lösungen

Um die beiden Lösungen auf einen Blick vergleichen zu können werden sie in einer Tabelle (Tabelle 14.3 auf Seite 43) gegenübergestellt. Zur besseren Übersicht wird eine Bewertungsskala angewendet, die insgesamt fünf Unterteilungen (++, +, o, -, –) umfasst.

- ++ sehr gut
- \+ gut
- o mittelmäßig
- \- schlecht
- \- - sehr schlecht

Die Bewertungsskala orientiert sich an den Einzelbewertungen in Abschnitt 14.1 und 14.2. Die Gesamtbewertung eines Bewertungskriteriums ergibt sich aus dem Mittelwert der aufgeführten Unterpunkte. Hierfür werden die Einzelbewertungen der Unterpunkte mit 1 (- - sehr schlecht) bis 5 (++ sehr gut) bewertet, der Mittelwert gebildet und dieser kaufmännisch gerundet.

	Webservice	**RFC**
Flexibilität	**++**	**o**
Programmiersprache	++	o
Entwicklungsaufwand	**+**	**o**
Vorbereitung	++	-
Aufwand	o	o
Wartbarkeit	**-**	**++**
Codekomplexität	o	+
Lines of Code	-	++
Datentypen	- -	++
Aufrufkomplexität	o	+
Erweiterbarkeit	**-**	**++**
Service neu	-	++
Service geändert	-	++
Performance	**++**	**-**
Verbinden	++	- -
Aufrufen	+	-
Sicherheit	**+**	**+**

Tabelle 3: Gegenüberstellung der Bewertungen

14.4 Entscheidung

Es ist schwierig, bei den beiden verglichenen Lösungen eine klare Entscheidung zu treffen. Beide Lösungen haben spezifische Vor- und Nachteile, die sie für unterschiedliche Einsatzgebiete mehr oder weniger geeignet erscheinen lässt.

Verwendung von *Webservices*

Kommt es bei der Umsetzung eines Projekts auf

- Flexibilität,
- Entwicklungsaufwand und
- Performance

an, so ist die Verwendung von *Webservices* ratsam. Durch die standartisierte Schnittstelle von *Webservices* hat man in Bezug auf die Flexibilität bei der Programmiersprache

klare Vorteile gegenüber dem *RFC*. Beim Entwicklungsaufwand ist zwar zu erwähnen, dass SAP-seitig mehr Schritte bis zur Fertigstellung eines nutzbaren Services nötig sind, dies wird jedoch durch die oftmals durch Tools unterstützte Generierung der nötigen Artefakte in der konsumierenden Anwendung ausgeglichen. Im Punkt Performance ist eine Lösung über *Webservices* messbar schneller als die gleiche Lösung mit *RFC*.

Verwendung von *RFC*

Wird bei der Umsetzung eines Projekts besonderes Augenmerk auf

- Wartbarkeit und
- Erweiterbarkeit

gelegt, so bietet sich eine Realisierung mittels *RFC* an. Der sehr schlanke und übersichtliche Code auf der Seite der konsumierenden Anwendung sowie die ausschließliche Verwendung von Strings als Datentypen erhöht die Wartbarkeit im Vergleich zur *Webservice*-Lösung mit den unzähligen generierten Klassen für die Datentypen und Servicedefinitionen. Bei einer Erweiterung oder Anpassung der Services brauchen nicht wie bei der *Webservice*-Lösung Artefakte neu generiert werden. Es wird nur die Verwendung des geänderten oder hinzugefügten Services neu implementiert.

14.5 Aufgetretene Probleme

Im Rahmen der Bachelorarbeit traten einige Probleme bzw. Schwierigkeiten auf, die zwar nicht im direkten Zusammenhang mit der Arbeit stehen, die ich an dieser Stelle trotzdem gerne erwähnen möchte.

- Es kam leider häufiger vor, dass die VM mit dem SAP-System der Uni-Hamburg abgestürzt bzw. nicht erreichbar war. Die Fehler wurden zwar immer schnell behoben, aber hielten die Entwicklung teilweise auf.
- Durch die Verwendung der Version *NetWeaver 2004s Trial* war es nicht möglich andere Klienten außer den voreingestellten Klienten 000 zu verwenden, wobei dieser eigentlich nicht für Entwicklungsarbeiten verwendet werden soll. Diese Restriktion musste abgestellt werden, um die Webservicefunktionalität zu nutzen.
- Das SAP-System war manchmal äußerst langsam, was die Entwicklung erschwerte.
- Die SAP-seitige Dokumentation (speziell zu *RFC* und zum *JCo*) war sehr dürftig. In diesem Punkt hätte ich mehr erwartet.

Glossar

4GL	*Fourth Generation Language*, beschreibt Programmiersprachen und Programmierumgebungen, deren Zielsetzung es ist, den Programmieraufwand durch kürzere und verständlichere Programme zu verringern, die Wartbarkeit und Erweiterbarkeit zu verbessern und die daraus resultierenden Kosten zu senken (Wikipedia/4GL, 2008), 6
ABAP	*Advanced Business Application Programming*, bezeichnet die SAP-eigene Programmiersprache, 6
At most Once	Serviceeigenschaft, ein Service wird höchstens einmal ausgeführt und gilt nach dem ersten gescheiterten Versuch als fehlgeschlagen, 30
BAPI	*Business Application Programming Interface*, aus dem SAP Business Framework, stellt Schnittstellen an Komponentengrenzen dar, 11
BAPI-Explorer	Oberfläche in SAP zur Verwaltung, Darstellung und Benutzung mehrerer *BAPIs* verschiedener SAP-Module, 11
Connection Pool	Dient der Verbindungsverwaltung und stellt eine Verbindung auf Anforderung zur Verfügung, 31
CPI-C	*Common Programming Inferface for Communication*, eine standartisierte Schnittstelle für eine systemübergreifende Kommunikation von Programmen (IBM), 30
Datenelement	Ein Datenelement stellt einen in SAP selbst definierten Datentyp dar, 28
Dynpro	*Dynamisches Programm*, bildet die klassische Oberfläche eines ABAP-basierten SAP-Programms, 3
EDT	*Event Dispatch Thread*, ein Thread in dem alle Aktionen einer *Swing*-Applikation ausgeführt werden, 25
ERP	*Enterprise Ressource Planning* (Unternehmens-Informationssystem), 5

Exactly Once — Serviceeigenschaft, ein Service wird genau einmal ausgeführt. Ist das Zielsystem nicht erreichbar, so kommt der Aufruf in eine Warteschlange, 30

Exactly Once in Order — Serviceeigenschaft, ein Service wird genau einmal ausgeführt. Ist das Zielsystem nicht erreichbar, so kommt der Aufruf in eine Warteschlange. Die Reihenfolge der Aufrufe bleibt erhalten, 30

FatJar — Ein Eclipse-Plugin zur Erstellung von ausführbaren .jar-Archiven, 26

Funktionsbaustein — Nicht eigenständig lauffähiger Baustein, kapselt eine bestimmte Funktionalität, kann aus allen lauffähigen Programmen aufgerufen werden, 7

Funktionsgruppe — Eigener Programmtyp in SAP, stellt eine Sammlung von Funktionsbausteinen dar, 7

ICF — *Internet Communication Framework*, ermöglicht Programmen des *SAP-AS* direkten Internetzugang, ist Basis für Webservices unter SAP, 3

invokeLater — Fügt ein Runnable zur asynchronen Abarbeitung in den EDT ein, 25

JCo — Von SAP zur Verfügung gestellter Java-Connector zur Anbindung von Java-Anwendungen an SAP via *RFC*, 31

Lose Kopplung — Lose Kopplung bezeichnet geringe Abhängigkeit von Softwarekomponenten untereinander, 11

MVC — Model-View-Controller-Prinzip, ein Architekturmuster für Software, Model = Datenmodell auf das zugegriffen wird, View = Sicht auf die Daten / GUI, Controller = Steuerung zur Behandlung und Verarbeitung von Benutzeraktionen und den dazugehörigen Reaktionen, 3

Parameterliste — Eine Liste aller Parameter eines entfernten Funktionsbausteins. Es gibt eine Import-, Export-, Table- und Changingparameterliste, 33

Positionstransparenz — Auch: Ortstransparenz, erlaubt den Zugriff auf Ressourcen ohne Kenntnis über deren Position, 3

Property Datei	Eine Datei, die den Inhalt eines Java-Property-Objekts in Form von Key/Value-Paaren (Key = Bezeichnung, Value = konkreter Wert) repräsentiert, 32
RFC	*Remote Function Call*, SAP-eigenes, proprietäres Protokoll für das Aufrufen von Funktionen in entfernten Systemen, 3
RFCFassade	Beschreibt die Komponente der *RFC*-Lösung, die die eigentliche Kommunikation mit dem SAP-System und somit den *RFC* ausführt. Die RFCFassade wird vom Controller aufgerufen und liefert die Ergebnisse eines Aufrufs an den Controller zurück, 37
SAP-AS	*SAP Application Server* (SAP Applikationsserver der NetWeaver Plattform), 5
SAP-BP	*SAP-Business Partner*, ein SAP-Modul zur zentralen Verwaltung von Geschäftspartnern und dazugehörigen Kontaktdaten, 5
SAP-GUI	SAP-Graphical User Interface, Benutzeroberfläche für SAP-Nutzer und Vorraussetzung für die Ausführung von ABAP-Programmen, 5
Servicefassade	Beschreibt die Komponente der *Webservice*-Lösung, die die eigentliche Kommunikation mit dem SAP-System und somit die Aufrufe der Webservices ausführt. Die Servicefassade wird vom Controller aufgerufen und liefert die Ergebnisse eines Aufrufs an den Controller zurück, 16
Swing	Grafikbibliothek zur Programmierung von UIs, entwickelt und stetig weiterentwickelt von Sun Microsystems, 11
Tabellentyp	Ein Tabellentyp ist eine Darstellung einer Tabelle bestehend aus einem bestimmten Datentyp (Eine Zeile des Tabellentyps entspricht dem spezifizierten Datenelement), 28
UDDI	*Universal Description, Discovery and Integration Protocol*, ein Verzeichnisdienst für Webservices, vergleichbar mit den ‘gelben Seiten’, 7
VPN	*Virtual Private Network*, ermöglicht Kommunikation mit einem benachbarten Netzwerk, 3

