Fachhochschule Köln
Cologne University of Applied Sciences

Zugriff auf nachrichtenorientierte Kommunikationsdienste von mobilen Geräten

Entwicklung einer bidirektionalen Verwaltungskomponente

Access to message oriented communication services for mobile devices

Development of a biderectional management component

Bachelorarbeit

ausgearbeitet von

Jochen Christian Todea

zur Erlangung des akademischen Grades

Bachelor of Science

vorgelegt an der

Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und Ingenieurwissenschaften

im Studiengang

Medieninformatik

Erster Prüfer:	Prof. Dr. Kristian Fischer Fachhochschule Köln
Zweiter Prüfer:	Prof. Dr. Stefan Karsch Fachhochschule Köln

Gummersbach, im August 2010

Adressen: Jochen Todea
Ackerstraße 34
51702 Bergneustadt
jochen@todea.de

Prof. Dr. Kristian Fischer
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
kristian.fischer@fh-koeln.de

Prof. Dr. Stefan Karsch
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
stefan.karsch@fh-koeln.de

Kurzfassung

Diese Arbeit befasst sich mit der Problematik der Zugriffe auf nachrichtenorientierte Kommunikationsdienste von mobilen Geräten. Da diese Kommunikationsdienste nicht dafür ausgelegt sind via Netzwerk zugänglich zu sein, sondern nur lokal in der gleichen virtuellen Maschine ausgeführt werden können, wird ein zusätzlicher Dienst entwickelt der diesen Zugriff möglich macht. Zusätzlich beinhaltet er die verarbeitende Komponente und fungiert somit als Schnittstelle. Es wird detailliert darauf eingegangen wie genau ein Lösungsansatz aussehen könnte, in Rücksichtnahme auf gewählte Softwarekomponenten. Eine Abwägung der jeweiligen Komponenten und Darstellung der Alternativen, sowie eine Diskussion des Problem- sowie Lösungsraums, gehen dem Ganzen voraus. Im Mittelpunkt der Arbeit steht die Verarbeitung der Anfragen und Weiterleitung an bestehende nachrichtenorientierte Systeme mittels Web Services, sowie ihre Darstellung. Somit wird die Funktionalität eines solchen Systems, in diesem Fall Java Message Service, genutzt um eine Kommunikation zwischen diversen Nutzern zu etablieren. Der Gedanke dabei ist, einen Nachrichtenaustausch zwischen diesen, mittels wohl definierter Schnittstellen, zu ermöglichen. Weiterhin wird die Funktion anhand einer prototypischen Implementation demonstriert und ein Ausblick gegeben in wie weit sich dieser Prototyp erweitern lässt um dem Gesamtkonzept zu genügen.

Abstract

This work deals with the problem of access to message oriented services via mobile devices. Since these communication services are not designed to be accessible via the network, but only can be carried out locally in the same virtual machine, an additional service is developed which makes this access possible. In addition, it includes the processing component and therefore acts as an interface. It is discussed detaily how a solution looks like in consideration of selected software components. An assessment of the respective components, a presentation of alternatives and a discussion of the problem and solution space will advance the whole. The focus is set to the processing of requests and forwarding to existing systems using Web services as well as their representation. Thus the functionality of such a system in this case Java Message Service used to establish a communication between various user. The idea is to exchange messages between them by using well defined interfaces. Furthermore, the function is demonstrated with a prototypical implementation and an outlook is given to what degree this Prototype can meet to the overall concept.

Inhaltsverzeichnis

1 Einleitung

Heutzutage treten mobile Geräte immer mehr in den Vordergrund. Es werden ständig neue und verbesserte Modelle auf den Markt gebracht und somit ergibt sich auch für die Softwareindustrie ein lukrativer Markt. Applikationen für diese Geräte sind nicht mehr weg zu denken und die Qualität dieser Anwendungen wächst stetig, da auch die zur Verfügung stehenden Recourcen einem deutlichen Wachstum unterliegen. Wenn vor einigen Jahren noch leistungsschwache Geräte auf dem Markt waren die auf eingeschränkte Entwicklungsumgebungen angewiesen waren, so ist es heute kaum noch der Fall. Diverse Geräte haben schon 1GHz Prozessoren verbaut und sind in der Lage eine Vielzahl von Aufgaben zu verarbeiten denen ältere Geräte nicht gewachsen waren. Neue und leistungsfähigere Betriebsysteme und Software-Plattformen fanden ihren Einzug. Wie ein auf Java basierendes frei zugängliches System von Google (Android). Eine sehr interessante Entwicklung die einer genaueren Betrachtung bedarf. So könnten neue Bereiche für mobile Geräte zugänglich gemacht werden. Ein weiterer wichtiger Aspekt ist die Kommunikation zwischen solchen Geräten und ihren Benutzern. Diese Aspekte werden aufgegriffen, um ein Konzept einer dienstorientierten Anwendung im mobilen Bereich zu diskutieren. Im Mittelpunkt steht der Nachrichtenaustausch zwischen bekannten Nutzern. Erweitert wird das Ganze um Funktionen zum Abonnieren von Texten um somit auf dem neusten Stand gehalten zu werden, falls sie vom Autor aktualisiert werden. Zu Demonstrationszwecken wird ein Prototyp entwickelt. Hierbei werden verschiedene Software-Elemente miteinander gekoppelt, um eine Schnittstelle für mobile Geräte zur Verfügung zu stellen. Benutzer sind in der Lage diesen Dienst mittels einer Anwendung zu nutzen. Der Gedanke ist, mit nur einer Applikation mehrere Bereiche abzudecken, für die es bis dato verschiedene Applikationen erforderte.

1.1 Gliederung

Die Arbeit gliedert sich in mehrere Teile. Zum einen die Vorstellung einer Architektur, die Diskussion der gewählten Softwarekomponenten und als zentraler Punkt die Entwicklung des Web Dienstes, der anhand einer Implementation demonstriert wird. Hierzu zählt noch die Entwicklung eines mobilen Clients mit der zugrundeliegenden Programmlogik und einer grafischen Oberfläche, die als Eingabe für den Benutzer dient. Abschließend wird noch die Kommunikationskomponente zwischen den jeweiligen Einheiten betrachtet.
In Kapitel eins erfolgt nach der Einleitung ein Überblick über die Problemstellung, das Konzept und ein Szenario. Im Zweiten wird die Architektur vorgestellt und anhand einer Skizze verdeutlicht, außerdem erfolgt eine Abwägung und Begründung wieso die Softwarekomponenten gewählt wurden. Im dritten Kapitel werden alle Softwarekomponenten betrachtet und vorgestellt. Im Vierten wird der gesamte Entwicklungsprozess, vom Erstellen des Projekts bis zur detaillierten Implementation, beschrieben. In Kapitel fünf werden verwandte Arbeiten vorgestellt, die sich mit gleichen, ähnlichen oder Teilproblemen befassen, um zu verdeutlichen, dass in diesem Segment Nachholbedarf an Softwarelösungen besteht. In den beiden letzten Kapiteln wird ein Ausblick gegeben, in wie weit der Prototyp erweitert werden kann und ein Fazit gezogen.

1.2 Konzept

Der Grundgedanke besteht einmal darin, eine Kommunikation, die auf JMS basiert, zu realisieren, die es einem Benutzer erlaubt Nachrichten an andere bekannte Nutzer zu versenden. Desweiteren soll die Möglichkeit bestehen Blogs oder Newsfeeds zu lesen. Die direkte Kommunikation wird mittels der sogenannten Queue realisiert, die einen eins zu eins Nachrichtenaustausch zwischen zwei Personen darstellt. Hierbei wird lediglich eine Identifikation des Gegenüber benötigt, um eine Nachricht zu hinterlassen. Es können neben Textnachrichten auch binäre Daten hinterlegt werden, wie z.B. ein Bild oder ein Dokument. Falls der Gesprächspartner zur Zeit nicht erreichbar ist wird die Nachricht bei Wiederverfügbarkeit zugestellt. Man könnte dieses Szenario mit einem Instant Messanger vergleichen. Der andere Aspekt besteht darin, ein Newsfeed oder Blog zugänglich zu machen, mittels Topics. Hier soll die Eingabe einer URL von Nöten sein um den Blog zu abonnieren und das System soll sich um alles weitere kümmern. Das bedeutet, falls neue Einträge vorhanden sind, soll dies erkannt und der neue Text soll in das Topic übernommen werden. In beiden Fällen wird der Benutzer darüber informiert, das neue Nachrichten eingetroffen sind bzw. werden sie ihm dargestellt. Dieses Abonnement kann auch wieder rückgängig gemacht werden.

Anhand einer Übersicht werden die abonnierten Feeds und die bereits eingetragenen Freunde angezeigt. Ein Vorteil dieses Konzepts wäre, mit Freunden zu kommunizieren ohne jegliche Anmeldung bei einem Provider, wie das zum Beispiel im Bereich des Instant Messaging nötig ist. Weiterhin stellt dies eine schnellere Art der Kommunikation dar als das beim E-Mail-Verkehr der Fall ist. Das Hauptaugenmerk liegt hier nicht auf der Entwicklung des Clients, sondern auf einer Programmierschnittstelle, die zur Verfügung gestellt wird, um Clientapplikationen zu implementieren die wie beschrieben agieren.

1.3 Szenario

Max ist 25 Jahre alt und Besitzer eines Mobiltelefons mit Internetzugang. Er möchte seinen Freunden Nachrichten schreiben ohne auf SMS zurückgreifen zu müssen, da hierbei, je nach Anzahl, der Kostenfaktor hoch sein kann. E-Mails zu schreiben ist ihm zu langwierig um sich mal nach dem Wohlergehen diverser Leute zu erkundigen. Weiterhin ist Max auch sehr interessiert an manchen Blogs und möchte diese abonnieren um immer auf dem Laufenden zu sein. Dafür benötigt er normalerweise zwei verschiedene Applikationen auf seinem Gerät. Deswegen entscheidet er sich für das System das beides implementiert. Da hierbei keine Anmeldung erforderlich, sondern lediglich die Installation der Software von Nöten ist, erzählt er seinen Freunden davon, die sich daraufhin auch für die Nutzung entscheiden. Jetzt kann Max seine Freunde unter einem Namen in seine Freundesliste aufnehmen und ihnen Nachrichten senden. Sobald seine Freunde unter dem gleichen Namen prüfen ob sie Nachrichten erhalten haben, werden sie ihnen zugestellt. Nun sind sie in der Lage ihm zu antworten oder auch von sich aus Max eine Nachricht zu hinterlassen. Max kann außerdem seinen Lieblingsblog in seine Liste aufnehmen, indem er die URL einträgt. Die Software prüft nun ob neue Einträge vorhanden sind und zeigt sie ihm ggf. an. Diese werden in einem dafür vorgesehenen Bereich in der Applikation dargestellt, ebenso wie eintreffende Nachrichten von seinen Freunden. Hierbei hat Max die Möglichkeit, der Anwendung die Erlaubnis zu erteilen selbstständig Neuerungen abzurufen und darzustellen.

1.4 Problemstellung

Das Kernproblem ist, eine nachrichtenorientierte Architektur so umzusetzen, dass sie von mehreren mobilen Geräten übers Netz zugänglich ist, ihren vollen Funktionsumfang beibehält und die Kommunikation dieser Geräte untereinander möglich ist. Üblicherweise ist eine solche Architektur nur für den Nachrichtenaustausch innerhalb einer Virtuellen Maschine ausgelegt und muss daher erweitert werden. So sind aktuelle

Lösungen meist nicht in der Lage mobile Geräte zu unterstützen und sind nur für feste Clients ausgelegt. Die Erweiterung geschieht mittels eines weiteren Dienstes, der als Schnittstelle fungiert und die Rolle der verwaltenden Komponente übernimmt. Dieser Dienst ist über ein Netzwerk erreichbar, ist bidirektional und versorgt so sein Endsystem, in diesem Fall ein JMS Provider, der üblicherweise selbst als Middleware-Komponente dient, mit den erforderlichen Daten und Parametern, die ihm vom Nutzer übermittelt werden. Ein weiteres Problem stellt das Auslesen der Blogs dar bzw. die Benachrichtigung ob neue Inhalte verfügbar sind. Dieses Problem wird jedoch in dieser Arbeit nicht näher betrachtet. Hier wird auf eine Arbeit, die in Kapitel fünf vorgestellt wird, verwiesen, die als mögliche Lösung dafür angesehen werden kann.

1.5 Einsatzbereiche der Softwarekomponenten

Der Gedanke JMS und Web Services als Kommunikationsgrundlage für einen Nachrichtenaustausch zwischen Nutzern zu verwenden, verfolgt einen neuen Ansatz. JMS ist eine Implementation einer MOM[1] und wird somit als Middlewarekomponente in Softwaresystemen eingesetzt. Dabei handelt es sich um einen Nachrichtenaustausch zwischen Softwarekomponenten, wobei eine Unterstützung von mobilen Clients nicht gegeben ist.
Web Services sind aufgrund ihrer Funktionsweise sehr gut dafür geeignet Backend-Systeme wie Datenbanken, JMS Provider oder Ähnliche, vor dem Anwender zu verbergen. Es wird lediglich eine Schnittstelle, die über das Netzwerk erreichbar ist, zur Verfügung gestellt.
Der Zugang dieser Technologien zu mobilen Geräten ist zwar vorhanden, jedoch sehr eingeschränkt. So gestaltet sich z.B. die Implementation umfangreicher, da keine WSDL Unterstützung vorhanden ist. Weiterhin spielt die Performance eine wichtige Rolle. Manche Operationen, wie das Parsen, erfordern mehr Systemressourcen als andere, wobei diese unter mobilen Geräten, zumindest älterer Generationen, nicht zur Verfügung stehen. Aus diesen Gründen haben diese Komponenten noch keinen Einzug in den mobilen Kontext gehalten, doch wäre es denkbar im Zuge der Weiterentwicklung heutiger Geräte.

[1] Message oriented Middleware

2 Architektur

2.1 Skizze und Beschreibung

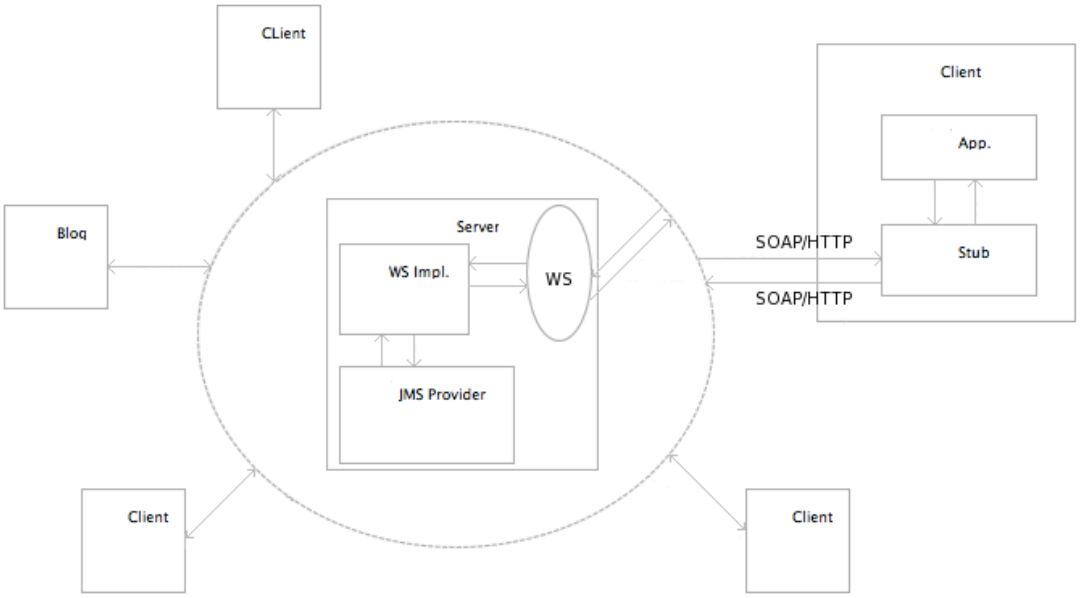

Abbildung 2.1: Architektur

Die obige Abbildung zeigt die gewählte Architektur. In der Mitte, hier als Kreis dargestellt, der Dienst und ihn umgebend die Clients. Diese können mit Hilfe des besagten Dienstes untereinander kommunizieren. Die Kommunikation zwischen Dienst und dem Client wird über SOAP und HTTP erfolgen. Der hier als "Blog"gekennzeichnete Client muss gesondert, auch hinsichtlich der Kommunikation, betrachtet werden. Er wurde nur vollständigkeitshalber in die Architektur aufgenommen. Bei näherer Betrachtung des Clients (ein Android basiertes Smartphone), zeigen sich zwei Komponenten. Zum einen die Applikation und zum anderen die Stubs, die von der Applikation genutzt werden, um die Kommunikation mit dem Dienst sicherzustellen und die erforderlichen Daten zu übertragen. Betrachtet man nun den Kreis (den Dienst) genauer, zeigen sich hier auch diverse Komponenten. Der Server stellt einen Web Server dar, der den Dienst bereitstellt und das Backend System (JMS Provider) betreibt. Der Dienst auch Web Service(WS) genannt ist in zwei Teile unterteilt. Zum einen die Schnittstelle nach *„Außen“*, die über das Netz zugänglich ist und die Implementation. Diese birgt die komplette Verarbeitungslogik. Die Kommunikation zwischen diesen beiden Komponenten ist intern geregelt. Sie sind miteinander verknüpft und werden hier nur für ein besseres Verständnis getrennt dargestellt. Die Kommunikation von der Web Service Implementation mit dem JMS System kann lokal sein, dann werden sie in einer oder mehreren Virtuellen Maschinen auf dem gleichen Rechner ausgeführt oder falls sie nicht lokal ist, wird im internen Netzwerk auf einen anderen Rechner zugegriffen, der den JMS Provider ausführt.

