FH JOANNEUM Gesellschaft mbH

Mobile Geräte mit dem Android-Betriebssystem als Webservice-Clients

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science in Engineering (BSc)
eingereicht am
Fachhochschul-Studiengang Software Design

Betreuer: DI Johannes Feiner

eingereicht von: Bernhard Eibegger
Personenkennzahl: 1010418050

September 2013

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Bernhard Eibegger, Judenburg, 16. September 2013

Kurzfassung

Mobile Geräte, wie Smartphones und Tablet-PCs, haben im Vergleich zu Desktop-PCs und Laptops eine weniger leistungsfähige CPU und einen kleineren Arbeitsspeicher. Des Weiteren setzt auch die Kapazität des Akkus eine Grenze, inwieweit diese Geräte einsetzbar sind. Außerdem bietet das Mobilfunknetz nicht die gleich hohen Datenübertragungsraten wie LAN oder WLAN. Dies muss beim Verwenden von Webservices entsprechend berücksichtigt werden. Auch der Implementierungsaufwand stellt ein wichtiges Kriterium für den Einsatz einer Technologie dar.

Für diese Arbeit werden Android-Anwendungen, die als Client-Anwendungen für die Nutzung von SOAP-basierten und RESTful Webservices verwendet werden können, entwickelt. Damit wird einerseits die grundsätzliche Verwendbarkeit der mobilen Geräte gezeigt und der Implementierungsaufwand bei den unterschiedlichen Technologien abschätzbar. Andererseits implementieren die Client-Anwendungen alle erforderlichen Methoden, um die Nutzung der CPU, des Arbeitsspeichers, der übertragenen Datenmengen, der Übertragungs- und Verarbeitungsdauer sowie des Stromverbrauchs zu ermitteln.

In dieser Arbeit wird gezeigt, dass es grundsätzlich möglich ist, sowohl SOAP-basierte als auch RESTful Webservices von mobilen Geräten mit dem Android-Betriebssystem aus zu nutzen. Dabei unterscheidet sich der Implementierungsaufwand nur unwesentlich zwischen den verschiedenen, verwendeten Technologien. Auch der Unterschied bzgl. CPU-Nutzung, Bedarf an Arbeitsspeicher und die Dauer der Verarbeitung ist nur unwesentlich im Gegensatz zu den Vorteilen der richtigen Art und Weise der Datenübertragung. D. h. es ist wesentlich für die Performance einer Anwendung, dass möglichst alle benötigten Daten auf einmal downgeloadet werden und überdies die Komprimierung gegebenenfalls zu verwenden ist. Außerdem sollte der Download, wann immer möglich, bei einer Verbindung mit einem WLAN erfolgen, anstelle der Verwendung des Mobilfunknetzes für den Datendownload.

Abstract

Mobile devices have a less powerful CPU and less memory than desktop computers and laptops. Furthermore, the capacity of the battery is also a limitation for the use of mobile devices. In addition, the mobile network does not provide the same high data transfer rates as LAN or WLAN. This all must be taken into account when using web services with mobile devices. Also, the implementation effort is an important criterion for the use of a technology.

For this thesis, Android applications, that can be used as client applications for SOAP-based and RESTful web services, are developed. On the one hand, developing such applications can be used as a proof of concept and the effort for the implementation can be assessed. On the other hand, the client applications implement all necessary methods to determine the usage of CPU and memory, the amount of data transferred, the duration for transmission and processing and the power consumption.

In this thesis, the possibility for using both SOAP-based and RESTful web services with mobile devices and the Android operating system is shown. The effort for the implementation differs only slightly between the used technologies. Also, the difference between the used technologies concerning the usage of CPU and memory and the duration for the processing is insignificant. In contrast, the way how the data is transferred has a huge impact on the performance of the applications. This means that all data, that might be needed, should be transferred at once. Additionally, data compression can be used. And whenever possible, a WLAN connection instead of the mobile network should be used for transferring data.