Web Dynpro	*Web Dynamisches Programm*, eine neue Technologie die browserbasierte Oberflächen (HTML) nach MVC für SAP-Systeme ermöglicht, 3
WSDL	*Web Service Description Language*, standartisierte, XML-basierte Sprache zur Beschreibung von Schnittstellen bei Webservices, 7
WSDL-Dokument	Dokument zur Beschreibung der Schnittstelle eines Webservices, 7
wsimport	*Webservice-Import*, ein seit dem JDK 6 beiliegendes Tool zur Generierung von Java-Code aus einem WSDL-Dokument. Um *wsimport* nutzen zu können muss der Pfad zum installierten JDK der Umgebungsvariablen des Betriebssystems hinzugefügt werden, 8

A Anwendungsfalldiagramm

Um einen besseren Überblick zu bekommen, wird hier mit einem Anwendungsfalldiagramm der Zusammenhang zwischen den einzelnen Anwendungsfällen dargestellt. Wie in Abbildung 17 zu sehen ist, gibt es auch Abhängigkeiten der einzelnen Anwendungsfälle untereinander. So verwenden die Anwendungsfälle *Anlegen*, *Ändern* und *Löschen* alle den Anwendungsfall *Anzeigen*, um nach einem verändernden Zugriff auf einen Business Partner die Anzeige der einzelnen Business Partner in der Anwendung zu aktualisieren. Abhängigkeiten von Anwendungsfällen untereinander werden durch einen gestrichelten Pfeil mit der Beschriftung *«uses»* dargestellt. Die detaillierten Anwendungsfälle befinden sich in Anhang B.

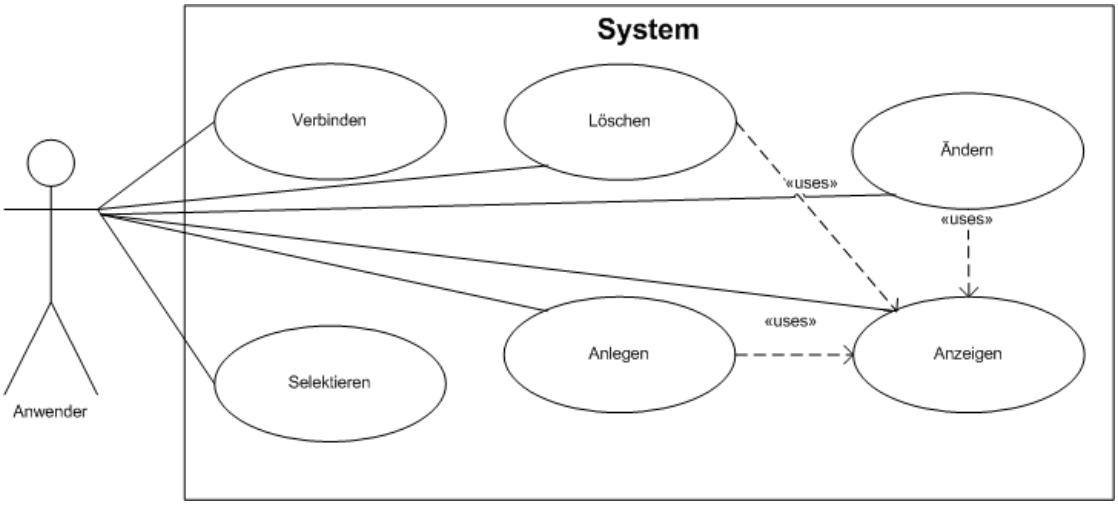


Abbildung 17: Anwendungsfalldiagramm

B Anwendungsfälle

Zur genaueren Spezifizierung der Funktionalität des zu entwickelnden Frontends werden hier die Anwendungsfälle für die geplanten Funktionalitäten des Frontends dargestellt. Für alle Anwendungsfälle werden neben den typischen, geplanten Abläufen auch mögliche fehlerhafte Abläufe berücksichtigt. Bei allen Anwendungsfällen wird auf die Angabe eines Autors verzichtet, da sich dieser bereits durch die Bachelorarbeit ergibt.

B.1 Selektieren

Akteure

- Anwender des Frontends

Auslöser

Interaktion des Anwenders zur Selektion eines einzelnen Business Partners.

Kurzbeschreibung

Selektieren eines Business Partners in der Hauptapplikation.

Vorbedingungen

- Services sind initialisiert (Anwendungsfall: B.2)
- Business Partner werden angezeigt (Anwendungsfall: B.3)

Typischer Ablauf

Der Anwender selektiert per Mausklick auf den entsprechenden Business Partner diesen aus der Anzeige in der Hauptapplikation.

B.2 Verbinden

Akteure

- Anwender des Frontends

Auslöser

Interaktion des Anwenders zur Initialisierung des Frontends.

Kurzbeschreibung

Initialisierung der Services im Frontend.

Vorbedingungen

- Es besteht eine Verbindung zum SAP-System (LAN oder VPN)

Typischer Ablauf

Der Anwender startet die Hauptapplikation und anschließend per Knopfdruck die Initialisierung der Services. In der Applikation werden daraufhin die Services für die Verwendung initialisiert und der Anwender aufgrund der Dauer der Initialisierung mit einer Progress-Bar über den Fortschritt informiert. Bis zum Abschluss der Initialisierung sind alle anderen Funktionalitäten der Applikation gesperrt. Der Anwender wird am Ende über die erfolgreiche Initialisierung informiert und die Applikation ist in einem Initialzustand bereit zur weiteren Verwendung.

Alternativer Ablauf: Service(s) nicht erreichbar

Sind einzelne oder mehrere Services nicht erreichbar, so kann die Initialisierung nicht abgeschlossen werden. In diesem Fall wird der Anwender über die fehlgeschlagene Initialisierung informiert und die Applikation verbleibt im Ausgangszustand.

B.3 Anzeigen

Akteure

- Anwender des Frontends

Auslöser

- Automatisch gestartetes Auslesen der Business Partner
- Interaktion des Anwenders mit dem Frontend

Kurzbeschreibung

Anzeigen der Business Partner (beinhaltet das Auslesen aus dem SAP-System und das Verarbeiten der ausgelesenen Daten zur Darstellung in der Applikation).

Vorbedingungen

- Services sind initialisiert (Anwendungsfall: B.2)

Typischer Ablauf

Für diesen Anwendungsfall gibt es zwei mögliche typische Abläufe. Zum einen kann das Anzeigen von Business Partnern durch den Anwender direkt ausgeführt werden, zum anderen wird nach verändernden Zugriffen auf Business Partner (Anlegen, Ändern, Löschen) das Anzeigen automatisch gestartet, um die Ansicht zu Aktualisieren.

Anzeigen durch den Anwender: Der Anwender wählt die Funktion *Anzeigen* mittels Knopfdruck und aus dem SAP-System werden daraufhin alle Business Partner ausgelesen. Die ausgelesenen Daten werden verarbeitet und für die Darstellung in der Applikation aufbereitet. Die aufbereiteten Datensätze werden in der Applikation dargestellt, aufgrund der Dauer wird der Fortschritt des Auslesens dem Anwender mit einer Progress-Bar visualisiert.

Automatisches Anzeigen: Nach einem veränderndem Zugriff auf einen Business Partner wird das Anzeigen und das damit verbundene Auslesen der Business Partner automatisch gestartet, um die Ansicht zu aktualisieren. Es werden aus dem SAP-System alle Business Partner ausgelesen und die ausgelesenen Daten werden für die Darstellung in der Applikation aufbereitet. Der Fortschritt wird dem Anwender auch hier über eine Progress-Bar visualisiert.

Alternativer Ablauf: Service(s) nicht erreichbar

Ist der Service zum Anzeigen der Business Partner nicht erreichbar, so wird dem Anwender das fehlgeschlagene Anzeigen durch eine Fehlermeldung dargestellt.

Alternativer Ablauf: SAP-System gibt einen Fehler zurück

Gibt das SAP-System einen Fehler zurück, so wird dem Anwender dieser Fehler aufgezeigt. Das Auslesen und Anzeigen der Business Partner ist fehlgeschlagen und die Applikation verbleibt im Ausgangszustand. Der Ausgangszustand kann der Initialzustand der Anwendung direkt nach dem Starten oder die vorherige Darstellung der Business Partner sein.

B.4 Anlegen

Akteure

- Anwender des Frontends

Auslöser

Interaktion des Anwenders um einen neuen Business Partner im System anzulegen.

Kurzbeschreibung

Anlegen eines neuen Business Partners

Vorbedingungen

- Services sind initialisiert (Anwendungsfall: B.2)

Nachbedingungen

- Business Partner werden (erneut) angezeigt (Anwendungsfall: B.3)

Typischer Ablauf

Der Anwender wählt die Funktion *Anlegen* und trifft die Entscheidung, in welcher Kategorie ein neuer Business Partner angelegt werden soll. Daraufhin öffnet sich ein Dialog mit Textfeldern für die kategoriespezifischen Daten zum Anlegen des Business Partners. Die Daten werden bei Bestätigung auf Korrektheit der Eingabe bezüglich lokal prüfbarer Kriterien (z.B.: Länge von Eingaben) überprüft und anschließend zum SAP-System übertragen. Der Anwender wird über das erfolgreiche Anlegen des neuen Business Partners informiert und die Anzeige der Business Partner in der Applikation wird durch ein erneutes Auslesen der Business Partner aktualisiert.