2.2 Softwarekomponenten, Alternativen und Entscheidungsfindung

Da man bei der Entscheidung für die Softwarekomponenten aus einem sehr großen Pool schöpfen kann, wird hier vorab ein Einblick über Alternativen und die letztendlich entscheidenden Gründe der Auswahl, gegeben. Durch die Problemstellung und die Fragestellung die es zu bearbeiten galt, waren einige Dinge schon vorgegeben. Wie z.B. das Nutzen von JMS[1], eines mobilen Betriebsystems und einer Kommunikations- und Verwaltungskomponente. Somit galt es passende Implementationen zu finden.
Vorab werden an dieser Stelle noch zwei grundlegende Dinge erwähnt. Die Entscheidung für Maven und Spring ist damit zu begründen, dass diese einen professionellen Lösungsansatz darstellen und aus der heutigen Softwareentwicklung kaum noch weg zu denken sind.
Als mobiles Betriebsystem kam nur Android in Frage, da beispielsweise JavaME

[1] Java Message Service

und Symbian recht eingeschränkt sind. Sie bieten bei weitem nicht die Auswahl an Bibliotheken um eine Entwicklung moderner Anwendungen möglich zu machen. Die Unterstützung für Smartphones mit Touch-Display ist ebenfalls nicht so umfangreich. Zudem ist Symbian ein Nokia Produkt und nur auf Geräten dieses Herstellers lauffähig. Android hingegen ist ein Open-Source Produkt und hat eine recht große Entwicklergemeinde. Hinzu kommt, dass die Verbreitung auf gängigen Geräten rasant steigt.
Bei den JMS-Provider Implementationen gab es zwar eine große Auswahl, sowohl an komerziellen als auch an Open-Source Lösungen, aber da nur Open-Source Produkte in Frage kamen und diese auch einen sehr ähnlichen Funktionsumfang boten, fiel die Entscheidung letztendlich auf ActiveMQ, ein Produkt von Apache, da noch mehrere Produkte dieser Firma genutzen werden und auch eine gute Springintegration gegeben ist.
Bei der Entscheidung für die verwaltende Komponente galt die Aufmerksamkeit den Web Services. Zum einen wegen den im nächsten Kapitel beschriebenen Vorteilen dieser Architektur und zum anderen aus pragmatischen Gründen.
Jetzt galt es noch einen passenden Webserver zu finden und eine Lösung für das Kommunikationsproblem zwischen dem Dienst und seinen Clients. Als Webserver wurde *„Jetty“*, ein Apache Produkt, genutzt. Hierbei handelt es sich um einen kleinen und schnellen Webserver, der auch einen Container für das Veröffentlichen von Webdiensten bietet und sehr verbreitet ist. Eine Integration in Maven und das Starten aus Maven heraus war ebenfalls ein sehr ansprechender Grund. Hier gibt es wiederum eine Vielzahl von Lösungen, wie z.B. Tomcat, Geronimo und viele mehr.
Das größte Problem hat die Kommunikation mit dem Web Service dargestellt. Hier gibt es einige Lösungen für Desktopanwendungen wie z.B. CXF, Axis oder .Net. Im mobilen Bereich sind z.B. ksoap und gsoap nutzbar. Vorliegend sollen mehrere mögliche Ansätze erläutert werden, einmal eine Desktop-Lösung, die auch eine WSDL Unterstützung bietet und zwei mobile Lösungen, kSoap und Rest, um auch ein nicht auf Soap basierendes Konzept zu betrachten. Die beiden Soap Lösungen wurden gewählt aufgrund ihrer Verbreitung, ihrer Integration in Spring, da sie Java basiert sind und teils auch aus pragmatischen Gründen.

3 Softwarekomponenten

3.1 SOAP

Soap ist ein Netzwerkprotokoll, das zum Austausch von Daten verwendet wird. Hierbei bedient es sich gewisser Standards wie XML zur Repräsentation der Daten und Netzwerkprotokollen wie HTTP[1]. Es stellt gewisse Regeln auf, über den Aufbau von Nachrichten, die Abbildung von Daten und gibt bestimmte Konventionen vor wie entfernte Prozeduraufrufe zu implementieren sind. Eine Soapnachricht besteht aus einem Header, der Metainformationen enthält und dem Body, in dem die eigentlichen Daten untergebracht sind. Diese referenzieren auf ein Envelope Element welches beide Teile in sich kapselt. Soap kommt immer da zum Einsatz wo es nicht zweckmässig erscheint, einen direkten Zugriff einer Anwendung beispielsweise auf eine Datenbank zu ermöglichen, sei es aus Kompatibilitäts- oder Sicherheitsgründen. Die Schnittstelle wird über die oben genannten Methoden reglementiert.
Der Nachteil von Soap ist ein hoher Rechenaufwand und ein erhöhtes Übertragungsvolumen, bedingt durch den Zusammenbau der XML-Datei und der damit verbundenen zusätzlichen Metainformationen die mit geführt werden müssen. Allerdings können, durch den flexiblen Aufbau, komplexe Aufrufe in eine Anfrage verpackt werden und müssen nicht durch mehrere einzelne realisiert werden.[Wik10]
Soap unterstützt eine Beschreibungssprache für Netzwerkdienste, die sogenannte WSDL[2]. Damit wird eine Beschreibung des jeweiligen Dienstes, seinen Funktionen, Daten und Datentypen erstellt und über das Netzwerk zugänglich gemacht. Hiermit kann eine lokale Repräsentation generiert werden, um bei der Implementation die Funktionen und ihre Datentypen zu kennen.[Met08][Pap08]

3.1.1 kSoap

Ksoap ist eine Implementierung von Soap, die auf mobilen Geräten lauffähig gemacht wurde und somit bestimmten Einschränkungen unterliegt. Dieses Framework wurde seit 2006 nicht mehr weiterentwickelt, jedoch gibt es keine in Java implementierte Lösung, die neuer wäre und einen größeren Funktionsumfang bietet. So fehlt zum

[1] Hypertext Transfer Protocol
[2] Web Services Description Language

Beispiel die Unterstützung der WSDL, eine eingeschränkte Nutzung diverser Java Standarts wie z.B. die der Stringbearbeitung und der Unterstützung aller Datentypen, aus Performance Gründen. Des Weiteren ist es in Soap erforderlich die XML-Datei komplett in den Speicher zu laden, was auf mobilen Geräten mit eingeschränkten Speicherkapazitäten oftmals nicht möglich ist. So verfolgt Ksoap einen anderen Ansatz. Hierbei kommt eine andere Art von Parser zum Einsatz, der eine speicherorientiertere Verarbeitung möglich macht. Der Nachteil ist, dass er keine Validierung von XML-Dokumenten unterstützt und es hier auch keine API für die Nutzung gibt.[Sou10b]
Wie ein typischer Aufruf aussieht und was dabei zu beachten ist, wird im nächsten Kapitel behandelt.

3.1.2 CXF

CXF ist ein Framework, welches unter anderem Soap und WSDL als Web Service Standards unterstützt. Ebenso werden die in Java 6 eingeführten Annotationen zur Definition und Generierung von Web Services unterstützt. Es wird auch als frontend Lösung, die eine Implementierung des Clients ermöglicht, eingesetzt. Hier werden beide Ansätze, sowohl Bottom Up (Code-First), der mittels POJO[3] erreicht wird, sowie Top Down(Contract-First), der durch Verwendung der WSDL erreicht wird, unterstützt. Eine Rest Unterstützung ist auch gegeben. CXF wird in Java geschrieben und von mobilen Geräten nicht unterstützt.[Wik10]
Nichtsdestotrotz wird anhand einer Implementierung demonstriert wie eine solche Lösung aussieht.

3.2 Rest

Representational State Transfer, auch Rest genannt, ist ein Architekturstil für verteilte Anwendungen. Hierbei werden Daten über HTTP übertragen ohne zusätzliche Protokolle wie SOAP einzusetzen. Daten, die von Restkonformen Web Services zur Verfügung gestellt werden, werden unter Verwendung von Ressourcen, die wiederum eindeutig einer URI zugeordnet sind, zugänglich gemacht. Auf diese Weise entsteht eine einfache und standardisierte Art um das Angebot einer möglichst großen Anzahl von Nutzern zur Verfügung zu stellen. Die Repräsentation von Ressourcen kann sowohl als HTML als auch als XML vorliegen, diese können vom Client angefordert werden, jedoch die Verwaltung obliegt dem Server. Im Gegenzug ist der Client für die Verwaltung des Anwendungszustands verantwortlich, da der Server zu keiner Zeit Informationen darüber speichert. Deswegen muss jede Anfrage sämtliche Informationen,

[3] Plain Old Java Objekt

die notwendig sind, um diese zu verstehen, enthalten. Das bedeutet, dass Rest Web Services zustandslos sind und jede Anfrage in sich geschlossen ist. Client und Server müssen keine Zustandsinformationen zwischen zwei Nachrichten speichern. Weiterhin bedient sich Rest einiger HTTP-Operationen, wie z.B. *„GET, PUT, POST, DELETE“*, um die Verarbeitung der Ressourcen zu gewährleisten.[Wik10]

3.3 Web Service

Ein Web Service, auch Webdienst genannt, ist eine Softwareanwendung die Plattformunabhängig eine Zusammenarbeit zwischen verschiedenen Anwendungen ermöglicht. Die Interaktion geschieht dabei mittels XML basierter Nachrichten, die über internetbasierte Protokolle ausgetauscht werden. Web Services sind für Softwaresysteme gedacht, die automatisiert Daten austauschen. Hierbei stellt der Client eine Anfrage und der Dienst antwortet ihm mit den gewünschen Informationen oder Daten. Sie orientieren sich an der SOA[4] und vereinen verteilte und objektorientierte Programmierstandarts. Der Anbieter veröffentlicht eine Beschreibung des Dienstes in einem Verzeichnis. Der Konsument kann über diesen Weg den Dienst finden und eine dynamische Anbindung erfolgt. Anschließend kann der Konsument die Methoden des gewählten Dienstes nutzen. Das Veröffentlichen geschieht hierbei über einen Verzeichnisdienst UDDI[5], der sich aber global nicht durchgesetzt hat. Die Beschreibung wird als WSDL, dazu gleich mehr im nächsten Punkt, hinterlegt. Als Kommunikationsprotokolle kommen SOAP, XML-RPC oder Rest zum Einsatz. Ein beliebtes Beispiel stellt hier Google dar, die mit Hilfe eines Web Services ihre Suche für andere Programme zur Verfügung stellen. Über das Ansprechen einer Schnittstelle konnte so die Suchanfrage in Fremdprogramme eingebaut oder von denen genutzt werden. Jedoch gibt es unzählige Beispiele, wie Fluggesellschaften, Reisebüros oder Online-Shops, die ihre Dienste über diesen Weg zur Verfügung stellen. Ein großer Vorteil dabei ist, die Nutzung offener Standarts, die den Einsatz vielerorts ermöglichen. Dabei ist die Nutzung von HTTP ein wichtiger Aspekt, da nur selten Probleme mit Firewalls auftreten können, anders als bei anderen vergleichbaren Technologien. Dadurch entsteht eine offene und flexible Architektur die Plattform- und Programmiersprachenunabhängig ist. Der Hauptnachteil ist sicherlich der Sicherheitsaspekt, jedoch gibt es Ansätze, sei es HTTPS, falls ausreichend oder verschiedene XML Lösungen wie XML-Signature oder XML-Encryption, um das Problem in den Griff zu bekommen.[Wik10][Met08][Pap08]

[4] Serviceorientierten Architektur
[5] Universal Description, Discovery and Integration

3.3.1 WSDL

WSDL (Web Service Description Language) ist eine XML basierte, Programmiersprachen- und Plattformunabhängige Beschreibungssprache von Netzwerkdiensten. Es ist eine Metasprache mit deren Hilfe, Daten, Datentypen, Methoden und Protokolle des jeweiligen Dienstes beschrieben werden. Es werden die Operationen, die von Außen zugänglich sind, sowie ihre Parameter und Rückgabewerte festgehalten. Das Dokument beinhaltet alle notwendigen Informationen zum Zugriff auf den Dienst. Hierbei handelt es sich allerdings um eine rein syntaktische Spezifizierung, sozusagen die Art und Weise, wie ein Client auf den entsprechenden Dienst zugreifen kann. Eine semantische Spezifikation wie z.B. Kosten, Antwortzeiten, Sicherheitsbestimmungen oder die Effekte einer Operation werden mittels der WSDL nicht beschrieben. Dadurch sind Erweiterungen wie WSDL-S entstanden.[Wik10][Met08][Pap08]

3.4 Java Message Serivce

JMS ist eine Programmierschnittstelle, die eine Implementierung einer Message Oriented Middleware darstellt und zum Senden und Empfangen von Nachrichten aus einem Client heraus, der in Java geschrieben ist. Für die Anwendung braucht man einen Provider, der die API umsetzt und somit den Dienst bereitstellt. Dieser verwaltet die Queues, Topics und Sessions. JMS ermöglicht eine lose Kopplung und eine asynchrone Kommunikation zwischen Softwarekomponenten mittels Nachrichtenaustausch. Dafür gibt es kommerzielle als auch Open-Source Produkte. Damit wird versucht die enge Kopplung anderer Kommunikationsmöglichkeiten, durch eine zwischen den Clients gelegenen Komponente, aufzubrechen. Es wird somit sichergestellt das die Clients kein näheres Wissen über die Gegenseite benötigen, was den Einsatzbereich erhöht und auch die Wartung vereinfacht. JMS unterstützt zwei Ansätze zum Versenden von Nachrichten. Zum einen die Nachrichtenwarteschlange (auch Queue genannt) für eins zu eins Verbindungen und zum anderen ein Anmelde-Versendesystem (auch Topic genannt) für eine Publish-Subscribe Kommunikation. Bei der Warteschlange werden die Nachrichten an eine Queue gesendet, an die ein Empfänger gekoppelt ist. Ist dieser nicht erreichbar, werden die Nachrichten gespeichert und der Empfänger kann sie jederzeit abrufen. Beim Anmelde-Versendesystem werden die Nachrichten an ein Topic gesendet, das von einer Vielzahl von Empfängern, die sich vorher anmelden müssen, gelesen werden kann. Ist kein Empfänger angemeldet, kann die Nachricht nicht konsumiert werden, was aber unerheblich ist. Vergleichbar mit einer Fernsehsendung. JMS unterstützt diverse Nachrichtenformate wie z.B. Textnachrichten, Bytenachrichten, Streamnachrichten,

Objectnachrichten und Mapnachrichten. Die genaue Funktionsweise und Erläuterung der Nachrichtentypen wird in einem späteren Kapitel beschrieben.[Wik10][Ter02]

3.4.1 Apache ActiveMQ

ActiveMQ stellt einen JMS Provider dar. Wie an anderer Stelle schon erwähnt, gibt es da diverse kommerzielle und Open-Source Lösungen. Dies ist eine Open-Source Lösung von Apache. ActiveMQ unterstützt zwei Betriebsmodi, den Eingebetteten und den Eigenständigen. Soll heißen, dass er im eigenständigen Modus als eigenständiger Prozess laufen kann und somit separat von irgendwelchen JMS-Client-Prozessen. Die Kommunikation erfolgt z.B. über TCP/IP[6]. Im Eingebetteten Modus läuft er somit in der gleichen Virtuellen Maschine wie der JMS-Client.[Wik10]

3.5 WebServer

Ein WebServer ist ein Computer mit der entsprechenden Webserver-Software oder auch nur die Webserver-Software selbst, der Dokumente über ein Netzwerk zur Verfügung stellt und diese an den Client, der auch über eine entsprechende Software verfügen muss, überträgt. Zur Übertragung dient das HTTP Protokoll. Dies ist standardisiert und kommuniziert üblicherweise auf Port 80. Folglich ist es auch in jeder Firewall freigegeben. Die zur Verfügung gestellten Dokumente können Dateien wie z.B. unveränderliche HTML[7] oder Bilddateien sein oder dynamisch erzeugte Seiten, die individuell, gemäß dem Profil eines eingeloggten Benutzers, erstellt werden. Solch ein Server ist über eine eindeutige IP-Adresse zu identifizieren und zu erreichen.[Wik10]