Inhaltsverzeichnis

1 Einleitung......1

1.1 Motivation......1

1.2 Vorgangsweise......1

1.3 Zielsetzung......2

2 Allgemeines zu Webservices, REST und SOAP......4

2.1 Webservices......4

2.2 REST......5

2.3 SOAP......6

2.4 Zusammenfassung zu den Webservices, REST und SOAP......7

3 Die Serverseite......8

3.1 Verwendete Programmiersprache, Software und die Cloud......8

3.2 Erstellung der Webservices......9

3.3 Zusammenfassung zur serverseitig verwendeten Software, der Cloud und der entwickelten Webservices......13

4 Clientseitige Hard- und Software......14

4.1 Hardware......14

4.2 Entwicklungsumgebung......14

4.3 Die Testanwendungen......14

4.3.1 Ermitteln der CPU-Nutzung......18

4.3.2 Ermitteln des Speicherbedarfs......19

4.3.3 Ermitteln der übertragenen Datenmenge......20

4.3.4 Ermitteln des Stromverbrauchs......20

4.3.5 Ermitteln der Dauer des Testdurchlaufs und der Datenübertragung......23

4.4 Zusammenfassung zur clientseitigen Hard- und Software..............................24

5 Testaufbau, Testabläufe und Messergebnisse..25

5.1 Testaufbau..25

5.2 Tests bzgl. der Übertragungswege und -methoden..29

5.3 Tests bzgl. SOAP vs. REST+XML vs. REST+JSON.....................................39

5.4 Zusammenfassung zu den Tests...46

6 Zusammenfassung...48

7 Literaturverzeichnis..50

Anhang...52

A.1 Python-Skript zum Erstellen der Bilder..52

A.2 Python-Skript zum Erstellen der Hashwerte und Befüllen der Datenbank.....54

Abbildungsverzeichnis

Abbildung 4.1: Testanwendung..18

Abbildung 4.2: Comparison of battery drain rates for our benchmarks (Krintz/Wen/Wolski, 2004, S. 224)..21

Abbildung 5.1: Größe eines einzelnen Datensatzes in Byte......................................26

Abbildung 5.2: Größe von 100 Datensätzen in Form einer Liste in Byte..................27

Abbildung 5.3: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste und Verarbeitungszeit am Server berücksichtigt.....................................32

Abbildung 5.4: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste, ohne Berücksichtigung der Verarbeitungszeit am Server..........................33

Abbildung 5.5: Übertragungsdauer in Sekunden, komprimiert vs. unkomprimiert und Verarbeitungszeit am Server berücksichtigt.....................................34

Abbildung 5.6: Übertragungsdauer in Sekunden, komprimiert vs. unkomprimiert, ohne Berücksichtigung der Verarbeitungszeit am Server..................35

Abbildung 5.7: Datenvolumen in KB, komprimiert vs. unkomprimiert....................36

Abbildung 5.8: Durchschnittlich vorhandene Restkapazität des Akkus in %............37

Abbildung 5.9: utime + stime in Clock Ticks, Liste vs. Einzeln, komprimiert vs. unkomprimiert...41

Abbildung 5.10: Durchschnittswert der PSS in KB während der Testdurchläufe......42

Abbildung 5.11: Durchschnitt der jeweiligen Höchstwerte der RSS in KB...............43

Abbildung 5.12: Durchschnitt der jeweiligen Höchstwerte der VmSize in KB.........44

Abbildung 5.13: Ausführungsdauer in Millisekunden...46

Tabellenverzeichnis

Tabelle 2.1: Ausgewählte, von REST häufig verwendete, Statuscodes (vgl. Fielding / Gettys / Mogul / Frystyk / Masinter / Leach / Berners-Lee, 2013, S. 58 ff)....6

Tabelle 5.1: Größe eines einzelnen Datensatzes in Byte....26

Tabelle 5.2: Größe von 100 Datensätzen in Form einer Liste in Byte....27

Tabelle 5.3: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste und Verarbeitungszeit am Server berücksichtigt....32

Tabelle 5.4: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste, ohne Berücksichtigung der Verarbeitungszeit am Server....33

Tabelle 5.5: Übertragungsdauer in Millisekunden, komprimiert vs. unkomprimiert und Verarbeitungszeit am Server berücksichtigt....34

Tabelle 5.6: Übertragungsdauer in Millisekunden, komprimiert vs. unkomprimiert, ohne Berücksichtigung der Verarbeitungszeit am Server....35

Tabelle 5.7: Datenvolumen in Byte, komprimiert vs. unkomprimiert....36

Tabelle 5.8: Durchschnittlich vorhandene Restkapazität des Akkus in %....37

Tabelle 5.9: utime + stime in Clock Ticks, Liste vs. Einzeln, komprimiert vs. unkomprimiert....40

Tabelle 5.10: Durchschnittswert der PSS in KB während der Testdurchläufe....42

Tabelle 5.11: Durchschnitt der jeweiligen Höchstwerte der RSS in KB....43

Tabelle 5.12: Durchschnitt der jeweiligen Höchstwerte der VmSize in KB....44

Tabelle 5.13: Ausführungsdauer in Millisekunden....45

1 Einleitung

1.1 Motivation

Die Nutzung der IT-Infrastruktur mit mobilen Geräten wie Smartphones und Tablet-PCs gewinnt immer mehr an Bedeutung. Im Rahmen dieser Nutzung werden die mobilen Geräte auch als Clients von Webservices verwendet.