Alternativer Ablauf: Service(s) nicht erreichbar

Ist der Service zum Anlegen eines Business Partners nicht erreichbar, so wird dem Anwender das fehlgeschlagene Anlegen als Fehlermeldung dargestellt. Der Dialog mit den eingetragenen Daten bleibt geöffnet, um das Anlegen erneut zu probieren.

Alternativer Ablauf: Lokale Datenprüfung schlägt fehl

Schlägt die Prüfung der eingegebenen Daten fehl, so wird dem Anwender dies über eine Fehlermeldung angezeigt und der Business Partner wird nicht angelegt. Stattdessen kann der Anwender die Daten korrigieren und das Anlegen mit Überprüfung der Eingaben erneut durch Bestätigung ausführen lassen.

Alternativer Ablauf: SAP-System gibt einen Fehler zurück

Gibt das SAP-System einen Fehler beim Anlegen des Business Partners zurück, so wird dieser dem Anwender aufgezeigt. Der Business Partner wurde bei einem aufgetretenen Fehler nicht angelegt.

Alternativer Ablauf: Anwender bricht das Anlegen ab

Wird das Anlegen eines Business Partners durch den Anwender abgebrochen (über den *Zurück*-Button oder das Schließen des Dialogs), so werden die eingetragenen Daten verworfen und das Dialogfenster geschlossen.

B.5 Ändern

Akteure

- Anwender des Frontends

Auslöser

Interaktion des Anwenders um die Daten eines Business Partners zu ändern.

Kurzbeschreibung

Ändern eines bestehenden Business Partners

Vorbedingungen

- Services sind initialisiert (Anwendungsfall: B.2)
- Business Partner ausgewählt (Anwendungsfall: B.1)

Nachbedingungen

- Business Partner werden erneut angezeigt (Anwendungsfall: B.3)

Typischer Ablauf

Der Anwender hat in der Applikation einen Business Partner selektiert und wählt die Funktion *Ändern*. In Abhängigkeit von der Kategorie des selektierten Business Partners öffnet sich ein Dialog mit den kategoriespezifischen Feldern, die bereits mit den Daten des zu ändernden Business Partners gefüllt sind. Der Anwender ändert die gewünschten Daten ab und bestätigt den Änderungsvorgang, woraufhin die Daten auf ihre Korrektheit bezüglich lokal zu prüfender Kriterien (Eingabelänge, Pflichtfelder etc.) geprüft werden. Zusätzlich wird anschließend geprüft, ob die Remotedaten im SAP-System im Vergleich zum Zustand vor der Änderung wiederum selbst eine Veränderung erfahren haben, um die Konsistenz zu wahren. Ist die Änderung abgeschlossen, so wird der Anwender über die erfolgreiche Änderung informiert und die Anzeige der Business Partner in der Applikation durch erneutes Auslesen aktualisiert.

Alternativer Ablauf: Service(s) nicht erreichbar

Ist der Service zum Ändern eines Business Partners nicht erreichbar, so wird dem Anwender das fehlgeschlagene Ändern als Fehlermeldung dargestellt. Der Dialog mit den eingetragenen Daten bleibt geöffnet, um das Ändern erneut zu probieren.

Alternativer Ablauf: SAP-System gibt einen Fehler zurück

Gibt das SAP-System einen Fehler beim Ändern des Business Partners zurück, so wird dieser dem Anwender aufgezeigt. Der Business Partner wurde bei einem aufgetretenen Fehler nicht geändert.

Alternativer Ablauf: Anwender bricht das Ändern ab

Wird das Ändern eines Business Partners durch den Anwender abgebrochen (über den *Zurück*-Button oder das Schließen des Dialogs), so werden die eingetragenen Daten verworfen und das Dialogfenster geschlossen.

Alternativer Ablauf: Lokale Datenprüfung schlägt fehl

Schlägt die Prüfung der eingegebenen Daten fehl, so wird dem Anwender dies über eine Fehlermeldung angezeigt und der Business Partner wird nicht geändert. Stattdessen kann der Anwender die Daten korrigieren und das Ändern mit Überprüfung der Eingaben erneut durch Bestätigung ausführen lassen.

Alternativer Ablauf: Remotedaten weisen einen neueren Stand auf

Weisen die Remotedaten zu einem zu ändernden Business Partner im SAP-System einen anderen Stand auf als die Ausgangsdaten des Business Partners in der lokalen Applikation, so wird dem Anwender dieses mit einer Fehlermeldung signalisiert. Der Ändern-Dialog wird geschlossen und ein neuer Dialog mit den aktuelleren Daten des Business Partners geöffnet. Bereits eingetragene Veränderungen werden verworfen.

Alternativer Ablauf: Kein Business Partner ausgewählt

Wurde kein Business Partner in der Applikation selektiert, so erhält der Anwender eine Fehlermeldung, dass für eine Änderung ein Business Partner zu selektieren ist.

Alternativer Ablauf: Business Partner existiert nicht (mehr) im System

Wurde der Business Partner zwischenzeitlich aus dem SAP-System gelöscht, so erhält der Anwender eine Fehlermeldung. Eine Änderung ist für diesen Business Partner nicht mehr möglich.

B.6 Löschen

Akteure

- Anwender des Frontends

Auslöser

Interaktion des Anwenders um einen Business Partner zu löschen.

Kurzbeschreibung

Löschen eines ausgewählten Business Partners

Vorbedingungen

- Services sind initialisiert (Anwendungsfall: B.2)
- Business Partner ausgewählt (Anwendungsfall: B.1)

Nachbedingungen

- Business Partner werden erneut angezeigt (Anwendungsfall: B.3)

Typischer Ablauf

Der Anwender hat einen Business Partner in der Applikation selektiert und wählt die Funktion *Löschen*. Der Selektierte Business Partner wird aus dem SAP-System gelöscht und anschließend die Anzeige in der Applikation durch erneutes Auslesen der Business Partner aktualisiert.

Alternativer Ablauf: Service(s) nicht erreichbar

Ist der Service für das Löschen eines Business Partners nicht erreichbar, so erhält der Anwender eine Fehlermeldung.

Alternativer Ablauf: SAP-System gibt einen Fehler zurück

Gibt das SAP-System einen Fehler beim Löschen des Business Partners zurück, so wird dieser dem Anwender aufgezeigt. Der Business Partner wurde bei einem aufgetretenen Fehler nicht gelöscht.

Alternativer Ablauf: Kein Business Partner ausgewählt

Wurde kein Business Partner in der Applikation selektiert, so erhält der Anwender eine Fehlermeldung, dass für das Löschen ein Business Partner zu selektieren ist.

C UML-Diagramme für das Frontend

Um die Struktur des Frontends offen zu legen werden hier exemplarisch UML-Diagramme des Frontends dargestellt. Die Diagramme sind teilweise in Bezug auf Methoden und Attribute vereinfacht und nicht vollständig parametrisiert dargestellt und dienen dazu, sich einen Überblick über das System zu verschaffen.

C.1 Controller

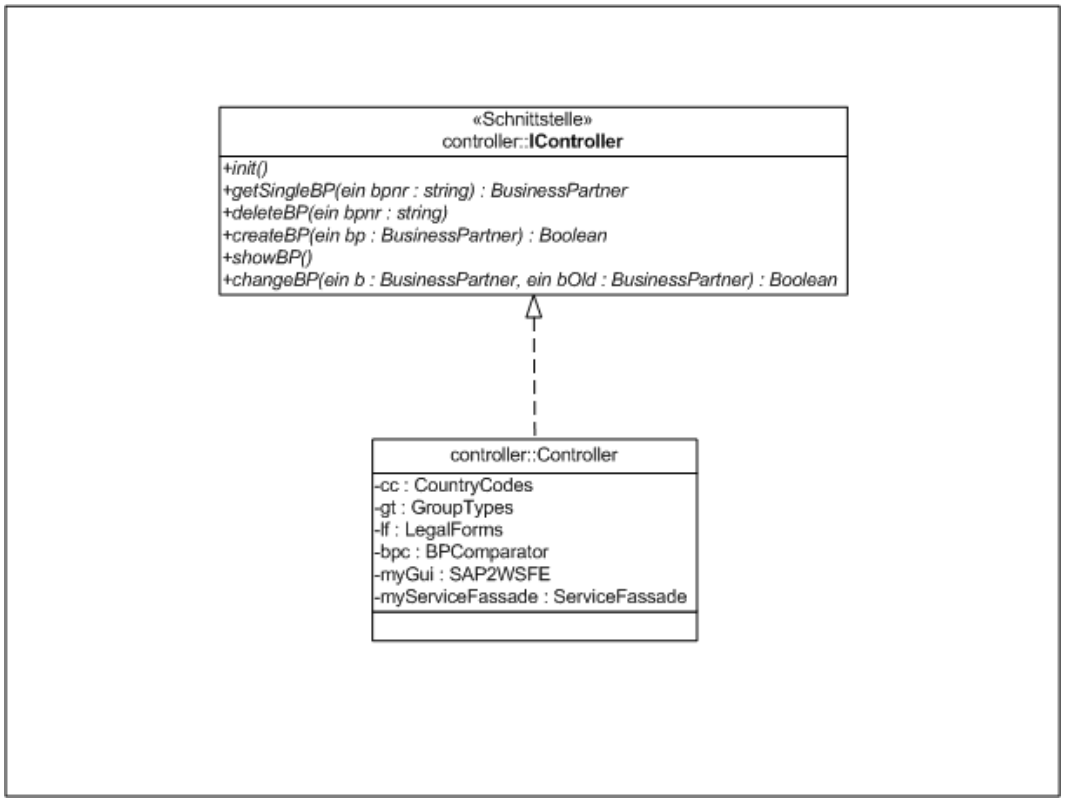


Abbildung 18: UML-Diagramm des Controllers

C.2 Services (Webservice-Lösung)