3.5.1 Jetty

Jetty ist ein solcher Webserver und Servlet-Container der wegen seiner geringen Größe auch leicht in andere Software integriert werden kann und von anderen Anwendungsservern genutzt wird, zur Verarbeitung von Servlets. Im Rahmen dieses Projekts wird er als Container genutzt, um den Webdienst öffentlich zu machen und übers Netzwerk bereit zu stellen.[Wik10]

3.6 Maven

Maven ist ein Build-Management-Tool von Apache und basiert auf Java, mit dessen Hilfe Java Programme erstellt und verwaltet werden können. Maven versucht den

[6] Transmission Control Protocol/Internet Protocol
[7] Hypertext Markup Language

Gedanken (Konvention vor Konfiguration) für den gesamten Zyklus abzubilden. Hierbei werden Softwareentwickler von der Anlage über das Kompilieren, Testen und Packen bis zum Verteilen der Software auf Anwendungsrechnern unterstützt, indem soviele Schritte wie möglich automatisiert werden. Ein Maven Projekt wird nach selbst- oder vordefinierten *„Archtypes“* erstellt, hierbei handelt es sich um die Einordnung des Projekts in einen bestimmten Kontext, wie z.B. ob es sich um eine Web-Applikation handelt oder um doch was gänzlich anderes. Abhängig davon wird eine standardisierte Verzeichnisstruktur angelegt, getreu dem Motto Konvention vor Konfiguration. Sofern sich ein Projekt daran hält, müssen keine Pfadnamen definiert werden, was die XML[8] basierte Konfigurationsdatei[9] vereinfacht. In dieser werden alle wichtigen Informationen zum Projekt gespeichert. Bei der Ausführung wird sie auf Vollständigkeit und syntaktische Gültigkeit überprüft. Zum Schluss, bevor das Projekt angelegt wird, sind noch einige Variablen fest zu legen, wie z.B. der Packetname, die Version und der Projektname. Was Maven noch auszeichnet ist die Auflösung von Abhängigkeiten. In der schon erwähnten Konfigurationsdatei werden auch Softwareabhängigkeiten angegeben, die ein Softwareprojekt zu anderen Projekten hat. Diese werden aufgelöst, indem ermittelt wird, ob sie, abhängig vom Namen und der Versionsnummer, schon im lokalen Verzeichnis (Repository) vorhanden sind. Dann werden sie beim Kompilieren aus diesem Verzeichnis verwendet ohne sie in das Projektverzeichnis zu kopieren. Anders beim Packen des Projekts, da werden alle benötigten Abhängigkeiten mit in die entsprechende, beispielsweise Jar-Datei oder Zip-Datei, kopiert. Kann nun die Abhängigkeit nicht aufgelöst werden, ist die Datei nicht im lokalen Pfad verfügbar und Maven versucht sich mit einem konfigurierten Maven-Repository im Internet zu verbinden, um diese in das lokale Verzeichnis zu kopieren, inkl. aller Unterabhängigkeiten. Es besteht in Firmen z.B. die Möglichkeit das lokale Repository auch Firmenweit über das Intranet zugänglich zu machen, um so gekaufte Bibliotheken und Frameworks allen Projekten zur Verfügung zu stellen.
Somit sind wir auch schon beim nächsten Punkt, der große Projekte in Firmen, betrifft. Da eine Aufgabenteilung beim Entwickeln einer Software in einem großen Team unabdingbar ist, unterstützt Maven das Einbinden von Unterprojekten in ein Übergeordnetes. Dabei werden die untergeordneten POMs in die des Hauptprojekts integriert. Somit ist eine Aufgabenteilung und Wiederzusammenführung am Ende ohne großen Aufwand zu bewerkstelligen. Darüber hinaus unterstützt Maven auch eine Vielzahl von Plugins die auf das Projekt anwendbar sind, beispielsweise beim *„compile, build“* und *„test“*. Hier kann z.B. ein Tool zum Erstellen von Client-Stubs aus einer WSDL eingebunden und ausgeführt werden. Es ist auch möglich ein Webserver oder

[8] Extensible Markup Language
[9] POM

JMS-Provider aus Maven heraus zu starten. Die Steuerung des Ganzen kann über die Eingabekonsole geschehen oder durch Plugins für Entwicklungsumgebungen wie Eclipse oder NetBeans. Hierbei ist meist nebst der Möglichkeit das Projekt anzulegen auch ein XML-Editor für die POM integriert, der das Erstellen und Verwalten der Einträge erleichtern soll.[Wik10][Apa10d]

3.7 Spring

Spring ist ein Framework, das sich zum Ziel gesetzt hat die Java Entwicklung zu vereinfachen, indem es ein Zusammenspiel unterschiedlicher Plattformen und Tools, angefangen von Servern über Persistenztools, wie Hibernate, die ein Objekt-Relational Mapping erlauben, bis hin zu Web-Integration, bietet. Dabei liegt das Hauptaugenmerk auf der Entkopplung der Applikationskomponente. Es gibt diverse Erweiterungen die alle innerhalb des Projekts entwickelt werden, wie Spring Web Services, Spring MVC zur Erstellung von Webanwendungen oder Spring .Net zur Portierung auf die .Net Plattform. Das Framework basiert auf folgenden Prinzipien:
Die Dependency Injection, die besagt, dass den Objekten die benötigten Ressourcen zugewiesen werden. Die Ressourcen werden in einer XML basierten Konfigurationsdatei angelegt, mittels sogenannter Beans. Hier wird z.B. ein Web Service Endpoint, eine Connection-Factory oder eine Destination für die JMS Nachrichten, definiert. Diese können dann im Java Code an beliebiger Stelle injiziert werden. Somit kann der Java Code davon freigehalten werden. So ist es zum Beispiel möglich eine Spring Bibliothek für eine Datenbankverbindung zu nutzen, die intern alle nötigen Schritte durchführt und man im Code selbst nur noch den Treiber laden oder die Ergebnisse verarbeiten muss. Das stellt eine weitere Stufe der Abstraktion zur Objekorientierten Programmierung dar. In späteren Kapiteln wird näher darauf eingegangen.
Das führt uns zur Aspektorientierten Programmierung, ein weiteres Prinzip, das eben genau das besagt. Der Code kann von technischen Aspekten, wie Transaktionen oder Sicherheit freigehalten werden, da diese isoliert werden. Weiterhin wurde versucht die Arbeit mit Programmierschnittstellen zu vereinfachen, indem Ressourcen automatisch aufgeräumt werden und Fehlersituationen einheitlich behandelt werden.[Wik10][Wal08][Wol10]

3.8 Android

Android ist ein Betriebssystem und eine Software-Plattform für Smartphones und Mobiltelefone. Basis hierfür ist ein Linux-Kernel. Er ist für die Speicherverwaltung, Prozesssteuerung und die Netzwerkkommunikation zuständig. Außerdem bildet er die Hardwareabstraktionsschicht für den Rest der Software und stellt die Gerätetreiber. Ein weiterer wichtiger Baustein ist *„Dalvik“*, eine auf Java basierende Virtuelle Maschine. Zum Programmieren von eigenen Anwendungen bietet das veröffentlichte Entwicklungssystem knapp 1500 Javaklassen und 400 Schnittstellen. Davon ist ca. jeweils ein Drittel Android spezifisch. Die VM ähnelt der normalen Java VM, da beide Byte-Code ausführen. Der wesentliche Unterschied ist die zugrundeliegende Prozessarchitektur, somit sind die Kompilate normaler Java-Compiler nicht für die Dalvik VM geeignet. Die Java VM[10] arbeitet nach dem Prinzip eines Kellerautomaten und die Dalvik VM nach dem einer Registermaschine. Heutzutage sind die meisten Prozessoren Registermaschinen, da dies oft effizienter ist, weil die CPU über schnell zugreifbare Speicherzellen (Register) verfügt. Der Zwischencode (Kellerautomatencode) wird schon zur Übersetzungszeit in den Registerautomatencode überführt. Die normale Java VM macht dies erst zur Laufzeit. Moderne Compiler generieren den Kellerautomatencode als Zwischencode, da dieser erlaubt von der Prozessarchitektur der Zielplattform zu abstrahieren.
Anwendungen für Android werden in Java geschrieben, allerdings greifen diese in geschwindigkeitskritischen Bereichen auf in C geschriebene native Bibliotheken zu. Dies ist der Fall im Bereich der Medienwiedergabe, bei der Nutzung eines Webbrowsers, der Nutzung von Grafikbibliotheken und Datenbanken. Bei der Entwicklung wird sowohl das Java-SDK als auch das Android-SDK gebraucht, weil der Quelltext erst mit dem normalen Java-Compiler übersetzt wird und danach mit einem Cross-Assembler für die Dalvik VM angepasst wird. Aus diesem Grund kann jede Java Entwicklungsumgebung genutzt werden. Das System setzt auf eine starke Modularität und somit sind alle Komponenten des Systems, ausgenommen der VM und dem Kernsystem, jederzeit austauschbar. Es ist z.B. möglich eine eigene Anwendung zu entwickeln und eine Vorhandene damit zu ersetzen. Android hat trotz seines noch nicht so hohen Alters eine sehr große Entwicklergemeinde und einen Markt für Anwendungen, der mehr als 80000 Apps umfaßt. Selbst erstellte Software kann von dem Entwickler auf dem Markt angeboten werden, da es sich hier auch um ein Open-Source Projekt handelt, das mittlerweile von Google aufgekauft wurde. Anfangs war HTC der einzige Hersteller von Mobiltelefonen, die Android einsetzten, jedoch haben sich mittlerweile einige namhafte Hersteller wie Motorola, Samsung, Sony Ericsson und Google selbst mit einem eigenen Gerät, angeschlossen. Eine Portierung von Android ist ebenfalls möglich, da der

[10] Virtuelle Maschine

Quellcode frei ist. Sobald eine ausreichende Leistungsfähigkeit des Zielsystems gegeben ist und dies einen Linux Kernel unterstützt, sind die Chancen sehr gut. Es sind bereits Portierungen für Smartphones erschienen, die ursprünglich mit einem anderen System ausgeliefert wurden oder auch für handelsübliche Desktop Computer. Auch gab es schon einen experimentellen Versuch das Ganze auf ein IPhone zu portieren.[Wik10][Dev10]

4 Entwicklung

Die Entwicklung gliedert sich in drei Teile. Einen Teil stellt der Web Service dar, der alle nötigen Funktionen implementiert um den JMS Provider mit den benötigten Informationen zu versorgen. Der zweite Teil beschreibt die API, die eine Verbindung des Clients zum Web Service ermöglicht. Hierbei werden alle generierten Stubs und benötigten Klassen, die für den Verbindungsaufbau nötig sind, in einer JAR-Datei verpackt, die im Client eingebunden werden kann. Zum Schluss der Client selbst, der in diesem Fall eine in Android geschriebene grafische Oberfläche darstellt, die es dem Benutzer ermöglicht die gewünschten Operationen auszuführen.
Als Entwicklungsumgebung wurde Eclipse verwendet, eine komfortable Umgebung um unter anderem Java Applikationen zu entwickeln. Hierfür ist lediglich die Installation des Java-SDK und Android-SDK nötig. Somit kann aus Eclipse heraus ein Android Projekt erstellt werden und alle Bibliotheken zur Entwicklung stehen zur Verfügung. Da außerdem noch Maven und Spring verwendet werden, ist die Installation dieser beiden Komponenten auch noch erforderlich. Hierfür stehen in Eclipse Plugins zur Verfügung, die den Umgang erleichtern sollen und eine Verwendung der Konsole überflüssig machen. Es wurde darauf verzichtet und weiterhin auf den Einsatz der Konsole vertraut. Um allerdings die Mavenbefehle in der Konsole bekannt zu machen muss noch die Systemvariable angepasst werden. Hierfür wird eine *„M2HOME“* Variable angelegt, diese verweist auf den Installationspfad von Maven und wird in die Systemvariable eingetragen. Weiterhin muss das lokale Mavenrepository auch in einer Variablen definiert werden. Diese Variable wird mit dem Namen *„M2REPO“* angelegt und verweist auf den Pfad des lokalen Repositorys, in der Regel *„/.m2/repository“*, falls dieser von den Standardeinstellungen nicht abweicht. Jetzt kann ein Mavenprojekt erstellt und in ein Eclipseprojekt konvertiert werden, um es anschließend in Eclipse laden zu können. Alle weiteren erforderlichen Bibliotheken werden in der Maven-Konfigurationsdatei eingetragen und aus einem online Repository ins lokale runtergeladen, wie schon an anderer Stelle beschrieben. Die Rolle von Spring wird in den folgenden Kapiteln erläutert.

4.1 Erstellung eines Mavenprojekts

Nachdem alle Installationen erfolgreich abgeschlossen sind, können alle Komponenten verwendet werden. An dieser Stelle wird die Erstellung des Mavenprojekts erläutert. Als erstes wird ein Entwicklungsverzeichnis angelegt und in dieses Verzeichnis gewechselt.

```
1 cd ~/development
```

Jetzt wird geprüft ob die Voraussetzungen für Maven auf dem System gegeben sind. Dies geschieht indem man die Version abfragt.

```
1 java -version
2 mvn --version
```

Anschließend wird das eigentliche Projekt mit Hilfe des Archtype-Plugins aufgesetzt.

```
1 mvn archetype:generate
```

An dieser Stelle erhält man eine Liste der vorhandenen Archetypen und kann einen auswählen, entsprechend einer Einordnung seines Projekts in einen bestimmten Kontext. Danach werden noch ein paar Abfragen seitens Maven ausgefüllt. In der folgenden Liste eine beispielhafte Erläuterung der einzelnen Schritte.
Auswahl:
Archtype Nummer 15 (ist ein maven-quickstart projekt)
GroupID: de.fh.koeln
ArtifactID: test
Version: 1.0-SNAPSHOT
Restliche Einstellungen können als default Werte übernommen werden. Eine kleine Erläuterung hierzu. Die GroupID legt das Java-Package fest, die ArtifactID das Verzeichnis, indem das Projekt angelegt wird. Nun wechselt man in das Verzeichnis *„test“* das eben angelegt wurde und erstellt ein Eclipse Projekt.

```
1 mvn eclipse:eclipse
```

Nun kann dieses Projekt in Eclipse importiert werden. Nach dem Start von Eclipse und dem Festlegen des Workspace kann dies unter folgendem Menüpunkt erledigt werden: *„File -> Import -> Existing Project into Workspace“*
Jetzt kann in Eclipse mit der Erstellung der gewünschten Klassen und dem Schreiben des Sourcecodes fortgefahren werden. Das Projekt kann auch unter Maven kompiliert werden, mit folgendem Befehl:

```
1 mvn compile
```

In der POM werden nun alle erforderlichen Bibliotheken in dem Tag *„dependencies“* definiert. An dieser Stelle können auch Plugins eingebaut werden wie der folgende

Ausschnitt einer POM zeigt. Hier wird das Jetty-Plugin eingebunden. Da es sich bei diesem Ausschnitt um die POM des Web Services handelt kann somit der Webserver über Maven aus der Konsole gestartet werden und der Service wird direkt veröffentlicht, da der Pfad der WAR-Datei angegeben wird.