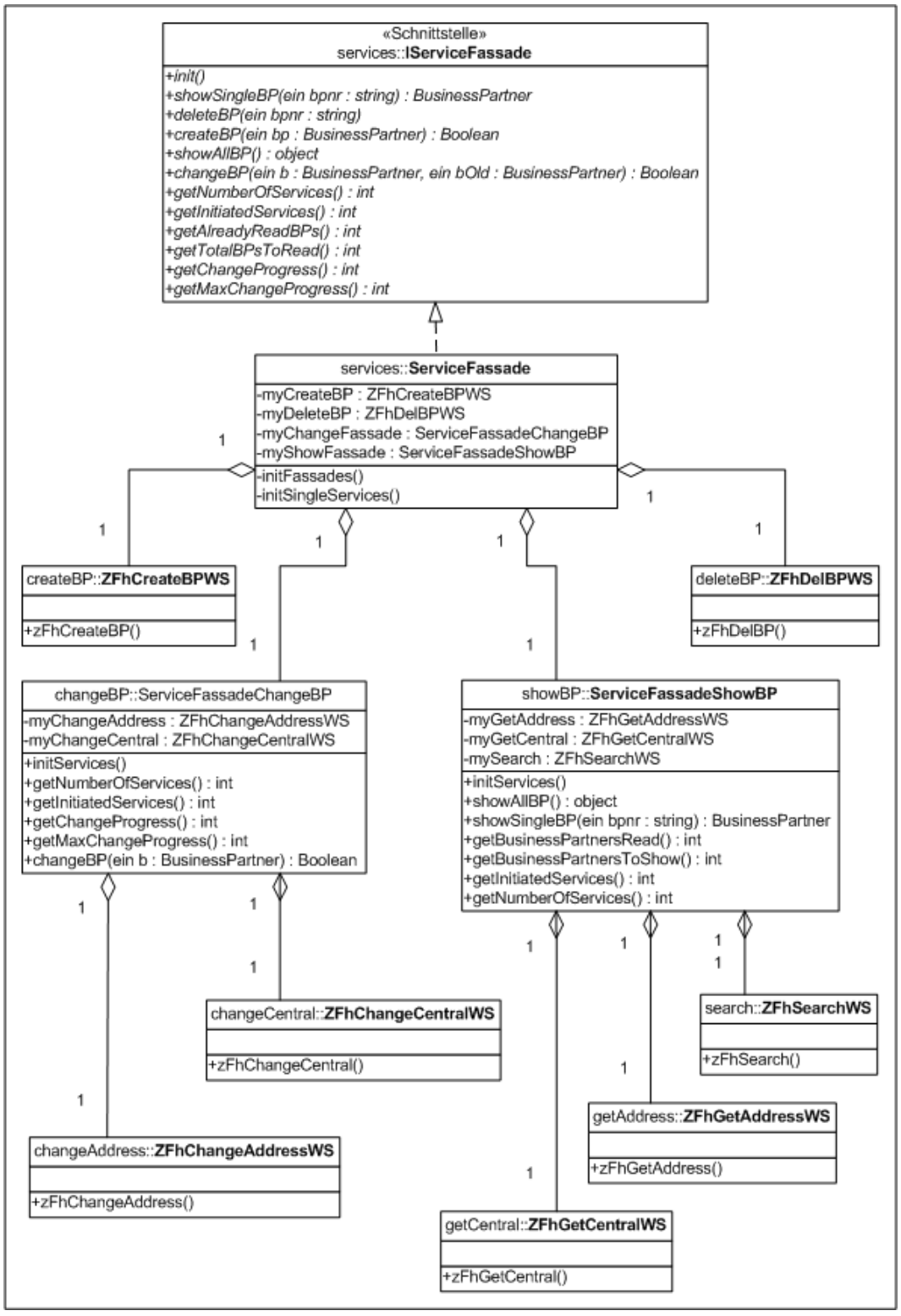


Abbildung 19: UML-Diagramm der Services und der Servicefassade (WS)

C.3 Services (RFC-Lösung)

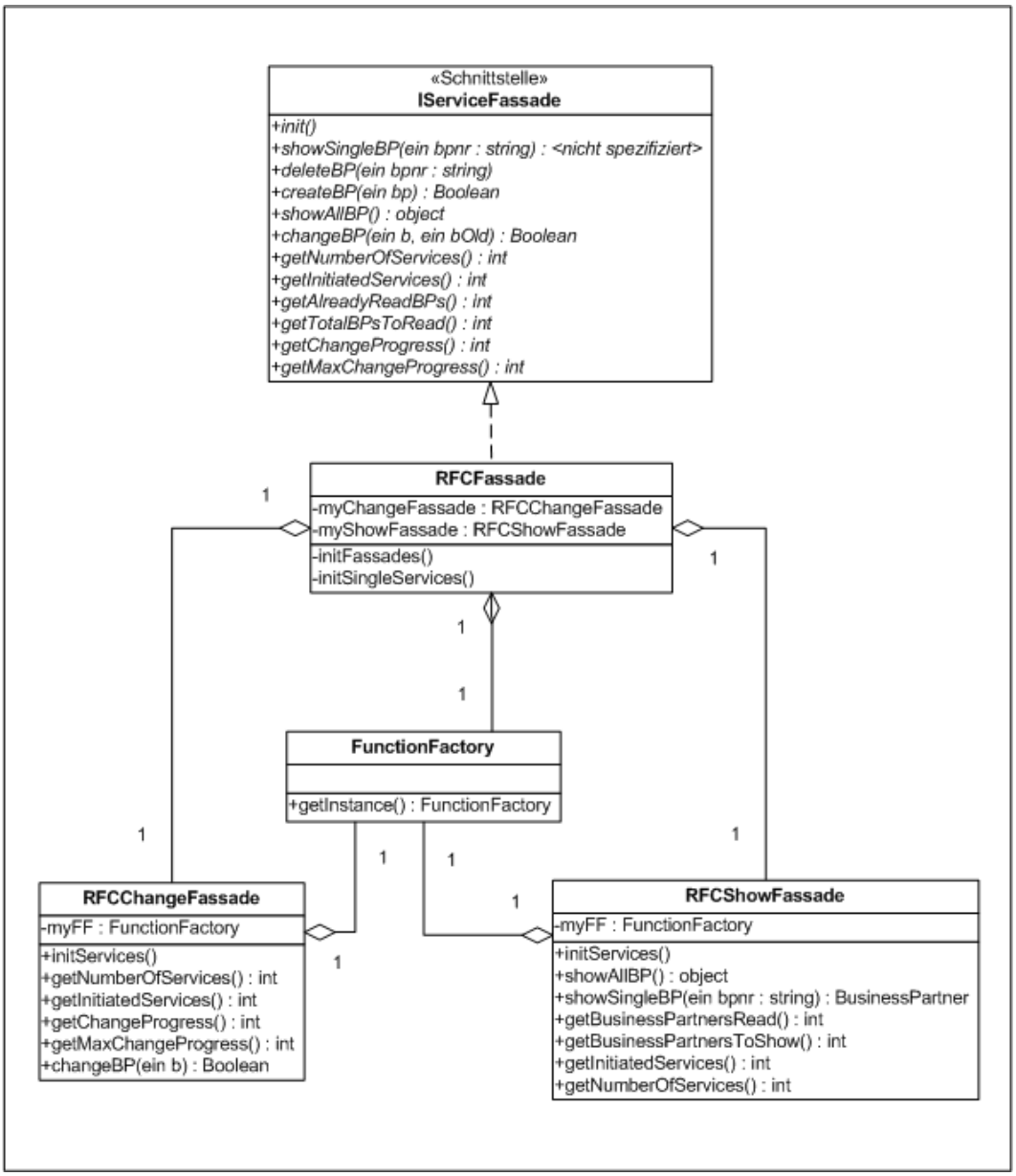


Abbildung 20: UML-Diagramm der Services und der Servicefassade (RFC)

D Sequenzdiagramme für das Frontend (Webservices)

Zur übersichtlicheren Darstellung des Kommunikationsflusses werden die Sequenzdiagramme teilweise vereinfacht dargestellt.