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
       ="http://www.w3.org/2001/XMLSchema-instance"
2    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http
       ://maven.apache.org/maven-v4_0_0.xsd">
3    <modelVersion>4.0.0</modelVersion>
4    <groupId>de.fhkoeln.ba</groupId>
5    <artifactId>service</artifactId>
6    <packaging>war</packaging>
7    <version>1.0-SNAPSHOT</version>
8    <name>service Maven Webapp</name>
9    <url>http://maven.apache.org</url>
10
11   <dependencies>
12     <dependency>
13       <groupId>org.springframework</groupId>
14       <artifactId>spring</artifactId>
15       <version>2.5.5</version>
16     </dependency>
17     <dependency>
18       <groupId>org.apache.cxf</groupId>
19       <artifactId>cxf-bundle</artifactId>
20       <version>2.2.5</version>
21     </dependency>
22     <dependency>
23       <groupId>org.apache.activemq</groupId>
24       <artifactId>activemq-all</artifactId>
25       <version>5.3.0</version>
26     </dependency>
27   </dependencies>
28
29     <build>
30     <finalName>service</finalName>
31     <plugins>
```

```
32        <plugin>
33          <groupId>org.mortbay.jetty</groupId>
34          <artifactId>maven-jetty-plugin</artifactId>
35          <configuration>
36            <webApp>{basedir}/target/service.war</webApp>
37          </configuration>
38        </plugin>
39      </plugins>
40    </build>
41  </project>
```

Die WAR-Datei erhält man mit folgendem Befehl:

```
1  mvn package
```

Da es sich in diesem Fall um ein Web-Projekt handelt, wird dabei eine WAR-Datei angelegt. Alternativ kann dies abhängig von dem jeweiligen Projekt geändert werden, um beispielsweise eine ZIP oder JAR Datei zu erhalten. Abschließend kann der Server mit folgendem Befehl gestartet werden:

```
1  mvn jetty:run
```

Der Dienst ist nun, falls er richtig konfiguriert wurde, unter der URL: *„http://localhost:8080/service/ServiceName“* erreichbar, doch dazu später mehr. Bei einem Web-Projekt wird noch eine *„WEB.xml“* Datei generiert, die dem Server bekannt ist und in der man weitere Konfigurationsdateien wie z.B. die *„Spring-config.xml“* bekannt machen muss. Abschließend noch der Befehl der das Projekt aufräumt:

```
1  mvn clean
```

4.2 Spring

Hier vorab noch ein paar grundlegende Dinge zu Spring. In Java ist man gezwungen Klassen zu instantiieren und somit werden Abhängigkeiten geschaffen. Mit Spring soll dieses Problem umgangen werden, um so beispielsweise eine Entkopplung der Geschäftslogik und der Datenebene zu erreichen. Die Konfiguration wird in Spring anhand von Beans definiert. Danach muss im eigentlichen Code eine getter und setter Funktion vorhanden sein. Damit werden die Beans in den Code injiziert oder das *„Autowire“* über den Konstruktor realisiert. Spring bietet ein Annotationen basiertes Autowiring, das es ermöglicht Bean-Definitionen und Injizierungsanweisungen nicht ausschließlich über XML-Dokumente zu beschreiben, sondern auch mittels Java-Annotationen und

eigenen.

Hier ein Beispiel für eine Bean und die dazugehörigen getter und setter:

```
<bean id="myJmsTemplate" class="org.springframework.jms.core
   .JmsTemplate">
    <property name="connectionFactory">
      <bean class="org.springframework.jms.connection.
         SingleConnectionFactory">
        <property name="targetConnectionFactory" ref="
           jmsFactory" />
      </bean>
    </property>
</bean>

public JmsTemplate getTemplate() {
        return template;
    }

public void setTemplate(JmsTemplate template) {
        this.template = template;
}
```

In der Spring-Konfigurationsdatei müssen die entsprechenden Pakete, bevor sie genutzt werden können, erstmal definiert oder importiert werden. So ist es z.B. nötig, falls eine Bean für das JMS-Template definiert werden soll, die entsprechenden Bibliotheken zu laden. Hier mal ein Ausschnitt:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:jaxws="http://cxf.apache.org/jaxws"
  xmlns:soap="http://cxf.apache.org/bindings/soap"

  xsi:schemaLocation="http://www.springframework.org/schema/
     beans
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd
  default-autowire="byName">

  <import resource="classpath:META-INF/cxf/cxf.xml" />
```

```
12  <import resource="classpath:META-INF/cxf/cxf-extension-
        soap.xml" />
13  <import resource="classpath:META-INF/cxf/cxf-extension-jms
        .xml" />
```

4.3 ActiveMQ

ActiveMQ kann wahlweise als Plugin in Maven eingebunden werden und über einen Run-Befehl ähnlich wie Jetty gestartet werden oder aus einer gesonderten Installation. Hierbei liegt der Vorteil darin, dass eine Admin-Konsole über den Browser erreichbar ist, in der eine Übersicht und Verwaltung der Queues und Topics inkl. der ankommenden und abgehenden Nachrichten möglich ist. In der Übersicht kann der Nachrichtenverkehr beobachtet werden und es können bereits angelegte Queues und Topics gelöscht werden. Weiterhin können auch Nachrichten aus dieser Umgebung an den Provider gesendet werden. Es wird eine Art Liste geführt wieviele Nachrichten noch auf ihre Zustellung warten und wieviele schon zugestellt sind. Das erleichtert die Entwicklung, weil so Tests durchgeführt werden können und beobachtet werden kann ob alles wie gewünscht abläuft.

4.3.1 Callback

Nachrichtenorientierte Middleware, in diesem Fall JMS, ist eine asynchrone Kommunikationsmethode. Das bedeutet, dass der Betroffene, der eine Nachricht erhalten soll nicht zwangsläufig verfügbar sein muss. Die Nachricht wird zwischengespeichert und weitergeleitet wenn er wieder verfügbar ist. Diese Entkopplung macht JMS zu einem sehr flexiblen System. Dies wird in ActiveMQ über ein Listener-Objekt realisiert. Es kann auf einer Nachrichtenwarteschlange oder einem Topic lauschen und eine Benachrichtigung bei eintreffenden Nachrichten verschicken. Somit kann die Nachricht direkt zugestellt werden und es braucht nicht seitens des Nutzers nachgefragt werden ob Nachrichten eingetroffen sind.
Web Services hingegen sind synchron. Hierbei müssen Sender und Empfänger gleichzeitig und über die Dauer der Übertragung verfügbar sein. Mit Hilfe des Listeners kann in einem Web Service eine Reaktion erfolgen, doch leider nur zwischen dem JMS Provider und dem Service. Es ist nicht möglich durch dieses Event die Nachricht vom Web Service an den Nutzer weiter zu leiten. Ein Web Service Aufruf muss immer durch jemanden initiiert werden. Aus diesem Grund kann der Listener des JMS-Providers nicht genutzt werden.
Die Asynchronität eines JMS-Providers kann also nicht ohne Weiteres auf die Syn-

chronität eines Web Services abgebildet werden. Die Kommunikation mit dem Nutzer ist synchron. Dieser Umstand wird in Kauf genommen, um im Gegenzug den JMS-Provider, über ein Netzwerk, für Mobile Geräte nutzbar zu machen, da die Vorteile der Asynchronität eines JMS-Providers, bei einer Kombination dieser beiden Systeme, erhalten bleiben. Die Nachrichten werden weiterhin gespeichert und können seitens des Nutzers abgerufen werden, wenn dieser wieder verfügbar ist.
Eine Möglichkeit um eine ähnliche Funktion zu implementieren wäre, die Software in regelmäßigen Abständen automatisch nachfragen zu lassen, ob Nachrichten eingetroffen sind und diese ggf. zuzustellen oder mittels Web Service Benachrichtigungs-Emails zu versenden, auf die eine Anwendung adäquat reagieren kann. Ebenfalls denkbar wäre, zusätzlich eine Softwarelösung zu nutzen, die sich mit diesem Problem auseinandersetzt und die Anwendung dahingehend erweitert.

4.4 Web Service

Die Entwicklung des Web Services ist der zentrale Punkt dieser Arbeit. Hier sollen alle Funktionen die der JMS Provider nutzt, um Nachrichten anzulegen und diese abzurufen, implementiert werden, um einen Netzwerkzugriff zu ermöglichen. Der Dienst wird mit CXF erstellt. CXF unterstützt sowohl den Rest als auch den Soap Endpoint. Hierfür wird erstmal ein Interface (Schnittstelle) erstellt, wo alle Methoden inklusive ihrer Parameter und Rückgabewerte definiert werden. An dieser Stelle müssen allerdings noch einige Annotationen angelegt werden. Im folgenden Beispiel sieht man ein solches Interface mit den notwendigen Annotationen sowohl für die Klassen als auch für die Methoden und Parameter. Bei den Klassen Annotationen handelt es sich um die Pfade wo der Dienst erreichbar ist, sowohl für Rest als auch für SOAP. Bei den Annotationen für die Methoden handelt es sich um Rest-Definitionen. Die Parameter-Annotationen sind von SOAP. Mit den Rest Annotationen wird festgelegt, welche Methode beim Post und Get genutzt werden soll und Soap Annotationen dienen zur Parameterbenennung um diese im Aufruf auch mit dem entsprechenden Namen anzusprechen. Dieses Interface wird in der Web Service Implementation eingebunden und somit ist es erforderlich alle Methoden genau so zu implementieren. Das Interface wird bei einem Zugriff auf den Dienst genutzt und dient auch zur WSDL Generierung. Somit ist eine Entkopplung der eigentlichen Implementation gegeben und diese ist vom Anwender versteckt.

```
1 @Path("/restService")
2 @WebService(targetNamespace="http://ba.fhkoeln.de/service/")
3 public interface IRemoteService {
```

```
4
5   @POST
6     @Produces("application/xml")
7   void sendMsg(@WebParam(name="msg")String msg,@WebParam(
       name="id")String id,@WebParam(name="name")String name,
       @WebParam(name="flag")String flag);
8
9   @GET
10    @Produces("application/xml")
11  String getMsg(@WebParam(name="id")String id,@WebParam(name
       ="name")String name,@WebParam(name="flag")String flag);
```

Die Web Service Implementation beinhaltet die ganze Verarbeitungslogik und zusätzlich noch einige benötigte Annotationen. Hier werden z.B. die Parameter-Annotationen für Rest definiert und die Methodenen-Annotation für SOAP. Auch hier wieder um sie mit dem entsprechenden Namen nutzen zu können. Zusätzlich wird hier noch der Endpoint für SOAP definiert. Dieser findet sich in den Web Service Beans wieder. Dazu später mehr. Außerdem wird der Pfad für den Rest Service definiert (*„@Path“*). Der folgende Auszug aus dem Service-Code zeigt, was bei dem Aufruf der *„sendMsg“* Methode ausgeführt wird. Hier werden abhängig davon, ob die Nachricht in eine Warteschlange oder in ein Topic geschrieben werden soll, die entsprechenden Producer oder Consumer beim Auslesen der Nachricht aufgerufen.

```
1  @Path("/restService")
2  @WebService(endpointInterface = "de.fhkoeln.ba.
      IRemoteService", targetNamespace = "http://ba.fhkoeln.de/
      service/")
3  public class RemoteService implements IRemoteService {
4
5    public ApplicationContext context;
6    protected JmsConsumer consumer;
7    protected JmsProducer producer;
8    protected TopicConsumer consumer2;
9    protected TopicProducer producer2;
10
11   @POST
12     @Path("set")
13   @WebMethod(action = "sendMsg")
```

```
14   public void sendMsg(@QueryParam("msg") String msg,
        @QueryParam("id")String id,@QueryParam("name")String
        name,@QueryParam("flag")String flag) {
15
16     context = AppContext.getApplicationContext();
17     if (Boolean.valueOf(flag)) {
18       producer2 = (TopicProducer) context.getBean("producer2
           ");
19       try {
20         producer2.start(msg, id, name);
21         tearDown();
22       } catch (JMSException e) {
23         e.printStackTrace();
24       } catch (Exception e) {
25         e.printStackTrace();
26       }
27     }
28     else{
29       producer = (JmsProducer) context.getBean("producer");
30       try {
31         producer.start(msg, id);
32         tearDown();
33       } catch (JMSException e) {
34         e.printStackTrace();
35       } catch (Exception e) {
36         e.printStackTrace();
37       }
38     }
39   }
```

Der Applicationcontext wird beim ersten Laden der Bean initialisiert und wird über eine Klasse immer wieder aufgerufen wenn er gebraucht wird, um sicher zu stellen, dass immer der Gleiche genutzt wird. Im Producer oder Consumer wird die Verbindung zum JMS-Provider hergestellt und die Nachricht abgelegt oder empfangen. Die Identifikation erfolgt über eine ID und somit wird sichergestellt, dass die Nachricht immer nur dem richtigen Empfänger zugestellt wird.

```
1  public class JmsProducer{
2
```

```
3  private JmsTemplate template;
4  private Destination destination;
5  private Connection connect;
6  private Session session;
7  private  MessageProducer producer;
8
9  public void start(final String msg, String id) throws
       JMSException {
10
11 ConnectionFactory fact = (ConnectionFactory) template.
       getConnectionFactory();
12 connect = fact.createConnection();
13 session = connect.createSession( false, TopicSession.
       AUTO_ACKNOWLEDGE );
14 producer = session.createProducer(destination);
15 connect.start();
16 TextMessage message = session.createTextMessage();
17 message.setStringProperty("next", id);
18 message.setText( msg );
19 producer.send( message );
20 }
21 }
```

Es handelt sich hier um einen hierarchischen Aufbau. Als erstes wird eine ConnectionFactory mit dem definierten Template erstellt, anschließend wird die Verbindung angelegt. Jetzt da die Verbindung existiert, kann eine Session definiert werden und abhängig davon wird ein Producer angelegt, dem eine Destination übergeben wird. Nun kann die Verbindung gestartet werden und über ein Message Objekt vom jeweiligen Typ kann die gewünschte Nachricht erstellt werden. Wenn dies alles geschehen ist, kann der vorher definierte Producer die Nachricht absenden, die wiederum beim JMS-Provider in eine Queue oder ein Topic geschrieben wird. Analog dazu wird die Nachricht auch ausgelesen und weitergeleitet. Das Template, die Destination, der Producer und Consumer werden in den Spring Beans konfiguriert und müssen lediglich über getter und setter in den Code injiziert werden. Dazu später mehr.

Bei der Verbindung besteht noch die Möglichkeit die Spring Klassen zu nutzen und direkt über das Template zu senden. Das reduziert den Code, aber die Verarbeitung ist komplett intern und man hat keinen Zugriff. Der Aufbau ist ähnlich, jedoch kapselt Spring den Zugriff auf die unter dem Template liegenden Schichten.

4.4.1 Annotation

Als Annotation wird im Zusammenhang mit der Programmiersprache Java ein Sprachelement bezeichnet, das die Einbindung von Metadaten in den Quelltext erlaubt. Ein Annotation Prozessor ist ein Compiler-Plugin, das Annotationen beim Kompilieren auswerten kann, um damit Warnungen und Fehlermeldungen zu unterdrücken oder auszulösen oder weiteren Quellcode oder andere Dateien zu generieren. Eingesetzt werden Annotationen unter anderem im Java-EE-Umfeld, um Klassen um Informationen zu erweitern, die vor Java 5 in separaten Dateien hinterlegt werden mussten.[Wik10][Ora10] Wie oben schon beschrieben, dienen die Annotationen der Konfiguration des Web Services und ermöglichen so die Definition einiger grundlegender Dinge. Wie z.B. unter welchem Namen die Methoden bekannt gemacht werden und somit erreichbar sind. So sagt die Methoden-Annotation *„@WebMethod(action = sendMsg)“* aus, dass in SOAP die Methode zum Senden von Nachrichten *„sendMsg“* heißt und ihre Parameter werden über *„@WebParam(name="msg") String msg“* definiert. Zusätzlich zum Typ wird auch der Name dieses Parameters festgelegt und ist in der WSDL auch dementsprechend gekennzeichnet. Standardmässig würde da *„arg0, arg1“* etc. stehen. Das wäre nicht gut lesbar. Genauso wird im Rest auch verfahren. Hier werden über *„@POST, @Path(set)“* und *„@QueryParam("msg")“* die Parameter und Methoden definiert, die beim Aufruf genutzt werden können. Wie später noch erläutert, wird im Aufruf beim Wegfall des *„@QueryParam“* Parameters die Information nicht im Header sondern im Body übergeben.

4.4.2 Namespace

Zusätzlich zu den Annotationen für die Methodenaufrufe und die Parameter wird noch der Namespace definiert. Sowohl in SOAP als auch im Rest. Im Rest wird mit *„@Path("/restService")“* beschrieben, dass der Dienst unter der weiter oben genannten URL plus dem hier definierten Pfad zu erreichen ist. Im SOAP sieht das folgendermaßen aus: *„@WebService(endpointInterface = "de.fhkoeln.ba.IRemoteService", targetNamespace = "http://ba.fhkoeln.de/service/")“*. Hier wird zusätzlich noch ein Endpoint angegeben unter dem später die WSDL zu finden ist. Der verweist auch auf das Interface des Web Services und nicht auf die Implementation des Dienstes.

4.4.3 Beans

In den folgenden Beans, die aus dem Web Service stammen, werden die benötigten Klassen geladen. Sie dienen als Konfiguration und Instanziierung der im Code verwendeten Objekte wie z.B. dem Template. Somit wird der Code davon freigehalten und sie

müssen lediglich zur Laufzeit durch getter und setter geladen werden. Damit kann auch der Endpoint des Web Services gesetzt werden und auf die Interface Klasse verwiesen werden. Der Aufbau der Bean sieht wie folgt aus: Es muss ein ID Element vorhanden sein, das den Namen der Bean definiert. Über diesen kann sie angesprochen werden. Danach wird die Klasse, auf die verwiesen wird, definiert. Jetzt können zusätzlich noch Eigenschaften angelegt werden, die auf zusätzliche Klassen verweisen oder eine Referenz einer anderen Bean enthalten.

```
1  <bean id="contextApplicationContextProvider" class="de.
      fhkoeln.ba.ApplicationContextProvider"></bean>
2  <bean id="remoteService" class="de.fhkoeln.ba.
      RemoteService"></bean>
3
4  <!-- WSEndpoint -->
5  <jaxws:endpoint id="jmsservice" implementor="#
      remoteService"
6    address="/WebService" />
7
8  <!-- JAX-RS -->
9  <jaxrs:server id="restjmsService" address="/">
10   <jaxrs:serviceBeans>
11     <ref bean="remoteService" />
12   </jaxrs:serviceBeans>
13 </jaxrs:server>
14
15 <!-- JMS ConnectionFactory to use -->
16 <bean id="jmsFactory" class="org.apache.activemq.
      ActiveMQConnectionFactory">
17   <property name="brokerURL" value="tcp://localhost:61616"
        />
18 </bean>
19
20 <!-- Spring JMS Template -->
21 <bean id="myJmsTemplate" class="org.springframework.jms.
      core.JmsTemplate">
22   <property name="connectionFactory">
23     <bean class="org.springframework.jms.connection.
         SingleConnectionFactory">
```

```
24          <property name="targetConnectionFactory" ref="
               jmsFactory" />
25        </bean>
26      </property>
27    </bean>
28
29    <bean id="consumerJmsTemplate" class="org.springframework.
         jms.core.JmsTemplate">
30      <property name="connectionFactory" ref="jmsFactory" />
31    </bean>
32
33    <!-- a sample POJO which uses a Spring JmsTemplate -->
34    <bean id="producer" class="de.fhkoeln.ba.JmsProducer">
35      <property name="template" ref="myJmsTemplate" />
36      <property name="destination" ref="destination" />
37    </bean>
38
39    <!-- a sample POJO consumer -->
40    <bean id="consumer" class="de.fhkoeln.ba.JmsConsumer">
41      <property name="template" ref="consumerJmsTemplate" />
42      <property name="destination" ref="destination" />
43    </bean>
44
45    <!-- Destination -->
46    <bean id="destination" class="org.apache.activemq.command.
         ActiveMQQueue"
47      autowire="constructor">
48      <constructor-arg value="myQueue" />
49    </bean>
50
51    <!--Topic-->
52    <bean id="producer2" class="de.fhkoeln.ba.TopicProducer">
53      <property name="template" ref="myJmsTemplate" />
54      <property name="destination" ref="destination2" />
55    </bean>
56
57    <bean id="consumer2" class="de.fhkoeln.ba.TopicConsumer">
58      <property name="template" ref="consumerJmsTemplate" />
```

```
59      <property name="destination" ref="destination2" />
60    </bean>
61
62    <bean id="destination2" class="org.apache.activemq.command
         .ActiveMQTopic">
63    </bean>
```

4.4.4 MSG Typen

JMS unterstützt verschiedene Nachrichtentypen wie in einem anderen Kapitel beschrieben. Bevor eine Nachricht an den JMS Provider gesendet werden kann, muss diese einem bestimmten Typen zugeordnet werden. Hier werden exemplarisch Text- und Bytenachrichten betrachtet. Die Methoden des Web Services sind nach Nachrichtentypen gegliedert. Es gibt also für jeden Typ (Text/Byte), jeweils eigene Methoden. Die Producer und Consumer dagegen sind in Topic und Queue unterteilt. Textnachrichten werden als Strings abgebildet. Der Eingabetext wird in eine Stringvariable eingelesen und bei der Übergabe in ein Objekt des Typs TextMessage geschrieben. Dies wird im Anschluss, durch den jeweiligen Producer, in die Queue oder das Topic geschrieben. Ganz analog dazu geht das Auslesen der Nachricht, durch den jeweiligen Consumer, von statten. Hier als Beispiel der Aufbau einer dieser Methoden:

```
1   @GET
2     @Path("get")
3   @WebMethod(action = "getMsg")
4   public String getMsg(@QueryParam("id")String id,
       @QueryParam("name")String name,@QueryParam("flag")
       String flag) {
5
6     context = AppContext.getApplicationContext();
7
8     if(Boolean.valueOf(flag)){
9       consumer2 = (TopicConsumer) context.getBean("consumer2
           ");
10      try {
11        consumer2.start(name, id);
12        TextMessage msg = (TextMessage) consumer2.onMessage
             ();
13        String respMsg = "nix da";
14            if (msg != null){
```

```
15                respMsg = msg.getText();
16            }
17        tearDown();
18            return respMsg;
19
20      } catch (JMSException e) {
21        e.printStackTrace();
22      } catch (Exception e) {
23        e.printStackTrace();
24      }
25    }
26    else{
27      consumer = (JmsConsumer) context.getBean("consumer");
28      try {
29        consumer.start(id);
30        TextMessage msg = (TextMessage) consumer.onMessage()
             ;
31        String respMsg = "nix da";
32            if (msg != null){
33               respMsg = msg.getText();
34            }
35        tearDown();
36            return respMsg;
37
38      } catch (JMSException e) {
39        e.printStackTrace();
40      } catch (Exception e) {
41        e.printStackTrace();
42      }
43    }
44    return null;
45  }
```

Bei Bytenachrichten werden die Daten in einem ByteArray gespeichert und dieses wird dann im Anschluss dem BytesMessage Objekt zugewiesen und durch die Producer in die Queue geschrieben. Die Methode des Web Services sieht ganz analog zur Textnachricht aus, lediglich der Producer hat eine separate Methode erhalten, die als Übergabe ein ByteArray statt einem String benötigt. Bei dem Consumer kann die gleiche Methode

genutzt werden, da dieser ein Objekt des übergeordneten Typs *„Message“* als Rückgabe hat, das erst im Web Service zugeordnet wird. Dargestellt in der *„onMessage()“* Methode die das Ergebnis liefert.

```
1  public class JmsConsumer {
2  private JmsTemplate template;
3  private Destination destination;
4  private Connection connection;
5  private Session session;
6  private MessageConsumer consumer;
7  private Message message;
8
9  public void start(String id) throws JMSException {
10 String selector = "next = '" + id + "'";
11
12 ConnectionFactory factory = template.getConnectionFactory();
13 connection = factory.createConnection();
14
15 connection.start();
16
17 session = connection.createSession(false, Session.
      AUTO_ACKNOWLEDGE);
18 consumer = session.createConsumer(destination, selector);
19 }
20
21 public Message onMessage() throws JMSException {
22 message = consumer.receive(2000);
23 return message;
24 }
```

An der Web Service Methode zum Senden hat sich bis auf den Typ nichts geändert. Beim Empfang kann der Byteinhalt nicht direkt aus dem Messagetyp ausgelesen werden wie bei TextMessages. Text- und BytesMessages enthalten außer der eigentlichen Nachricht noch eine Vielzahl von Parametern, wie z.B. Queuename und ID. Mit der *„getText()“* Methode kann die eigentliche Nachricht aus einer TextMessage ausgelesen werden. Bei Bytes muss ein kleiner Umweg gegangen werden. Hier kann entweder eine Iteration über die Länge der Bytenachricht durchlaufen werden und den Inhalt in ein Array schreiben oder es kann die Methode *„getBytes(byte [])“* benutzt werden. Dabei

wird auf die Iteration verzichtet, jedoch muss ein temporäres Array von der Länge der Bytenachricht erstellt werden. In dieses werden die Daten ausgelesen.

```
1  @GET
2  @Path("getb")
3  @WebMethod(action = "getByteMsg")
4  public byte[] getByteMsg(@QueryParam("id") String id,
       @QueryParam("name") String name, @QueryParam("flag")
       String flag)
5  throws Exception {
6      context = AppContext.getApplicationContext();
7      if(Boolean.valueOf(flag)){
8        consumer2 = (TopicConsumer) context.getBean("consumer2
           ");
9        try {
10         consumer2.start(name, id);
11         BytesMessage msg = (BytesMessage) consumer2.
             onMessage();
12         byte[] respMsg = null;
13             if (msg != null){
14               byte[] bmsg = new byte[(int) msg.getBodyLength
                   ()];
15               msg.readBytes(bmsg);
16               respMsg = bmsg;
17             }
18         tearDown();
19             return respMsg;
20
21       } catch (JMSException e) {
22         e.printStackTrace();
23       } catch (Exception e) {
24         e.printStackTrace();
25       }
26     }
```

4.4.5 JMS-Connector

Es gibt verschiedene Möglichkeiten sich zu dem JMS Provider zu verbinden. In den folgenden Unterkapiteln wird eine Übersicht darüber gegeben. Die Entscheidung fiel

letztendlich auf die ActiveMQ Klassen, da eine allgemeinere Implementation angestrebt war, um nicht nur über CXF, sondern auch mit Rest und kSoap mit dem Dienst kommunizieren zu können. Außerdem stammen diverse Lösungen aus älteren Versionen, zwischenzeitlich sind bessere und einfachere Lösungen entstanden.

4.4.5.1 CXF JMS Transport

CXF bietet ein Transport-Plugin das Endpunkten die Verwendung von Java Message Service (JMS)-Warteschlangen und Themen ermöglicht. CXF's JMS Plugin verwendet das JNDI[1] um Hinweise auf die JMS-Provider, Makler für die JMS-Destinationen zu erhalten. Sobald CXF eine Verbindung zu einem JMS-Provider aufgebaut hat, unterstützt CXF die Weitergabe von Nachrichten entweder als JMS ObjectMessage oder als JMS TextMessage verpackt. Hierbei mußte die JMS Konfiguration noch in den Web Service Endpoint integriert werden. Das„ *jms:conduit*“ Element enthielt die Konfiguration eines Consumer Endpoint, die „ *jms:destination*“ die eines Provider Endpoints.[Apa10c]

```
1  <beans xmlns="http://www.springframework.org/schema/beans"
2         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3         xmlns:ct="http://cxf.apache.org/configuration/types"
4         xmlns:jms="http://cxf.apache.org/transports/jms"
5         xsi:schemaLocation="http://www.springframework.org/
             schema/beans http://www.springframework.org/schema
             /beans/spring-beans.xsd
6                            http://cxf.apache.org/jaxws http
                                :://cxf.apache.org/schemas/
                                jaxws.xsd
7                            http://cxf.apache.org/transports/
                                jms http://cxf.apache.org/
                                schemas/configuration/jms.xsd
                                ">
8  <jms:conduit name="{http://cxf.apache.org/jms_endpt}
      HelloWorldJMSPort.jms-conduit">
9    <jms:address destinationStyle="queue"
10                jndiConnectionFactoryName="
                     myConnectionFactory"
11                jndiDestinationName="myDestination"
12                jndiReplyDestinationName="myReplyDestination"
```

[1] Java Naming and Directory Interface

```
13                connectionUserName="testUser"
14                connectionPassword="testPassword">
15     <jms:JMSNamingProperty name="java.naming.factory.initial
          "
16                 value="org.apache.cxf.transport.jms.
                      MyInitialContextFactory"/>
17     <jms:JMSNamingProperty name="java.naming.provider.url"
18                 value="tcp://localhost:61616"/>
19   </jms:address>
20 </jms:conduit>
21 </beans>
```

4.4.5.2 CXF JMS-Config Feature

Mit CXF Version 2.0.9 und 2.1.3 wurde eine einfachere und flexiblere Methode implementiert, die konformer zu der Spring Dependency Injection ist. Zusätzlich hat die neue Konfiguration mehr Möglichkeiten. Zum Beispiel ist es nicht mehr nötig JNDI zu nutzen, um die Connection-Factory aufzulösen. Stattdessen kann es in der Spring-config definiert werden. Der Endpoint enthält nun einen Verweis auf eine Spring-Config Bean, die wiederum eine Connection-Factory und eine Target-Destination implementieren muss.[Apa10c]

```
1  <jaxws:endpoint
2    xmlns:customer="http://customerservice.example.com/"
3    id="CustomerService"
4    address="jms://"
5    serviceName="customer:CustomerServiceService"
6    endpointName="customer:CustomerServiceEndpoint"
7    implementor="com.example.customerservice.impl.
        CustomerServiceImpl">
8    <jaxws:features>
9      <bean class="org.apache.cxf.transport.jms.
          JMSConfigFeature"
10       p:jmsConfig-ref="jmsConfig" />
11   </jaxws:features>
12 </jaxws:endpoint>
13
14  <bean id="jmsConfig" class="org.apache.cxf.transport.jms.
      JMSConfiguration"
```

```
15   p:connectionFactory-ref="jmsConnectionFactory"
16   p:targetDestination="test.cxf.jmstransport.queue"
17 />
18
19  <bean id="jmsConnectionFactory" class="org.springframework.
       jms.connection.SingleConnectionFactory">
20   <property name="targetConnectionFactory">
21     <bean class="org.apache.activemq.
          ActiveMQConnectionFactory">
22       <property name="brokerURL" value="tcp://localhost
            :61616" />
23     </bean>
24   </property>
25 </bean>
```

4.4.5.3 SOAP over JMS specification

SOAP over JMS Messaging bietet einen alternativen Mechanismus zu SOAP über HTTP. SOAP over JMS-Spezifikation beschreibt eine Reihe von Standards für den Transport von SOAP-Nachrichten über JMS. Der Hauptzweck ist die Interoperabilität zwischen den verschiedenen Web-Services-Anbieter Implementierungen zu gewährleisten. SOAP over JMS Transport unterstützt die meisten Konfigurationen von JMS Transport und stellt einige Erweiterungen zur Verfügung, die die SOAP over JMS-Spezifikation unterstützen. SOAP over JMS Transport nutzt die URI um JMS Adressierungsinformationen zu beschreiben und bietet neue Erweiterungen für die WSDL JMS-Konfiguration. Hierbei wurde keine Springintegration gefunden. Es kann entweder über die WSDL oder über Java Code implementiert werden.[Apa10c]

```
1  <wsdl11:binding name="exampleBinding">
2    <soapjms:jndiContextParameter name="name" value="value" />
3  <soapjms:jndiConnectionFactoryName>ConnectionFactory
4  </soapjms:jndiConnectionFactoryName>
5  <soapjms:jndiInitialContextFactory>
6  org.apache.activemq.jndi.ActiveMQInitialContextFactory
7  </soapjms:jndiInitialContextFactory>
8  <soapjms:jndiURL>tcp://localhost:61616
9  </soapjms:jndiURL>
10 <soapjms:deliveryMode>PERSISTENT</soapjms:deliveryMode>
11 <soapjms:priority>5</soapjms:priority>
```

```
12    <soapjms:timeToLive>200</soapjms:timeToLive>
13  </wsdl11:binding>
14
15  <wsdl11:service name="exampleService">
16    <soapjms:jndiInitialContextFactory>
17      com.example.jndi.InitialContextFactory
18    </soapjms:jndiInitialContextFactory>
19    <soapjms:timeTolive>100</soapjms:timeToLive>
20    ...
21    <wsdl11:port name="quickPort" binding="tns:exampleBinding
         ">
22      ...
23      <soapjms:timeToLive>10</soapjms:timeToLive>
24    </wsdl11:port>
25    <wsdl11:port name="slowPort" binding="tns:exampleBinding">
26      ...
27    </wsdl11:port>
28  </wsdl11:service>
```

4.4.5.4 Camel

Seit CXF 2.1.3 gibt es eine neue Art der Konfiguration von JMS (Verwenden des JMSConfigFeature). Es macht die JMS Konfiguration für CXF so einfach wie mit Camel. Camel für JMS hat sich bewährt, falls man die umfangreichen Funktionen für das Routing von Camel und andere Integrationsszenarien, die von CXF nicht unterstützt werden, verwenden möchte. Bei der Verwendung von Camel muss ein Camel Endpoint definiert werden. Da hier eine neue Softwarekomponente notwendig gewesen wäre, wurde darauf verzichtet auf diese Weise zu implementieren.[Apa10b]

4.4.5.5 Spring JMS

Das Spring Framework bietet auch eine API[2] um JMS zu nutzen. Hier wird auf die Spring Klassen zurückgegriffen, beispielsweise für das Template. Die Konfiguration ist ähnlich der von uns genutzten und unterscheidet sich nur in einigen Bean Definitionen und der genutzten Klassen.[Sou10a]

```
1  <bean id="activeMQConnectionFactory"
2    class="org.apache.activemq.ActiveMQConnectionFactory">
```

[2]Application Programming Interface

```
3   <property name="brokerURL" value="tcp://jmsserver:9999"/>
4 </bean>
5
6 <bean id="sampleQueue"
7 class="org.apache.activemq.command.ActiveMQQueue">
8 </bean>
9
10 <bean id="sampleTopic"
11 class="org.apache.activemq.command.ActiveMQTopic">
12
13 <bean id="sampleJmsTemplate"
14 class="org.springframework.jms.core.JmsTemplate">
15 <property name=" connectionFactory" ref="
     activeMQConnectionFactory" />
16 </bean>
17
18 <bean id="messageReceiver"
19 class=" net.javabeat.spring.jms.MessageReceiver">
20 <property name="jmsTemplate" ref=" sampleJmsTemplate" />
21 <property name="destination" ref=" sampleTopic" />
22 </bean>
23
24 <bean id="messageSender"
25 class=" net.javabeat.spring.jms.MessageSender">
26 <property name="jmsTemplate" ref=" sampleJmsTemplate" />
27 <property name="destination" ref=" sampleTopic" />
28 </bean>
```