D.1 Webservice-Lösung

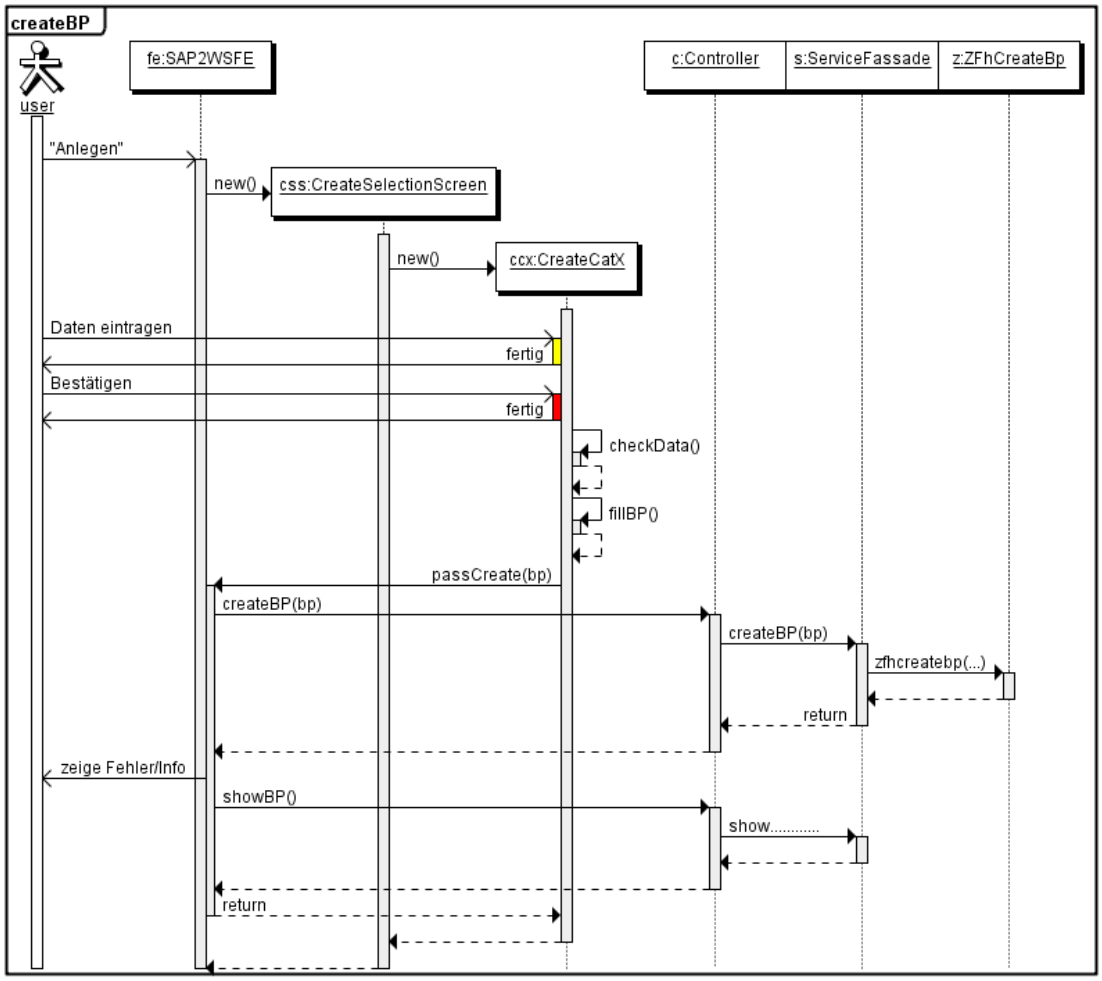


Abbildung 21: Sequenzdiagramm: Erfolgreiches Anlegen eines Business Partners (WS)

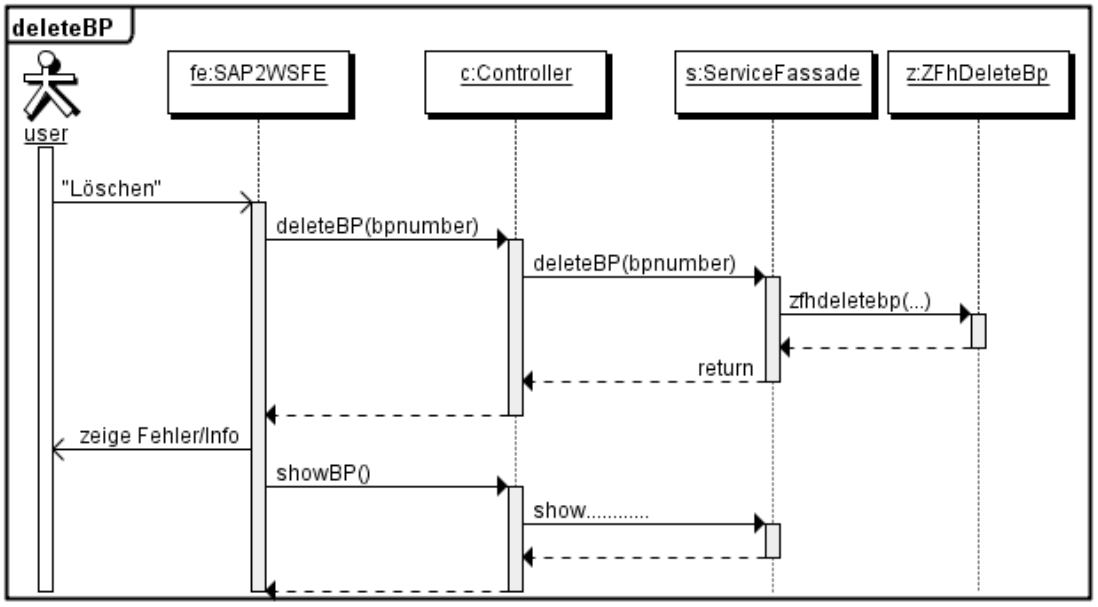


Abbildung 22: Sequenzdiagramm: Erfolgreiches Löschen eines Business Partners (WS)

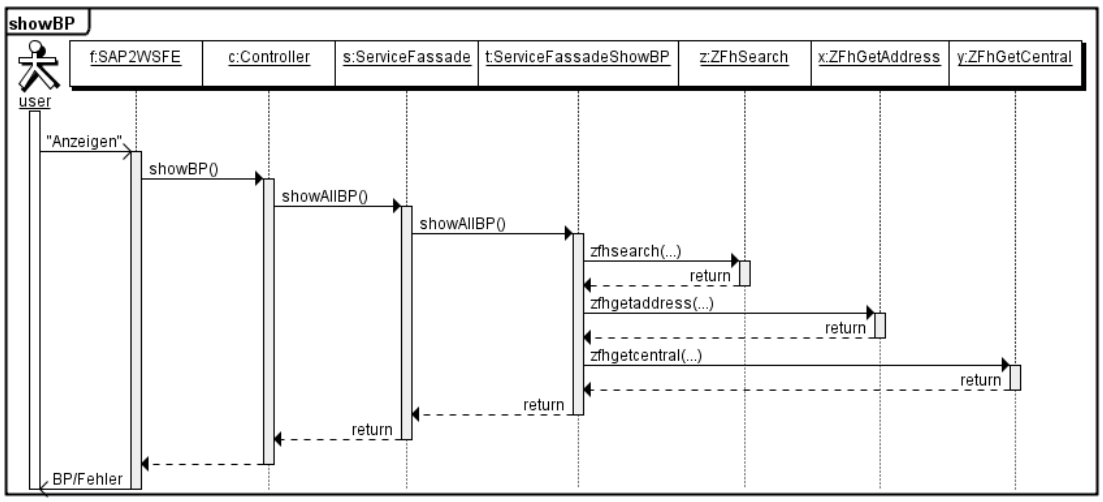


Abbildung 23: Sequenzdiagramm: Erfolgreiches Anzeigen der Business Partner (WS)

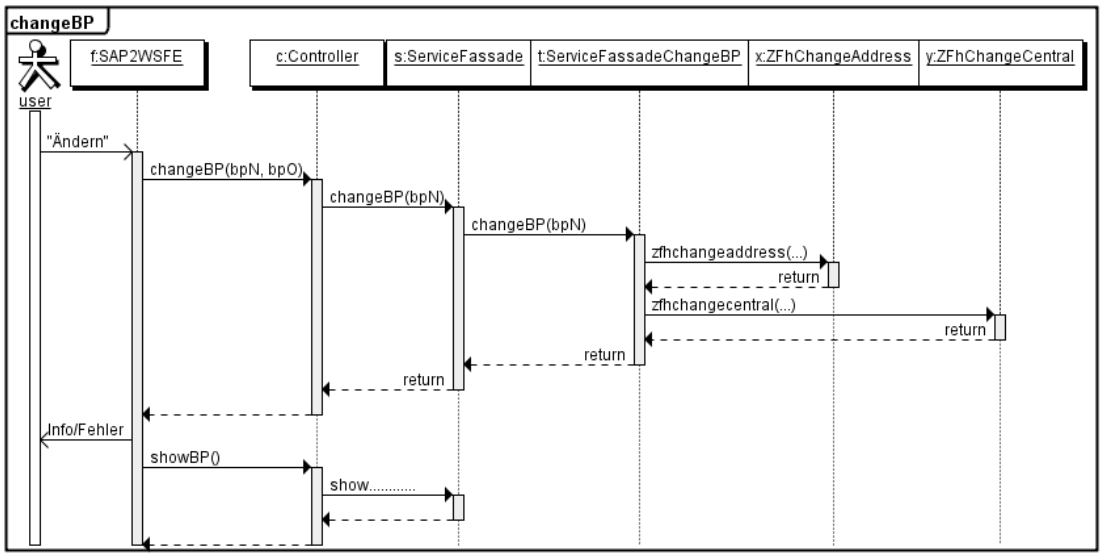


Abbildung 24: Sequenzdiagramm: Erfolgreiches Ändern der Business Partner (WS)

D.2 RFC-Lösung

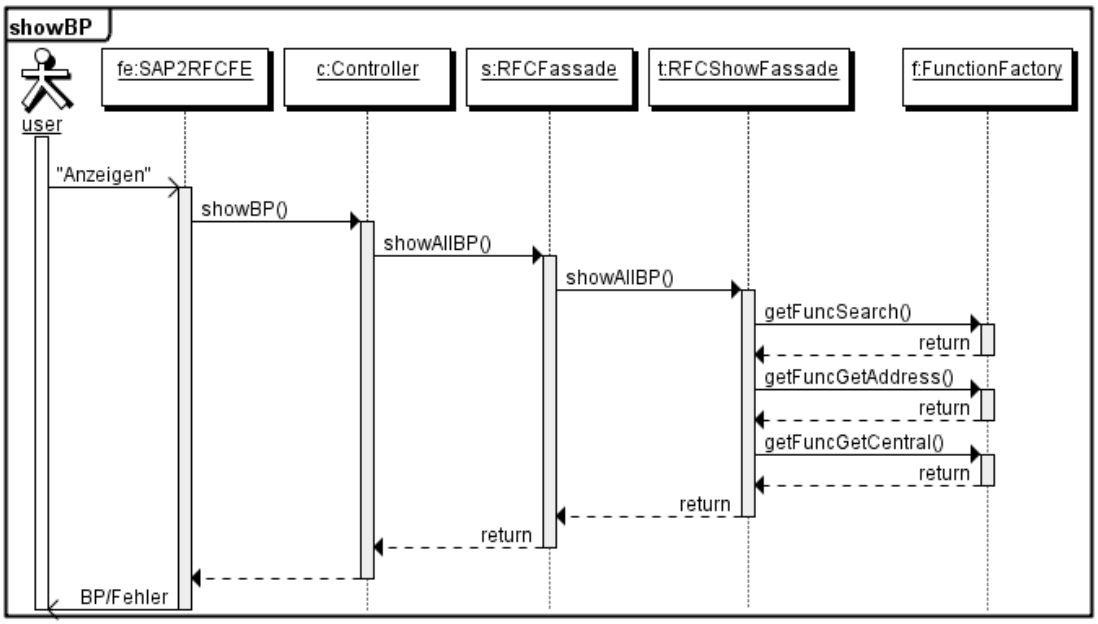


Abbildung 25: Sequenzdiagramm: Erfolgreiches Anzeigen der Business Partner (RFC)