Spring unterstützt eine handliche Abstraktion, JMSTemplate, mit dem einige der unteren JMS Detail Ebenen ausgeblendet werden können, welches das Senden von Nachrichten erlaubt. Das JMSTemplate erstellt eine neue Verbindung, eine Session und einen Produzenten für jede gesendete Nachricht, um sie im Anschluss wieder zu schließen.

```
1 public class MessageSender
2 {
3     private Destination destination;
4     private JmsTemplate jmsTemplate;
5
```

```
6       public MessageSender() {}
7
8       public void setJmsTemplate(JmsTemplate jmsTemplate)
9       {
10          this.jmsTemplate = jmsTemplate;
11      }
12
13      public void setDestination(Destination destination)
14      {
15          this.destination = destination;
16      }
17
18      public void sendMessage()
19      {
20          MessageCreator creator = new MessageCreator()
21          {
22              public Message createMessage(Session session)
23              {
24                  TextMessage message = null;
25                  try
26                  {
27                      message = session.createTextMessage();
28                      message.setStringProperty("text", "Hello
                            World");
29                  }
30                  catch (JMSException e)
31                  {
32                      e.printStackTrace();
33                  }
34                  return message;
35          }
36      };
37      jmsTemplate.send(destination, creator);
38      }
39  }
```

4.4.5.6 ActiveMQ

Die JMS Connection-Factory für ActiveMQ erlaubt eine Verbindung zu einem remote Broker mit spezifischem Host-Namen und Port.[Apa10a] Die Konfiguration der Beans und die Java Implementation sehen ähnlich wie bei der Spring Methode aus. Wesentlicher Unterschied ist, dass nicht über das Template gesendet wird. Damit werden die unteren Schichten nicht versteckt ausgeführt, man hat Möglichkeiten den einzelnen Schichten weitere Details zu zu ordnen und es dient aus Demonstrationszwecken einem besseren Verständnis. Folgende Beans müssen definiert werden:
Die ConnectionFactory, die die Broker URL[3] enthält:

```
1 <bean id="jmsFactory" class="org.apache.activemq.
     ActiveMQConnectionFactory">
2     <property name="brokerURL">
3       <value>tcp://localhost:61616</value>
4     </property>
5   </bean>
```

Die Templates, die die ConnectionFactory implementieren:

```
1   <bean id="myJmsTemplate" class="org.springframework.jms.
       core.JmsTemplate">
2     <property name="connectionFactory">
3       <bean class="org.springframework.jms.connection.
          SingleConnectionFactory">
4         <property name="targetConnectionFactory" ref="
            jmsFactory" />
5       </bean>
6     </property>
7   </bean>
8
9   <bean id="consumerJmsTemplate" class="org.springframework.
       jms.core.JmsTemplate">
10    <property name="connectionFactory" ref="jmsFactory" />
11  </bean>
```

Des Weiteren sind noch die Producer und Consumer Bean, sowie die Destination, aufzuführen:

```
1 <bean id="producer" class="de.fhkoeln.ba.JmsProducer">
```

[3]Uniform Resource Locator

```
    <property name="template" ref="myJmsTemplate" />
    <property name="destination" ref="destination" />
  </bean>

  <bean id="consumer" class="de.fhkoeln.ba.JmsConsumer">
    <property name="template" ref="consumerJmsTemplate" />
    <property name="destination" ref="destination" />
  </bean>

  <bean id="destination" class="org.apache.activemq.command.
     ActiveMQQueue"
    autowire="constructor">
    <constructor-arg value="myQueue" />
  </bean>
```

Der Endpoint für den Web Service muss auch noch als Bean definiert werden.

```
<jaxws:endpoint id="jmsservice" implementor="#remoteService"
    address="/WebService" />
```

Da nun nicht über das Template gesendet wird, wurde die Producer Klasse angelegt. Diese wird aus der Web Service Methode aufgerufen.

```
  TopicConnectionFactory fact = (TopicConnectionFactory)
     template.getConnectionFactory();

  connect = fact.createTopicConnection();
  session = connect.createTopicSession( false, TopicSession.
     AUTO_ACKNOWLEDGE );

  Topic topic = new ActiveMQTopic(name);
  sender = session.createPublisher( topic );
  connect.start();

  TextMessage message = session.createTextMessage();
  message.setStringProperty("next", id);
  message.setText( msg );

  sender.publish( message );
```

4.4.6 Probleme

Bis auf die üblichen Probleme, die bei der Entwicklung auftreten, gab es bis zu diesem Stand keine größeren Schwierigkeiten. Die Suche nach einer geeigneten Lösung und Entscheidung welche genutzt wird und welche funktioniert, gestaltete sich als umfangreich. Danach gelang die Implementation der JMS Funktionalitäten und die Integration in den Web Service recht reibungslos. Für Textnachrichten zumindest. Bei ObjectMessages und Bytenachrichten gestaltete sich die Sache etwas schwieriger als gedacht. Deswegen wurde aus Zeitgründen auf eine Implementation der ObjectMessages verzichtet.
Die Probleme gliederten sich wie folgt: Als Vorbedingung gilt, dass die Objekte serialisierbar sein müssen. Bei Bytenachrichten wird dies in Soap z.B. durch die Base64-Kodierung erfüllt. Dabei entsteht wieder ein String-Objekt welches serialisierbar ist. In Rest wird ein anderes Stream-Objekt genutzt. Somit ist es möglich in Rest jedes Objekt, das über einen Stream übertragen werden kann, zu senden oder es zu empfangen. Beim Senden von Objektnachrichten in Rest, bei denen Java-Objekte verschickt werden, ist dies durch die Ableitung von der Klasse *„Serializable“* getan. Damit ist das Senden problemlos möglich. Beim Empfang allerdings kann nicht festgestellt werden, um was für ein Objekt es sich handelt oder um welchen Dateityp bei Bytenachrichten, um die einwandfreie Wiederherstellung zu gewährleisten. Es müßte der MIME[4]-Type festgestellt werden. Dies könnte über ein DOM-Objekt[5] geschehen, welches verschickt wird. Bei dieser Methode wird eine XML-Datei erstellt, in der man z.B. den Dateinamen, den MIME-Type und ggf. zusätzliche Eigenschaften angeben kann, um diese beim Parsen auszulesen.

1. Direktes Senden / Empfangen von Dateien:
Vorteil: es wird direkt von der Platte gelesen / auf die Platte geschrieben (und durch die Input- Outputstreams geleitet); das Objekt muss nicht im Speicher des Clients aufgebaut werden
Nachteil: die Informationen zur Datei gehen verloren (man könnte allerdings einige ungenutzte Attribute der JMS Message nutzen um dem vorzubeugen, das System wird dadurch aber unflexibel)

2. Verwenden von DOM:
(*„dom4j“* bietet den besten Funktionsumfang und dadurch den geringsten Programmieraufwand)

[4]Internet Media
[5]Document Object Model

Vorteil: alle benötigten Informationen sind im Objekt enthalten (sehr flexibel); die XML Parser sind für fast alle Plattformen vorhanden
Nachteil: Das Objekt muss im Speicher des Clients aufgebaut werden, um es zu versenden bzw. um es beim Empfang zu Parsen. Somit werden die Daten und Informationen wieder extrahiert. (XML Doc's sind Texte, dass heißt, alles muss in Strings umgewandelt werden)
Die Verwendung von DOM-Objekten ist allerdings nur in Rest möglich. Somit muss in Soap auf JAXB[6] zurückgeriffen werden, um die Objekte in die Soap-Envelope schreiben zu können. Diese unterstützt nur Grunddatentypen (int, boolean, string...). Alternativ kann Soap mit Attachements verwendet werden, dass allerdings nicht in allen Soap-Engines unterstützt wird, wie z.B. bei kSoap. JAXB ermöglicht die Einbindung dadurch, dass aus einem XML-Dokument und einem XSD-Schema, mit deren Hilfe das Java-Objekt beschrieben wird, Java-Klassen generiert werden, ähnlich wie beim Generieren der Web Service Stubs.

4.5 Kommunikation

Hier wird die Clientseitige Repräsentation des Web Services, auch Stubs genannt, vorgestellt. Sie dienen als Kommunikationsgrundlage für die Clientanwendung mit dem Dienst. Es werden 3 Möglichkeiten behandelt, die sich alle etwas anders verhalten. Bedingt durch die Inkompatibilität einiger Softwarekomponenten. Die fertigen Klassen wurden als JAR exportiert und können so bei der Entwicklung eines Clients genutzt werden um die Verbindung aufzubauen.

4.5.1 CXF

CXF hat eine WSDL Unterstützung. Dadurch können die Client-Stubs aus der WSDL generiert werden. Dies geschieht mit Hilfe eines Maven-Plugins und wird zur Kompilierungszeit durchgeführt.

```
1 <plugin>
2   <groupId>org.apache.cxf</groupId>
3   <artifactId>cxf-codegen-plugin</artifactId>
4     <version>2.1.2</version>
5     <executions>
6       <execution>
7       <id>generate-wsdl-sources</id>
8       <phase>generate-sources</phase>
```

[6] Java Architecture for XML Binding

```
9       <configuration>
10        <sourceRoot>{basedir}/src/main/java</sourceRoot>
11          <wsdlOptions>
12            <wsdlOption>
13              <wsdl>http://localhost:8080/service/WebService
                   ?wsdl</wsdl>
14            </wsdlOption>
15          </wsdlOptions>
16      </configuration>
17        <goals>
18          <goal>wsdl2java</goal>
19        </goals>
20      </execution>
21    </executions>
22  </plugin>
```

Nachdem die Stubs generiert wurden, können mit deren Hilfe die Web Service Methoden aufgerufen werden. Dadurch ist sehr wenig Java Code nötig um die Verbindung aufzubauen. Es wird ein Service Objekt angelegt in Abhängigkeit der URL, dem Namespace und einem Namen für den Dienst. Zusätzlich wird über den Namespace, die URL, einem Portnamen und das SoapBinding ein Port angelegt. Beim Binding wird festgelegt über welches Netzwerkprotokoll Soap übertragen wird. In diesem Fall ist es HTTP. Jetzt wird der Port geholt in Abhängigkeit des Service Interfaces, das auch als lokale Repräsentation vorhanden ist. Dieser Aufruf wird in ein vorher angelegtes Objekt vom Typ des Interfaces geschrieben.

```
1  static IRemoteService call;
2
3  public static IRemoteService Connect()  {
4       String url = "http://192.168.99.14:8080/service/
           WebService";
5         Service service;
6     try {
7       service = Service.create(
8             new URL( url + "?wsdl" ),
9             new QName( "http://ba.fhkoeln.de/service/", "
                 RemoteServiceService" ) );
10          service.addPort(
```

```
11                  new QName( "http://ba.fhkoeln.de/service
                       /", "RemoteServiceServicePort" ),
12                  SOAPBinding.SOAP11HTTP_BINDING, url );
13
14          call = service.getPort( IRemoteService.class );
15      }
16      catch (MalformedURLException e) {
17        e.printStackTrace();
18      }
19      return call;
20      }
```

Nun kann man über das Call Objekt alle Methoden des Dienstes aufrufen.

```
1 call.sendMsg("Hallo","foo","test", "false");
```

In Soap werden die Nutzdaten und Steuerinformationen in einer XML Datei übertragen. Bytenachrichten müssen erst einer Kodierung unterzogen werden um sie im Anschluss auch in der XML Datei übertragen zu können.

4.5.2 Rest

Rest unterstützt keine WSDL. Außerdem werden keine Daten über die Verbindung und die Teilnehmer gespeichert, somit muss jeder Aufruf komplett neu aufgesetzt werden. Hierbei werden alle Methoden des Web Services gesondert behandelt. Für jede Methode muss die Verbindung zum Dienst aufgebaut werden und dann kann die Methode ausgeführt werden. Bei einer Textnachricht wird diese in einen String eingelesen und bei Rest im Header des Aufrufs übertragen. Die Annotation *„@QueryParam“* besagt, dass der Paramter beim Rest-Aufruf im Header übertragen wird. Ohne diese Annotation werden die Informationen im Body übertragen. Dies ist bei Bytenachrichten der Fall. Als erstes muss die URL definiert werden. Diese stellt sich aus dem Methodennamen, den Parameternamen und den Werten zusammen. Im Anschluss wird eine *„HTTPConnection“* mit der eben zusammengestellten URL geöffnet. Nun können verschiedene Parameter für diese Verbindung festgelegt werden. In Rest werden die Daten mittels Stream ein- und ausgelesen. Wenn dies geschehen ist können die Informationen gesendet werden. Da Rest auf HTTP Methoden beruht, wird auch immer ein Antwortcode gesendet. Mit dessen Hilfe wird ermittelt, ob der Aufruf erfolgreich war oder nicht.

```
1 public String sendMsg(String msg, String id, String name,
     String flag) throws IOException{
```

```
2
3     URL myurl = new URL(url + "/set?" + "msg=" + msg + "&id
          =" + id + "&name=" + name + "&flag=" + flag);
4
5     HttpURLConnection con = (HttpURLConnection) myurl.
          openConnection();
6     con.setRequestMethod(methodPost);
7     con.setDoOutput(true);
8     con.setDoInput(true);
9     con.connect();
10    OutputStreamWriter out = new OutputStreamWriter(con.
          getOutputStream());
11    out.write(msg);
12    out.flush();
13
14    if (con.getResponseCode() == HttpURLConnection.
          HTTP_NO_CONTENT ) { /* 204 */
15      result = "Erfolgreich";
16    } else {
17      result = "Fehler: " + con.getResponseCode() + " " +
            con.getResponseMessage();
18    }
19    out.close();
20    con.disconnect();
21    return result;
22  }
```

Beim Empfang der Daten geht man wie folgt vor:

```
1 public String getMsg(String id, String name, String flag)
      throws IOException{
2
3     URL myurl = new URL(url + "/get?" + "id=" + id + "&name
          =" + name + "&flag=" + flag);
4
5     HttpURLConnection con = (HttpURLConnection) myurl.
          openConnection();
6     con.setRequestMethod(methodGet);
7     con.setDoInput(true);
```

```
8     con.connect();
9     if (con.getResponseCode() == HttpURLConnection.HTTP_OK )
          { /* 200 */
10      InputStream in = con.getInputStream();
11      StringBuffer buf = new StringBuffer();
12      int c;
13      while ( (c = in.read()) != -1 ){
14        buf.append( (char) c);
15      }
16      in.close();
17      result = buf.toString();
18    } else {
19      result = "Fehler: " + con.getResponseCode() + " " +
            con.getResponseMessage();
20    }
21    con.disconnect();
22    return result;
23  }
```

Der Aufbau ist der Selbe wie beim Senden, es wird lediglich ein anderer Reader genutzt, um die Daten in die Ausgabe zu schreiben. Bei Bytenachrichten verhält es sich analog.

```
1 public String sendByteMsg(byte[] msg, String id, String name
     , String flag) throws IOException{
2
3     URL myurl = new URL(url + "/setb?" + "&id=" + id + "&
        name=" + name + "&flag=" + flag);
4
5     HttpURLConnection con = (HttpURLConnection) myurl.
        openConnection();
6     con.setRequestMethod(methodPost);
7     con.setDoOutput(true);
8     con.setDoInput(true);
9     con.connect();
10    OutputStream out = con.getOutputStream();
11    out.write(msg);
12
13    if (con.getResponseCode() == HttpURLConnection.
        HTTP_NO_CONTENT ) { /* 204 */
```

```
14        result = "Erfolgreich";
15      } else {
16        result = "Fehler: " + con.getResponseCode() + " " +
             con.getResponseMessage();
17      }
18      out.close();
19      con.disconnect();
20      return result;
21    }
```

Die Daten werden wie man in der URL sehen kann, nicht im Header, sondern im Body der Nachricht übertragen. Dafür ist der fehlende *„@QueryParam“* die Voraussetzung. Dann werden die Daten nicht in der URL übergeben, sondern über den Stream geschrieben.

4.5.3 kSoap

KSoap hat auch keine WSDL Unterstützung. Hier muss, ähnlich wie bei Rest, jede Methode mit erneutem Verbindungsaufbau implementiert werden. Der Aufbau sieht wie folgt aus:

```
1  private final String SOAP_ACTION = "sendMsg";
2      private final String METHOD_NAME = "sendMsg";
3      private final String SOAP_ACTION2 = "getMsg";
4      private final String METHOD_NAME2 = "getMsg";
5      private final String SOAP_ACTION3 = "subscribeTopic";
6    private final String METHOD_NAME3 = "subscribeTopic";
7    private final String SOAP_ACTION4 = "unsubscribeTopic";
8    private final String METHOD_NAME4 = "unsubscribeTopic";
9    private final String SOAP_ACTION5 = "sendByteMsg";
10   private final String METHOD_NAME5 = "sendByteMsg";
11   private final String SOAP_ACTION6 = "getByteMsg";
12   private final String METHOD_NAME6 = "getByteMsg";
13     private final String NAMESPACE = "http://ba.fhkoeln.de/
         service/";
14     private final String URL = "http://192.168.99.14:8080/
         service/WebService";
```

Es werden Konstanten für die Soapaktion, Methodennamen, Namespace und die URL erstellt. Diese dienen als Parameter für die Methoden und vereinfachen die Arbeit.

```
1  public void connect(String a, String b, String c, String d){
2
3          SoapObject request = new SoapObject(NAMESPACE,
               METHOD_NAME);
4            request.addProperty("msg",a);
5            request.addProperty("id",b);
6            request.addProperty("name",c);
7            request.addProperty("flag",d);
8
9          SoapSerializationEnvelope envelope = new
               SoapSerializationEnvelope(SoapEnvelope.VER11);
10         envelope.setOutputSoapObject(request);
11
12         AndroidHttpTransport androidHttpTransport = new
               AndroidHttpTransport (URL);
13         {
14           try {
15             androidHttpTransport.call(SOAP_ACTION, envelope)
                   ;
16           }
17           catch(Exception E) {
18             E.printStackTrace();
19           }
20         }
21  }
```

Der Aufbau ist immer der Gleiche. Es wird ein SoapObjekt angelegt mit dem Namespace und dem Methodennamen. Hiermit wird festgelegt, wo der Web Dienst zu finden ist und welche Methode genutzt wird. Anschließend werden die zu übergebenden Daten als Eigenschaft des Objekts definiert. Ein Envelope wird erstellt, das als Container dient für den Header und den Body der Soapnachricht. Mit dessen Hilfe wird das zuvor erstellte Objekt als Output gesetzt. Hierbei werden die zu versendenden Informationen in das Envelope geschrieben. Nun fehlt noch der letzte Schritt. Dabei wird das Transport-Objekt erstellt. In diesem Fall*„HTTPTransport“*. Hier wird die URL übergeben, über die eine Verbindung geöffnet wird. Jetzt da eine Verbindung offen ist, kann die Nachricht versendet werden. Dies geschieht mit der *„call()“* Methode, der das Envelope und die Aktion die ausgeführt werden soll, übergeben wird.
Beim Empfang geht man wie folgt vor:

```
1  public String getText(String a, String b, String c){
2          //CALL the web service method
3          SoapObject request = new SoapObject(NAMESPACE,
               METHOD_NAME2);
4          request.addProperty("id",a);
5          request.addProperty("name",b);
6          request.addProperty("flag",c);
7
8
9          SoapSerializationEnvelope envelope = new
               SoapSerializationEnvelope(SoapEnvelope.VER11);
10         envelope.setOutputSoapObject(request);
11
12         AndroidHttpTransport androidHttpTransport = new
               AndroidHttpTransport (URL);
13         {
14
15           try {
16             androidHttpTransport.call(SOAP_ACTION2, envelope
                   );
17             SoapPrimitive result = (SoapPrimitive) envelope.
                   getResponse();
18             temp = result.toString();
19           }
20           catch(Exception E) {
21             E.printStackTrace();
22           }
23         }
24       return temp;
25     }
```

Der Aufbau ist der selbe wie beim Senden. Es kommen lediglich zwei Zeilen Code hinzu die das Ergebnis verarbeiten. Es wird ein SoapPrimitiv Objekt angelegt, das mit den Daten, die aus dem Envelope ausgelesen werden, gefüllt wird. Dies wird nun in String umgewandelt und ausgegeben. Bei Bytenachrichten sieht das Ganze ähnlich aus, jedoch muss wie schon bei CXF erwähnt, der Byteanteil der Nachricht kodiert werden. Dies geschieht hier mittels *„Base64“* Kodierung.

```
1 SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME5)
     ;
2     request.addProperty("msg", new SoapPrimitive(
         SoapEnvelope.ENC, "base64", Base64.encode(a)));
3     request.addProperty("id",b);
4         request.addProperty("name",c);
5         request.addProperty("flag",d);
```

Dies gilt auch für den Empfang. Dabei muss das kodierte Objekt wieder dekodiert werden.

```
1 byte[] temp = null;
2       try {
3         androidHttpTransport.call(SOAP_ACTION6, envelope);
4         SoapPrimitive result = (SoapPrimitive) envelope.
             getResponse();
5         temp = Base64.decode(result.toString());
6       } catch (Exception E) {
7         E.printStackTrace();
8       }
9       return temp;
```

4.5.4 Einschränkungen

Einschränkungen gab es bei Android. Dies unterstützt nicht alle Java Bibliotheken, da es eine andere VM verwendet. CXF gehört zu einer solchen. Deswegen konnte CXF leider nicht dazu verwendet werden Web Service Aufrufe zu tätigen. Obwohl dieses Framework den größten Funktionsumfang liefert.
Vorteil: über das „wsdl2java“ werden die Stubs für den Client generiert und somit automatisch die passende Schnittstelle geschaffen (hohe Flexibilität).

So musste auf kSoap oder alternativ Rest zurückgegriffen werden. KSoap hat keine WSDL Unterstützung und dadurch fällt der Code auch länger aus, jedoch wird es von Android unterstützt.
Nachteil: kann Binärdaten nur in base64-encodete Strings verarbeiten, da keine Attachment-Unterstützung vorhanden -> Probleme mit größeren Dateien und base64-encodete Daten werden um 30 Prozent größer.
Um kSoap nutzen zu können muss dieses externe Packet erstmal eingebunden werden.

Rest wird seitens des Betriebssystems unterstützt, da es auf HTTP beruht. Hier ist kein Einbinden anderer Bibliotheken von Nöten.
Vorteil: könnte auch größere Dateien versenden, da die HTTP Connection einen „ChunkedStreamingMode“ unterstützt und somit über die Chunk-Länge die Kommunikation entsprechend gesteuert werden kann.
Nachteil: da es keine Generierung von Client-Stubs gibt, muss die Client-Schnittstelle zum Webservice programmiert und manuell gewartet werden (unflexibel).

4.6 Mobiler Client

Das User Interface (UI) wurde in Android geschrieben und ist somit auf Android basierten Mobiltelefonen und Smartphones lauffähig. Dafür wurden von Hand ein paar Skizzen erstellt, um das Aussehen und Anforderungen zu bestimmen. Nachdem nun festgelegt wurde welche Ein/Ausgabe Felder und Schaltflächen benötigt werden und wie die Web Service Methoden implementiert werden sollen, wurde mit der Entwicklung begonnen. In Android wird das Layout in einer XML Datei hinterlegt. Hier werden die Objekte generiert, mit einer ID versehen und so positioniert, wie sie am späteren Bildschirm angeordnet sein sollen.

```
1  <TextView android:id="@+id/label_ID" android:text="ID:" />
2
3    <EditText android:id="@+id/text_ID" android:background="
        @android:drawable/editbox_background"
4      android:editable="true" android:singleLine="true" />
5
6    <EditText android:id="@+id/output" android:background="
        @android:drawable/editbox_background"
7      android:editable="false" android:lines="5" android:
          layout_marginTop="20px" android:inputType="
          textMultiLine" android:
          scrollbarAlwaysDrawVerticalTrack="true" android:
          scrollbars="vertical"/>
8
9    <TableRow>
10     <CheckBox android:id="@+id/async" android:text="CallBack
          " android:layout_marginLeft="18px"/>
11     <CheckBox android:id="@+id/topic" android:text="Topic"
          android:layout_marginLeft="10px"/>
```

```
12    </TableRow>
13
14    <TableRow>
15      <Button android:id="@+id/send" android:layout_below="@id
          /output"
16        android:layout_alignParentRight="false" android:
            layout_marginLeft="18px"
17        android:text="Send" />
18    </TableRow>
```

Später auf dem Gerät wird die XML-Struktur interpretiert und die entsprechenden Felder werden gezeichnet. In der folgenden Abbildung wird das verdeutlicht:

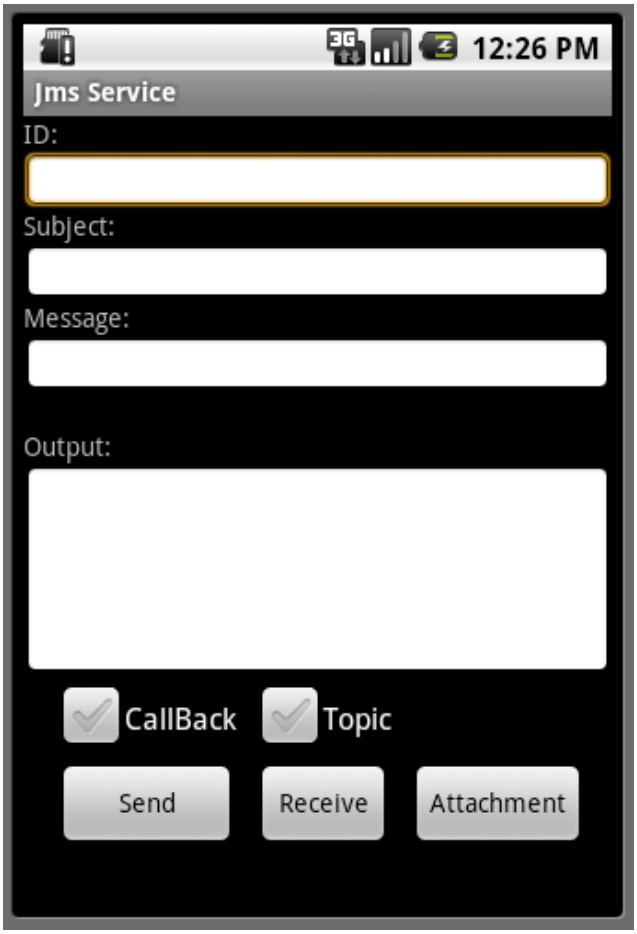

Abbildung 4.1: Android UI

Nun können diese Objekte im Java Code angesprochen werden und es können ihnen Aktionen zugewiesen werden. Zuerst werden Objekte erstellt, abhängig von ihrer Klasse z.B. *„Button“* und es wird ihnen die im XML dafür hinterlegte ID zugewiesen.

```
1   public Button sendButton;
2   public Button receiveButton;
3   public Button attachment;
4   public EditText idText;
5   public EditText betrText;
6   public EditText nachrText;
7   public EditText outText;
8   public CheckBox topic;
9   public CheckBox call;
10  public AlertDialog alertDialog;
11
12  this.sendButton = (Button)this.findViewById(R.id.send);
13  this.receiveButton = (Button)this.findViewById(R.id.
        receive);
14  this.attachment = (Button)this.findViewById(R.id.bt_Anhang
        );
15  this.idText = (EditText)this.findViewById(R.id.text_ID);
16  this.betrText = (EditText)this.findViewById(R.id.
        text_Betreff);
17  this.nachrText = (EditText)this.findViewById(R.id.
        text_Nachricht);
18  this.outText = (EditText)this.findViewById(R.id.output);
19  this.topic = (CheckBox) this.findViewById(R.id.topic);
20  this.call = (CheckBox) this.findViewById(R.id.async);
```

Jede Androidanwendung muss die folgende Methode implementieren. Diese wird beim Start des Apps ausgeführt. Dabei wird z.B. das vorher erstellte Layout in der *„main.xml“* geladen.

```
1 public void onCreate(Bundle savedInstanceState) {
2       Log.i("Test", "anfang");
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.main);
```

Nun kann jedes Element angesprochen werden oder es können Listener gesetzt werden, die eine Interaktion des Benutzers mit dem jeweiligen Objekt registrieren und darauf

reagieren. Wie z.B. im Fall des *„sendButtons“*. Hier wird nun der Web Service Aufruf getätigt und alle dafür nötigen Informationen werden aus den Eingabefeldern gelesen.

```
1 this.sendButton.setOnClickListener(new OnClickListener() {
2         public void onClick(View v) {
3                 connection.connect(nachrText.getText().
                     toString(),idText.getText().toString(),
                     betrText.getText().toString(),Boolean.
                     toString(topic.isChecked()));
4         }
5 });
```

4.6.1 Probleme

Probleme bei der Entwicklung gab es kaum. Lediglich mit der Syntax und der Arbeitsweise von Android, da an dieser Stelle etwas Einarbeitungsarbeit geleistet werden musste. Das Augenmerk lag nicht auf der Erstellung des UI, infolgedessen ist der Umfang des Prototypen angepasst.

5 Ähnliche Arbeiten

5.1 PubSubHubbub

Ein Projekt das sich mit JMS und dem publish/subscribe Prinzip befasst ist PubSubHubbub. Dabei handelt es sich um ein Server zu Server Protokoll, als Erweiterung zu Atom und RSS. Damit kann ein Server, der an einem bestimmten Topic interessiert ist, mittels einer Callback-Funktion über einen Nachrichtenaustausch, informiert werden ob sich was geändert hat. Hierfür muss beim Anlegen eines Topics ein Hub definiert werden in der Atom oder RSS XML Datei. Dieser läuft beim Publisher des Feeds. Ein Subscriber, der Server der Interesse an dem Topic hat, benutzt die Atom URL, die einen Hub definiert um sich zu registrieren und wird so über Updates am Laufenden gehalten. Wenn der Publisher nun Neuigkeiten veröffentlicht, wird über den Hub signalisiert, dass Updates vorhanden sind. Hierbei verteilt der Hub die Neuigkeiten an alle registrierten Subscriber dieses Topics.[Goo10b]

5.2 A Pro-active Mobility Extension for Pub/Sub Systems

Dieses Paper zeigt eine neuartige und effiziente mobile Erweiterung zu bestehenden pub/sub Systemen, die einige Vorteile bietet. So wäre eine Pufferung während einem Verbindungsverlust, bedingt durch Netzinstabilitäten, leeren Akkus oder Ähnlichem, bei Datenübertragungen wünschenswert. Deswegen besteht eine erhöhte Nachfrage nach Middleware zur Erfüllung der dynamischen Art des Mobile-Computing. Pub/sub Systeme sind wegen ihrer Funktionsweise sehr gut dafür geeignet. Da eine Entkopplung der Verleger und Abonnenten in der Zeit besteht. Zusätzlich bieten sie eine Vielzahl anonymer Funktionen. Der Verleger publiziert Daten und Abonnenten bekunden ihr Interesse mittels Abonnements. Ein Ereignis-Broker leitet diese Neuerungen an die Interessenten weiter.
Bestehende Systeme sind für statische Clients entwickelt und sind sehr unflexibel. Es würde einige Addons und Plugins benötigen, um mobile Clients zu unterstützen. Demzufolge wird hier ein neuer Ansatz verfolgt. Basierend auf einem Nachbar-Graphen, wird mittels Proxy ein Dummy des Teilnehmers erstellt, um an seiner Stelle zu handeln. Hinzu kommt, dass das Abonnement nur dann aktiv ist, wenn die Verbindung getrennt

ist. Wenn dies nicht der Fall sein sollte werden die Aktionen nicht beachtet und vom System entsorgt.
Damit wird der Nachrichtenverlust um 50 Prozent reduziert und die Doppelverarbeitung von Nachrichten strebt gegen null. Weiterhin entstehen geringere Kosten, weil ein kleinerer Overhead beim Nachrichtentransfer ensteht. Der Durchsatz ist jedoch erhöht. Die Ergebnisse der Untersuchung werden in Tabellen visualisiert und durch Hilfe mobiler Teilnehmer evaluiert.[AG08]

5.3 Mobile devices as Web service providers

In dieser Masterarbeit beschreibt der Autor die erfolgreiche Nutzung und Veröffentlichung von Web Services von mobilen Geräten. Die Verwaltung, Kontrolle und der Zugang selbst werden thematisiert. Weiterhin wird ein Framework entwickelt, das dies ermöglicht.
Web Services stellen eine Entkopplung und einen verteilten Zugriff zwischen Informationssystemen dar. Sie bieten eine Schnittstelle als Zugriff. Die Nutzung von Web Services verbessert die Wiederverwendbarkeit von Diensten und es müssen weniger Applikationen entwickelt werden. Die Kommunikation wird optimiert und eine effizientere Nutzung der Kommunikationskanäle ist die Folge, da eine Möglichkeit besteht nur benötigte Informationen abzurufen statt einer gesamten Web Seite beispielsweise. Weiterhin wird durch die Übertragung serialisierter XML-Daten die Größe verringert. Diese Vorteile qualifizieren Web Services für diesen Einsatz. Der Zugang zu mobilen Geräten ist durch verschiedene Api's wie kSoap, gSoap, Rest etc. gegeben.[Sku08]