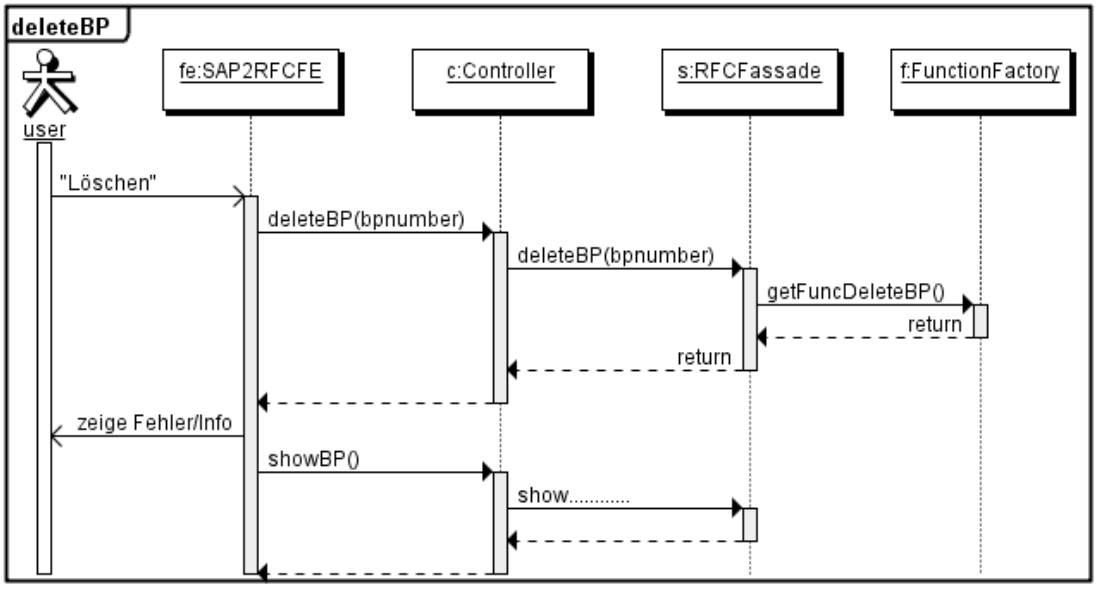


Abbildung 26: Sequenzdiagramm: Erfolgreiches Löschen eines Business Partners (RFC)

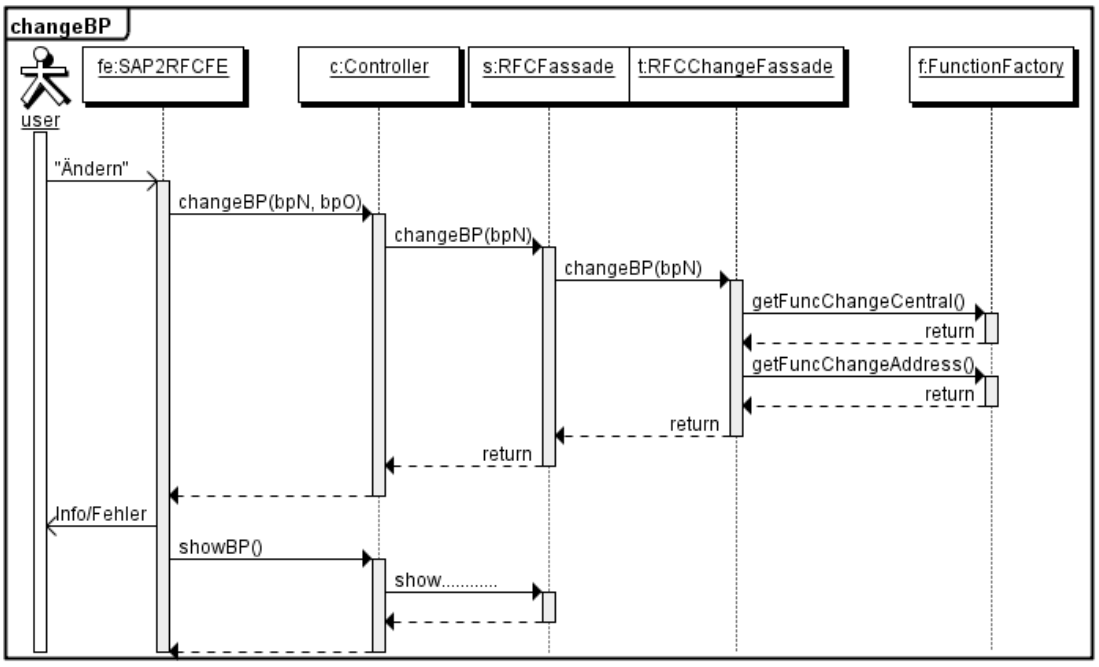


Abbildung 27: Sequenzdiagramm: Erfolgreiches Ändern der Business Partner (RFC)

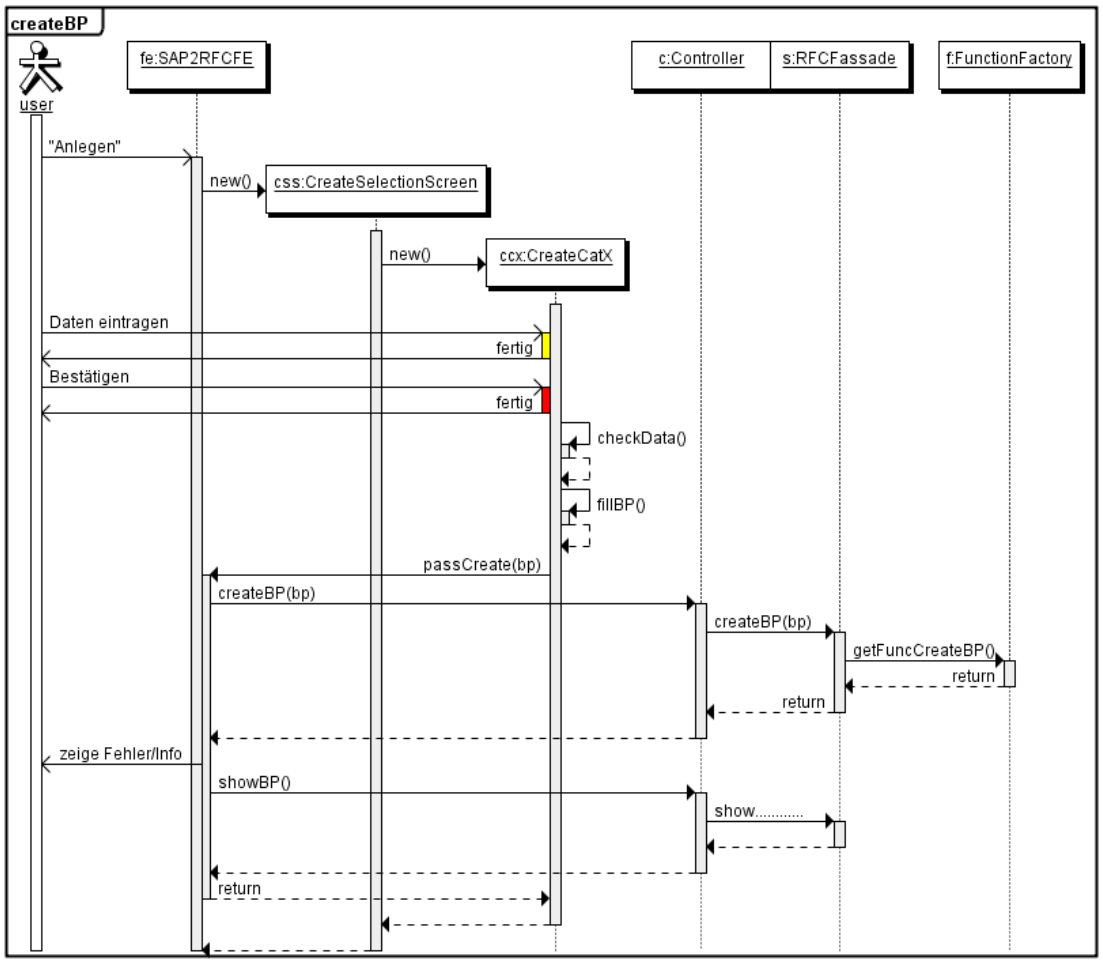
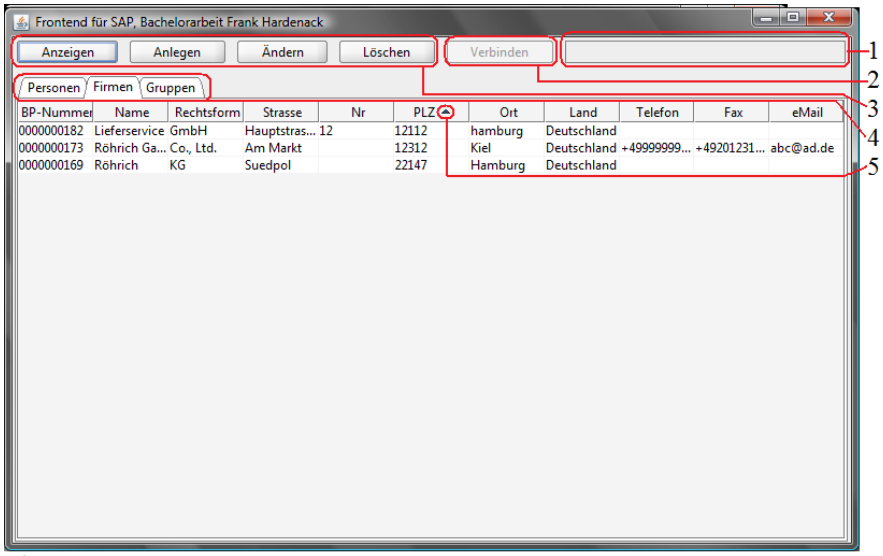


Abbildung 28: Sequenzdiagramm: Erfolgreiches Anlegen eines Business Partners (RFC)

E GUI des Frontends

Darstellung der grafischen Benutzeroberfläche des Frontends mit Legende und Beschreibung der einzelnen Komponenten.