5.4 Amazon Simple Queue Service (SQS)

Amazon SQS ist ein Web Service, der auf der weboptimierten Mitteilungs-Infrastruktur von Amazon beruht. Jeder Computer mit Verbindung ins Internet kann ihn verwenden und Mitteilungen empfangen oder senden. Anwendungen die SQS verwenden, können unabhängig ausgeführt werden und müssen nicht zur selben Zeit und im gleichen Netzwerk ausgeführt werden. SQS bietet eine Warteschlange, zum Speichern von Nachrichten, während diese zwischen Computern weitergeleitet werden. Entwickler können Daten zwischen verteilten Komponenten ihrer Anwendung, die verschiedene Aufgaben ausführen, verschieben. Dabei können Mitteilungen bis zu einer Größe von 8 KB, in einem beliebigen Textformat, verschickt werden und es muss nicht immer jede Komponente verfügbar sein. Während der Verarbeitung sind die Mitteilungen gesperrt. Somit wird eine gleichzeitige Verarbeitung von mehreren Computern verhindert. Schlägt die Verarbeitung fehl oder braucht eine Anwendung mehr Zeit zur Verarbeitung,

wird die Sperre wieder aufgehoben bzw. kann sie durch den Entwickler, dynamisch, mittels einer Operation, geändert werden. Weitere Operationen sind beispielsweise: *„CreateQueue, DeleteQueue, SendMessage, ReceiveMessage“* etc..
SQS ist entwickelt worden, um mit anderen Amazon-Softwarelösungen zusammen zu arbeiten, wie beispielsweise EC2, S3 und SimpleDB. Weiterhin gestaltet sich die Entwicklung recht einfach, da nur die API benötigt wird und diese praktisch mit jeder Sprache und Plattform verwendet werden kann. SQS ist dafür ausgelegt, dass eine unbegrenzte Anzahl an Computern eine unbegrenzte Anzahl an Mitteilungen lesen und schreiben kann. Der Zugriff auf den Web Service ist über SOAP möglich. Der Dienst kann nach einer Anmeldung bei Amazon gebührenpflichtig genutzt werden. Hier wird nach Übertragungsvolumen abgerechnet.
Dazu ein Beispiel:
„Eine Videotranscodierungs-Website verwendet Amazon EC2, Amazon SQS, Amazon S3 und Amazon SimpleDB gemeinsam. Endanwender übertragen Videos zur Transcodierung an die Website. Die Videos werden in Amazon S3 gespeichert und eine Mitteilung (die Anforderungsmitteilung) mit einem Pointer auf das Video und das Zielformat des Videos in einer Amazon SQS-Warteschlange (der Eingangswarteschlange) abgelegt. Die auf einem Verbund aus Amazon EC2-Instanzen ausgeführte Transcodierungsmaschine liest die Anforderungsmitteilung aus der Eingangswarteschlange, ruft die Videos mit Hilfe des Pointers von Amazon S3 ab und transcodiert das Video in das Zielformat. Das konvertierte Video wird zurück in Amazon S3 abgelegt und eine andere Mitteilung (die Antwortmitteilung) wird in einer anderen Amazon SQS-Warteschlange (der Ausgangswarteschlange) mit einem Pointer auf das konvertierte Video platziert. Gleichzeitig können Metadaten zum Video (z. B. Format, Erstellungsdatum und Länge) zur leichten Abfragedurchführung in Amazon SimpleDB indiziert werden. Während dieses ganzen Ablaufs kann eine dedizierte Amazon EC2-Instanz durchgängig die Eingangswarteschlange überwachen und, basierend auf der Anzahl der Mitteilungen in der Eingangswarteschlange, die Anzahl der transcodierenden Amazon EC2-Instanzen dynamisch anpassen, um den Kundenanforderungen bezüglich der Reaktionszeit gerecht zu werden.“[Ama10]

5.5 Android Cloud to Device Messaging Framework (C2DM)

Android Cloud to Device Messaging ist eine Dienstleistung, die Entwicklern erlaubt, Daten von Servern, zu ihren Anwendungen auf Android-Geräten, zu senden. Der Service bietet einen einfachen und leichten Mechanismus, den Server verwenden können, um mobilen Anwendungen zu sagen, dass sie den Server direkt kontaktieren sollen, um aktualisierte Anwendungs- oder Nutzerdaten zu holen. Der C2DM Service kümmert sich um alle Aspekte. Wie z.B. die Verwaltung von Nachrichtenwarteschlangen und die Auslieferung an die Ziel-Anwendung. Es ermöglicht Drittanbietern von Anwendungsservern, Nachrichten an ihre Android-Anwendungen zu senden. Dieser Nachrichtendienst ist nicht für die Versendung von Benutzerdaten konzipiert. Vielmehr soll er verwendet werden, um Anwendungen zu signalisieren, dass neue Daten auf dem Server verfügbar sind, die abgeholt werden können. C2DM übernimmt keine Garantien für die Auslieferung oder die Reihenfolge der Einträge. So wird es zum Beispiel nicht verwendet, um die eigentlichen Nachrichten weiterzugeben, sondern vielmehr dafür genutzt, um einer Instant-Messaging-Anwendung, respektive dem Benutzer mitzuteilen, dass neue Nachrichten eingegangen sind. Die Anwendung auf einem Android-Gerät muss nicht ausgeführt sein, um Nachrichten zu empfangen. Das System wird die Anwendung aufwecken, wenn die Meldung eingeht. C2DM bietet keine integrierte Benutzeroberfläche oder sonstiges, bezüglich der Handhabung der Meldung. Diese Roh-Nachricht, über neu eingetroffene Daten, wird direkt an die Applikation weitergeleitet, die die volle Kontrolle darüber hat, wie damit umgegangen wird. Zum Beispiel könnte die Anwendung nach einer Anmeldung, eine Benutzeroberfläche zeigen oder stillschweigend die Daten synchronisieren. C2DM erfordert Geräte mit Android 2.2 oder höher, die auch die Market-Anwendung installiert haben. Weiterhin nutzt C2DM bestehende Google-Dienste, so dass die Einrichtung eines Google-Kontos, auf den mobilen Geräten, von Nöten ist.

Bei der Nutzung dieses Dienstes, gibt es drei primäre Prozesse die ablaufen: Als erstes die Registrierung. Hierbei registriert sich eine Anwendung, die beabsichtigt C2DM zu nutzen, mittels ID, bei dem C2DM Server, um Mitteilungen empfangen zu können. Wenn die Registrierung erfolgreich war, wird der Applikation eine ID zugewiesen, welche sie dem Application-Server mitteilt. Dieser verwaltet alle registrierten IDs in einer Datenbank, bis die Anwendung die Registrierung rückgängig macht.

Der zweite Schritt ist das Senden der Nachricht. Die unter Punkt eins erwähnten Schritte gelten als Vorbedingung um Nachrichten versenden zu können. Weiterhin muss der Application-Server ein Token implementieren, das ihm die Genehmigung erteilt sich am Client einzuloggen. Das muss vom Entwickler, für die Anwendung, auf dem Server

eingerichtet werden. Dies wird dann zum Versenden der Nachrichten, an bestimmte Anwendungen, genutzt. Der Application-Server hat ein Token für ein bestimmtes 3rd-Party-App und mehrere registrierte IDs. Jede registrierte ID repräsentiert ein bestimmtes Gerät.
Die Abläufe beim Senden sehen nun wie folgt aus: Der Applikations-Server sendet eine Nachricht an den C2DM Server. Google speichert die Nachricht, für den Fall, dass das Gerät inaktiv ist. Wenn das Gerät nun online ist, sendet Google die Meldung weiter. Auf dem Gerät wird die Nachricht vom System an die entsprechende Anwendung weitergeleitet, so dass nur die gewünschte Anwendung die Nachricht erhält.
Im dritten Teil geht es um die Verarbeitung der eingegangenen Nachricht, seitens der Anwendung. Nach dem Empfang werden die Schlüssel aus der Nachricht extrahiert, um mit deren Hilfe die Rohdaten zu erhalten und diese dann im Anschluss zu verarbeiten.[Goo10a]

6 Fazit

Diese Arbeit beschäftigte sich zum größten Teil mit der Entwicklung der Middleware-Komponente. Den gesamten Umfang des Projekts so ausführlich zu betrachten, hätte den Zeitrahmen gesprengt. Deswegen war das Ziel, einen lauffähigen Prototypen zu implementieren, der die grundlegenden Funktionen bietet. Der auch gleichzeitig zu Demonstrationszwecken genutzt wird, um zu zeigen, dass der eingeschlagene Weg von Erfolg gekrönt ist. Dafür musste allerdings erst recherchiert werden um ein in diesem Kontext funktionierendes Konzept zu erstellen. Ausgehend von der Fragestellung, wie ein Zugriff auf nachrichtenorientierte Kommunikationsdienste von mobilen Geräten aussehen könnte, wurde dies in Angriff genommen. Dabei traten an gewissen Stellen sicherlich einige Probleme auf, jedoch konnten diese zum Großteil beseitigt werden. Deswegen sehe ich den Erfüllungsgrad der Arbeit in Bezug auf die Fragestellung als angemessen an.
Das Konzept kann funktionieren, es ist offen und flexibel, die Wartung ist nicht aufwendig und die Möglichkeit gewisse Tagesabläufe im Internet zu vereinen, unter dem Aspekt einer leicht benutzbaren Software, klingt verlockend. Des Weiteren bietet es eine einfache und kostengünstige Art sich mit Freunden zu unterhalten, ohne erst ein Benutzerkonto anlegen zu müssen. Diesbezüglich gibt es auch keine Möglichkeit von anderen gefunden zu werden, außer es wird einem persönlich mitgeteilt. Das macht es anonymer und beschränkt das Ganze auf den engsten Freundeskreis, was sich im Bereich der gängigen Messanger-Systeme oder Community-Seiten als schwierig gestaltet. Hierbei nimmt man in Kauf, unerwünschte Nachrichten zu erhalten, nur um eine gemeinsame Plattform mit Freunden zu haben.

7 Ausblick

Da auf einige Dinge aus Zeitgründen verzichtet wurde und einige andere nicht Mittelpunkt der Arbeit waren, ist eine beidseitige Erweiterung durchaus denkbar. Der Web Service könnte, um einige Funktionalitäten des JMS-Systems und um dem Konzept zu genügen, erweitert werden. Hier ist die Implementierung aller relevanter Nachrichtentypen zu nennen oder die Erweiterung des Topics. Bei Letzterem müsste eine Möglichkeit geschaffen werden, Updates im abonnieren Feed oder Blog zu registrieren und zu verarbeiten.
Eine erweiterte Fehlerbehandlung, sowohl im Dienst als auch im Clientbereich, sollte nicht außer Acht gelassen werden.
Auf der anderen Seite kann die Clientapplication um einige Dinge erweitert werden um dem Konzept zu genügen. Da wäre einmal die Verwaltung der Freunde und abonnierter Feeds zu nennen, sowie die Möglichkeiten der Liste neue Einträge hinzu zu fügen oder löschen zu können. Weiterhin könnte die Implementierung der restlichen Nachrichtentypen vorgenommen werden, wie z.B. bei Bytenachrichten die Möglichkeit ein Foto oder Ähnliches vom Dateisystem auswählen und versenden zu können. Außerdem muss eine Identifikation der hier versendeten Dateitypen ergänzt werden, um beispielsweise aus einem JPG-Bild auch wieder ein Solches beim Empfang zu erstellen. Hier könnte auf schon beschriebene Methoden zurückgegriffen werden oder Klassen des JDK 6 nutzen, die eine Bestimmung des MIME-Types aus dem Dateinamen ermöglichen. Diese beidseitigen Erweiterungen des Prototyps würden dazu führen, ihn im gesamten Umfang wie beschrieben nutzen zu können.

Abbildungsverzeichnis

Glossar

API ..... ist eine Programmierschnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.

ATOM ..... Atom ist ein XML-Format, dass den plattformunabhängigen Austausch von Informationen ermöglicht.

DOM ..... Das Document Object Model ist eine Spezifikation einer Schnittstelle für den Zugriff XML-Dokumente.

HTML ..... ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten. HTML-Dokumente sind die Grundlage des World Wide Web und werden von einem Webbrowser dargestellt.

HTTP ..... Hypertext Transfer Protocol: ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten aus dem World Wide Web (WWW) in einen Webbrowser zu laden.

JAXB ..... ist eine Programmschnittstelle in Java, die es ermöglicht, Daten aus einer XML-Schema-Instanz heraus automatisch an Java-Klassen zu binden und diese Java-Klassen aus einem XML-Schema heraus zu generieren. Somit ist ein Arbeiten mit XML-Dokumenten möglich, ohne dass der Programmierer direkt Schnittstellen zur Verarbeitung von XML wie SAX oder DOM verwenden muss.

JNDI ..... ist eine Programmierschnittstelle innerhalb der Programmiersprache Java für Namensdienste und Verzeichnisdienste. Mit Hilfe dieser Schnittstelle können Daten und Objektreferenzen anhand eines Namens abgelegt und von Nutzern der Schnittstelle abgerufen werden.

MIME-Type ..... er klassifiziert die Daten im Rumpf einer Nachricht im Internet. Es wird z. B. bei einer HTTP-Übertragung einem Browser mitgeteilt, welche Daten der Webserver sendet – ob es beispielsweise Klartext, ein HTML-Dokument oder ein PNG-Bild ist.

POJO ..... Plain Old Java Objekt: ein normales Objekt in der Programmiersprache Java

POM ..... Maven's Konfigurations-Datei

RSS ..... ist eine Familie von Formaten für die einfache und strukturierte Veröffentlichung von Änderungen auf Websites (z. B. News-Seiten, Blogs, Audio-/Video-Logs etc.) in einem standardisierten Format (XML).

Servlet ..... Als Servlets bezeichnet man Java-Klassen, deren Instanzen innerhalb eines Java-Webservers Anfragen von Clients entgegen nehmen und beantworten.

TCP/IP ..... ist eine Familie von Netzwerkprotokollen und wird wegen ihrer großen Bedeutung für das Internet auch als Internetprotokollfamilie bezeichnet.

UDDI ..... Ist ein Begriff aus dem Umfeld der Serviceorientierten Architektur und bezeichnet einen standardisierten Verzeichnisdienst, der die zentrale Rolle in einem Umfeld von dynamischen Web Services spielen sollte.

Virtuelle Maschine . Umgebung in der Java Code ausgeführt wird

XML ..... ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten.

Literaturverzeichnis

[AG08] Abdulbaset Gaddah, Thomas Kunz: *A Pro-active Mobility Extension for Pub/Sub Systems*. Diskussionspapier, Carleton University 1125 Colonel By Drive Ottawa, Ont., Canada K1S 5B6, 2008.

[Ama10] Amazon: *Amazon Simple Queue Service*, 2010. `http://aws.amazon.com/de/sqs/`. Sichtung: 27.06.2010.

[Apa10a] Apache: *ActiveMQ*, 2010. `http://activemq.apache.org/spring-support.html`. Sichtung: 27.06.2010.

[Apa10b] Apache: *Apache Camel*, 2010. `http://camel.apache.org/better-jms-transport-for-cxf-webservice-using-apache-camel.html`. Sichtung: 27.06.2010.

[Apa10c] Apache: *Apache cxf*, 2010. `http://cxf.apache.org/docs/`. Sichtung: 27.06.2010.

[Apa10d] Apache: *Apache Maven Dokumentation*, 2010. `http://maven.apache.org/guides/`. Sichtung: 27.06.2010.

[Dev10] Developers, Android: *The Developer's Guide*, 2010. `http://developer.android.com/guide/index.html`. Sichtung: 27.06.2010.

[Goo10a] Google: *Android Cloud to Device Messaging Framework*, 2010. `http://code.google.com/intl/de-DE/android/c2dm/`. Sichtung: 27.06.2010.

[Goo10b] Google: *PubSubHubbub*, 2010. `http://code.google.com/p/pubsubhubbub/`. Sichtung: 27.06.2010.

[Met08] Metzler, Ingo et al.: *Service-orientierte Architekturen mit Web Services*. Spektrum Verlag, Heidelberg, 3. Auflage, 2008. ISBN 978-3-8274-1993-4.

[Ora10] Oracle: *Java JDK 6 Documentation*, 2010. `http://download.oracle.com/javase/6/docs/`. Sichtung: 27.06.2010.

[Pap08] Papazoglou, Michael P.: *Web Services: Principles and Technology*. Pearson Education Limited, Essex, England, 1. Auflage, 2008. ISBN 978-0-321-15555-9.

[Sku08] Skulason, Magnus Agust: *Mobile devices as Web service providers*. Diplomarbeit, Technical University of Denmark, 2008. `http://webcache.googleusercontent.com/search?q=cache:H4j9DhT_9F0J:orbit.dtu.dk/getResource%3FrecordId%3D223471%26objectId%3D1%`

26versionId%3D1+Mobile+devices+as+Web+service+providers&cd=1&hl=de&ct=clnk&gl=de. Sichtung: 27.06.2010.

[Sou10a] Source, Spring: *Spring Framework, JMS*, 2010. http://static.springsource.org/spring/docs/2.5.x/reference/jms.html. Sichtung: 27.06.2010.

[Sou10b] Sourceforge: *ksoap2*, 2010. http://ksoap2.sourceforge.net/. Sichtung: 27.06.2010.

[Ter02] Terry, Shaun: *Enterprise JMS programming*. MT Books, 2002. ISBN 0-7645-4897-2.

[Wal08] Walls, Craig / Breidenbach, Ryan: *Spring im Einsatz*. Hanser, 1. Auflage, 2008. ISBN 978-3-446-41240-8.

[Wik10] Wikipedia: *SOAP, CXF, REST, Web Service, WSDL, Java Message Serivce, Apache ActiveMQ, WebServer, Jetty, Maven, Spring, Android*, 2010. http://de.wikipedia.org/wiki. Sichtung: 27.06.2010.

[Wol10] Wolff, Eberhard: *Spring 3*. dpunkt.verlag, 3. Auflage, 2010. ISBN 978-3-89864-572-0.

Anhang

Quellen CD

Der Quellcode der Arbeit liegt auf CD bei.

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 23. August 2010

Jochen Todea