1) Label zum Anzeigen von Info-Texten
2) "Verbinden"-Button zur Initialisierung der Services
3) Die vier Funktionalitäten des Frontends als einzelne Buttons
4) Die Reiter der TabbedPane für die einzelnen BP-Gruppen
5) Sortierbare Spalten

Abbildung 29: Grafische Benutzeroberfläche des Frontends mit Legende

F Diagramme zur Performanceanalyse

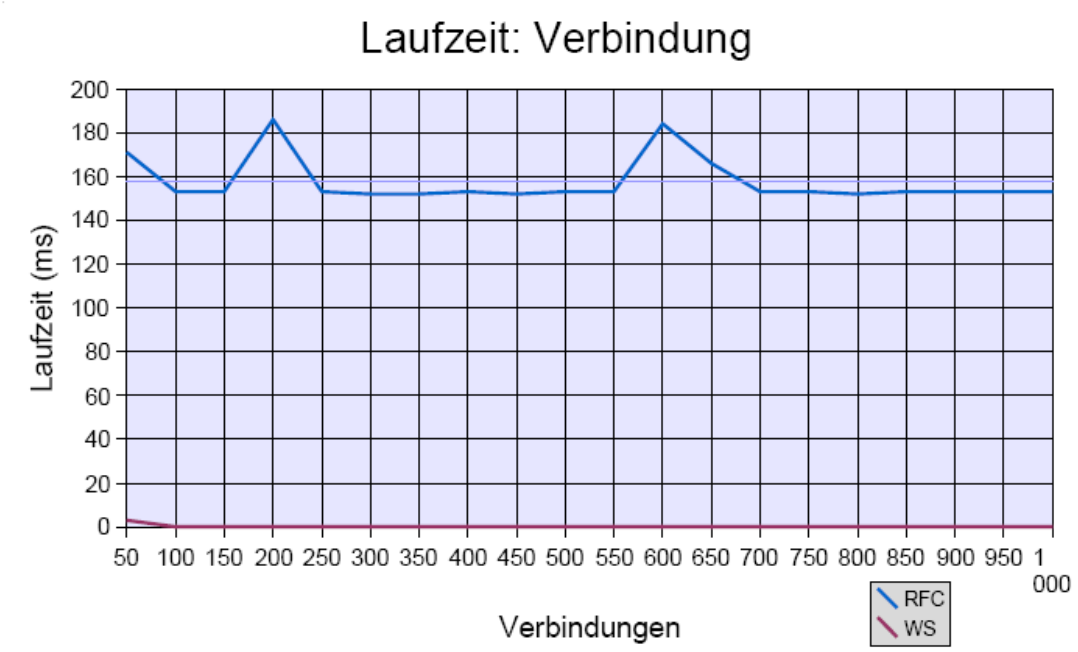


Abbildung 30: Diagramm für die Laufzeitanalyse: Verbindungen

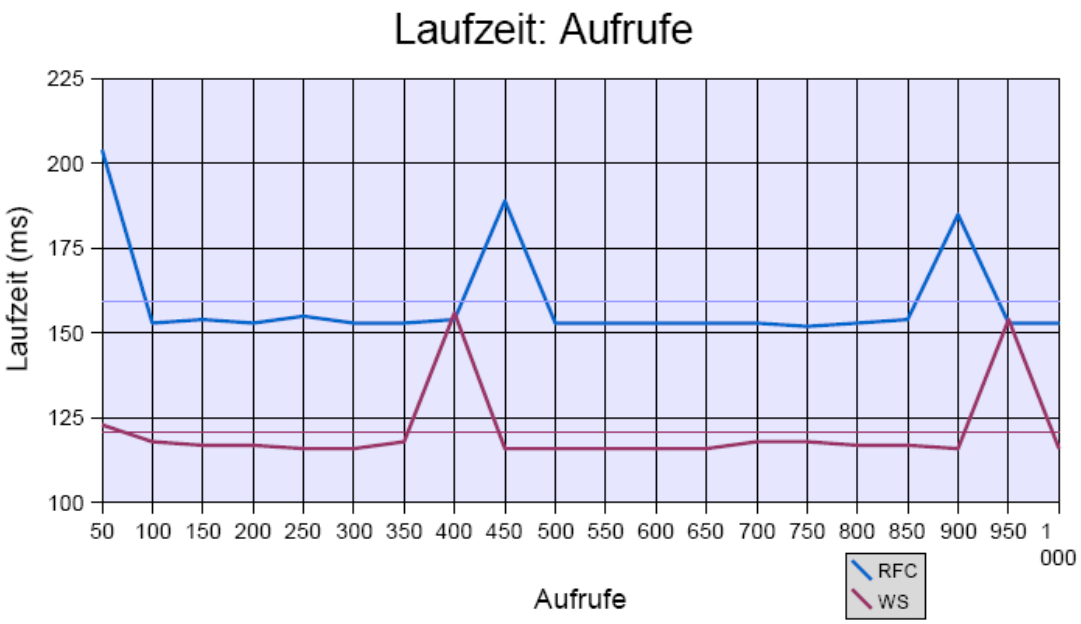


Abbildung 31: Diagramm für die Laufzeitanalyse: Aufrufe

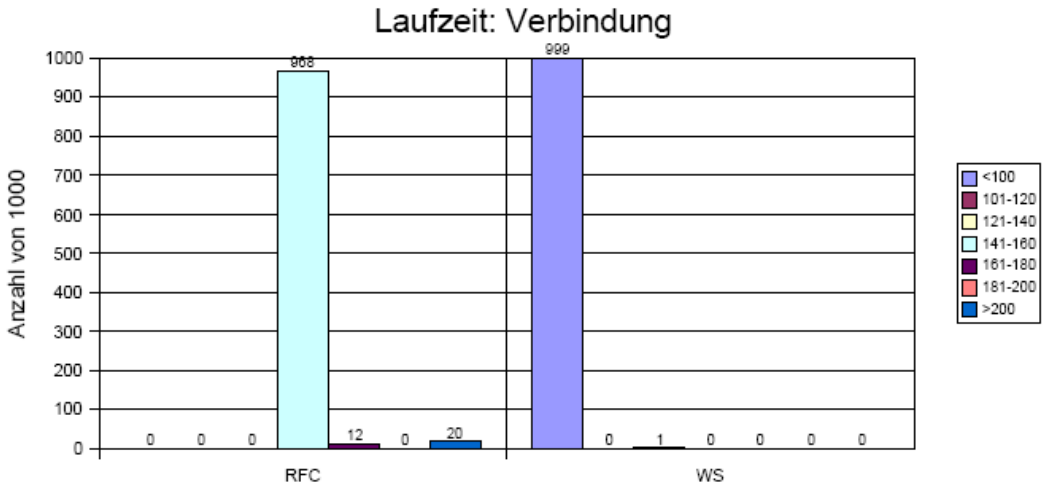


Abbildung 32: Histogramm für die Laufzeitanalyse: Verbindungen

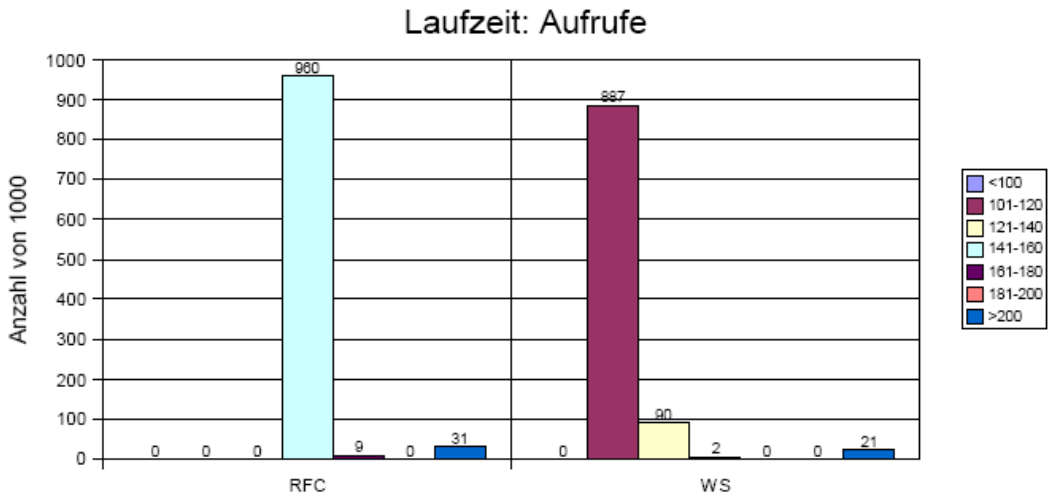


Abbildung 33: Histogramm für die Laufzeitanalyse: Aufrufe

G Übersicht der erstellten Grafiken und Diagramme

Die in dieser Bachelorarbeit dargestellten Diagramme und Grafiken wurden alle von mir persönlich erstellt. Zum Einsatz kamen sowohl kommerzielle als auch freie Tools. Verwendete Programme:

- Microsoft Visio 2003 (MSDNAA-Version)
- SDEdit (Freeware-Tool für Sequenzdiagramme)
- OpenOffice

Verwendetes Tool	Titel	Abbildung
MS Visio 2003	Relevante Methoden für das Anlegen von BPs	Abb. 7
	Relevante Methoden für das Anzeigen von BPs	Abb. 8
	Relevante Methoden für das Ändern von BPs	Abb. 9
	Relevante Methoden für das Löschen von BPs	Abb. 10
	Paketdiagramm des Frontends	Abb. 11
	Detaillierte Darst. der Servicefassade (WS-Lösung)	Abb. 12
	Detaillierte Darst. der Servicefassade (RFC-Lösung)	Abb. 16
	Anwendungsfalldiagramm	Abb. 17
	UML-Diagramm des Controllers	Abb. 18
	UML-Diagramm der Services und der Servicefassade (WS)	Abb. 19
	UML-Diagramm der Services und der Servicefassade (RFC)	Abb. 20
SDEdit	Sequenzdiagramm für erfolgr. Anlegen eines BP (WS)	Abb. 21
	Sequenzdiagramm für erfolgr. Ändern eines BP (WS)	Abb. 24
	Sequenzdiagramm für erfolgr. Anzeigen eines BP (WS)	Abb. 23
	Sequenzdiagramm für erfolgr. Löschen eines BP (WS)	Abb. 22
	Sequenzdiagramm für erfolgr. Anzeigen eines BP (RFC)	Abb. 25
	Sequenzdiagramm für erfolgr. Löschen eines BP (RFC)	Abb. 26
	Sequenzdiagramm für erfolgr. Ändern eines BP (RFC)	Abb. 27
	Sequenzdiagramm für erfolgr. Anlegen eines BP (RFC)	Abb. 28
OpenOffice	Diagramm für die Laufzeitanalyse: Verbindungen	Abb. 30
	Diagramm für die Laufzeitanalyse: Aufrufe	Abb. 31
	Histogramm für die Laufzeitanalyse: Verbindungen	Abb. 32
	Histogramm für die Laufzeitanalyse: Aufrufe	Abb. 33

Tabelle 4: Auflistung der einzelnen Grafiken und Diagramme

Index

4GL, 6

ABAP, 3, 6, 30
ABAP-O, 6
Anwendungsfall, 19, 35, 49, 50
    Ändern, 53
    Anlegen, 52
    Anzeigen, 51
    Diagramm, 19, **49**
    Löschen, 55
    Selektieren, 50
    Verbinden, 50
Artefakt, 8, 11, 26, 40
At most Once, 30

BAPI, **12**, 20, 27
    Explorer, 12
Bewertungskriterium, 38, 42
    Anpassbarkeit, 38, 40, 42
    Codequalität, 38, 40, 41
    Entwicklungsaufwand, 38, 39, 41
    Erweiterbarkeit, 38, 40, 42
    Flexibilität, 38, 39, 41
    Performance, 38, 40, 42, 66
    Sicherheit, 39, 41, 42
    Wartbarkeit, 38, 40, 41
Bewertungsskala, 42
BPComparator, 24
Business Partner, 5
    ändern, 13, 62, 63
    anlegen, 12, 60, 64
    anzeigen, 12, 61, 62
    löschen, 13, 61, 63
BusinessPartner, 24

C1WPS, 2
C#/.Net, 4, 41
Commit-Problem, 21
Connection Pool, **31**, 32, 34, 37, 41, 42
Controller, **17**, 34, 35, 57
CountryCodes, 24
CPI-C, 30

Datenelement, 27
    anlegen, 28
    eigenes, 21
Datentypen, 40, 41
    eigene, 25, 36
    SAP, 23
Dynpro, 3

EDT, 25, 26
Endpunkt, 7, 11
Exactly Once, 30
    In Order, 31

Fassade, 16
FatJar, 26, 37
Fehlerbehandlung, 17, **26**
Function Factory, 37
Funktionsbaustein, 7, 12, 20, 34, 37
    RFC-fähig, 7, 27, 32, 34, 36, 39, 41
Funktionsgruppe, 7, 12

GroupTypes, 24
GUI, 16, 17, 34, 35, 65

Hilfsklasse, 24, 36
    BPComparator, 24
    BusinessPartner, 24
    CountryCodes, 24
    GroupTypes, 24
    LegalForms, 24
Holder, 11, 12

ICF, 3, 7
Interface, 25, 35, 37
invokeLater, 26
itelligence AG, 2

JAR-File, 26, 32, 37
Java, 4, 11, 31, 38, 41
JButton, 17
JCo, 31, 32, 34, 41
JCoFunction, 33, 37
JComboBox, 24
JCoRepository, 33
JDialog, 25, 26

JProgressBar, 17, **26**
JTabbedPane, 17
JTable, 17

Kommunikation
    asynchron, 30
    synchron, 30
    zustandslos, 34

LegalForms, 24
Lines of Code, 40, 41
Live-Referenz-Problem, 21, 27
Look and Feel, 11
Lose Kopplung, 11, 16, 35

Modality Type, 25
    Application Modal, 25
MVC, 3, 16

Observable, 17, **26**
Observer, 17, **26**

Paketdiagramm, 18
Parameter
    Export-, 7, 40
    Import-, 7, 40
Parameterliste
    Changing-, 33
    Export-, 33, 37
    Import-, 33, 37
    Table-, 33
Positionstransparenz, 4, 39
Property-Datei, 32

Query-String, 8

RFC, 1, 3, 30, 33
    aRFC, 30, 41
    qRFC, 31, 41
    sRFC, 30, 34, 41
    tRFC, 30, 41
RFCFassade, 37
RPC, 30
Ruby, 41

SAP
    AS, 5, 8, 30
    BP, 3, 5
    GUI, 4, 5
    JCo, 31
    Modulkonzept, 5
    NetWeaver, 7
    Object Navigator, 7
    Transaktion, 7
Schnittstelle, 17, 31, 35, 39
SDEdit, 68
Secure Network Communication, 42
Sequenzdiagramm, 60
Service
    Provider, 4, 7, 11, 39
    Requester, 4, 7, 11
Servicedefinition, 7, 8, 22, 26, 27, 39
Servicefassade, 16, 35, 37
Singleton, 37
Swing, 11, 25

Tabellentyp, 27
    anlegen, 28
TCP/IP, 30

UDDI, 7, 11, 39
UML-Diagramm, 57

Visio 2003, 68
VPN, 3, 11, 33, 34, 51

W3C, 39
Wartbarkeit, 35, 38, 40, 41
Web Dynpro, 3
Webservice, 1, 7, 22
    anlegen, 7
Wertübergabe, 21
Wiederverwendbarkeit, 16, 25
Wrapper, 13, 20–22, 34, 36
    anlegen, 27
WSADMIN, 8
WSCONFIG, 7
WSDL, 7, 8
    Dokument, 7, 8, 24, 26, 27
    URL, 8
    Version, 8
wsimport, 8, **11**, 26, 39, 40

X.509, 33, 42

Literatur

[C1WPS 2008] C1WPS: *C1WPS GmbH*. 2008. – URL http://www.c1wps.de. – [Online; Stand 10. Oktober 2008]

[Eberhart und Fischer 2003] EBERHART, Andreas ; FISCHER, Stefan: *Web Services - Grundlagen und praktische Umsetzung mit J2EE und .Net*. Auflage 1. Carl Hanser Verlag, 2003. – ISBN 3-446-22530-7

[Gamma u. a. 1996] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Auflage 1. Addison Wesley, 1996. – ISBN 0-201-63361-2

[Keller und Krüger 2007] KELLER, Horst ; KRÜGER, Sascha: *ABAP Objects - ABAP Programmierung mit SAP NetWeaver*. Auflage 3. SAP-Press / Galileo-Press, 2007. – ISBN 978-3-89842-358-8

[Royce 1998] ROYCE, Walker: *Software Project Management. A Unified Framework*. Addison-Wesley, 1998. – ISBN 0-201-30958-0

[Royce 1970] ROYCE, Winston W.: Managing the Development of Large Software Systems, Concepts and Techniques. In: *Proceedings of IEEE WESCON*, 1970. – Der Artikel, in dem das Wasserfallmodell zum ersten Mal als Vorgehensmodell präsentiert wurde.

[SAP AG 2008] SAP AG: *SAP im Überblick*. 2008. – URL http://www.sap.com/germany/about/investor/ueberblick/index.epx. – [Online; Stand 12. Dezember 2008]

[SAP AG und SAP Entwickler 2008] SAP AG UND SAP ENTWICKLER: *SAP Developers Network and Forum*. 2008. – URL https://www.sdn.sap.com/irj/sdn. – [Online; Stand 10. Dezember 2008]

[Wikipedia/4GL 2008] WIKIPEDIA/4GL: *4GL — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=4GL&oldid=51746817. – [Online; Stand 14. Oktober 2008]

[Wikipedia/ABAP 2008] WIKIPEDIA/ABAP: *ABAP — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=ABAP&oldid=51219425. – [Online; Stand 14. Oktober 2008]

[Wikipedia/RFC 2008] WIKIPEDIA/RFC: *Remote Function Call — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=Remote_Function_Call&oldid=50302898. – [Online; Stand 26. Oktober 2008]

[Wikipedia/SAP 2008] WIKIPEDIA/SAP: *SAP — Wikipedia, Die freie Enzyklopädie*. 2008. – URL http://de.wikipedia.org/w/index.php?title=SAP&oldid=51838309. – [Online; Stand 17. Oktober 2008]

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 7. April 2009 Frank Hardenack