Smartphones und Tablet-PCs unterscheiden sich aber von Desktop-PCs und Laptops unter anderem durch eine weniger leistungsfähige CPU und kleinerem Arbeitsspeicher. Des Weiteren setzt auch die Kapazität des Akkus eine Grenze, inwieweit diese Geräte einsetzbar sind. Darüber hinaus bietet das Mobilfunknetz nicht die gleich hohen Datenübertragungsraten, wie dies im LAN oder WLAN der Fall ist.

Auch der Aufwand für die Implementierung ist ein wichtiger Faktor, ob bestimmte Geräte und Technologien eingesetzt werden können.

1.2 Vorgangsweise

In dieser Arbeit werden die Auswirkungen von RESTful Webservices mit JSON, RESTful Webservices mit XML und SOAP-basierte Webservices auf die CPU-Nutzung, den Speicherbedarf, die übertragenen Datenmengen und den Strombedarf von mobilen Geräten mit dem Android-Betriebssystem untersucht. Des Weiteren wird ermittelt, welche Art von Webservice bzgl. des Implementierungsaufwands als vorteilhaft anzusehen ist. Zusätzlich werden auch die Auswirkungen der verschiedenen Übertragungswege und die Auswirkungen der Komprimierung der übertragenen Daten auf die Performance untersucht. Es wird auch ein Vergleich zwischen geringere Datenmengen oft oder größere Datenmengen weniger oft zu übertragen gezogen.

Um die erforderlichen Werte ermitteln zu können, werden für die unterschiedlichen Arten von Webservices Android-Anwendungen entwickelt. Diese Anwendungen im-

plementieren alle für die CRUD[1]-Operationen erforderlichen Methoden. Das vollständige Implementieren dieser Methoden dient zum Abschätzen des Implementierungsaufwands bei den jeweiligen Webservices. Das Ermitteln der Werte bzgl. der CPU-Nutzung, des Speicherbedarfs, der übertragenen Datenmengen und des Stromverbrauchs, sowie die Auswirkungen der verschiedenen Übertragungswege und der Komprimierung, erfolgt ausschließlich beim Lesen von Daten, da für das Einfügen und Ändern von Daten eine stärkere Benutzer-Interaktion erforderlich ist und dementsprechend der Unterschied zwischen den einzelnen Technologien in den Hintergrund tritt.

Für den Server wird für jede der untersuchten Webservice-Arten jeweils eine Minimal-Variante eines Webservices entwickelt. Diese Services laufen einerseits bei einem PaaS[2]-Provider und andererseits auf einem lokalen Rechner.

Damit können die Tests in für mobile Clients typische Szenarien durchgeführt werden. In dieser Arbeit werden die folgenden drei Szenarien verwendet:

- Der mobile Client greift über das Mobilfunknetz auf einen Service in der Cloud zu.
- Der mobile Client ist mit einem WLAN verbunden und der Zugriff auf den Webservice erfolgt über das Internet.
- Der mobile Client ist mit einem WLAN verbunden und der Server befindet sich in der selben Organisation im LAN.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist das Erkennen der Unterschiede bzgl. der CPU-Nutzung, des Speicherbedarfs, der übertragenen Datenmengen und des Strombedarfs. Dies einerseits zwischen SOAP-basierten und RESTful Webservices und andererseits zwischen JSON und XML. Des Weiteren soll herausgefunden werden, bei welcher der

1 CRUD steht für Create, Read, Update, Delete und bedeutet das Einfügen, Lesen, Verändern und Löschen von in Datenbanken gespeicherten Daten.

2 PaaS steht für Platform as a Service und bezeichnet ein Service-Modell des Cloud-Computings.

genannten Technologien der Implementierungsaufwand geringer ist. Außerdem sollen die Auswirkungen auf die Performance bei der Nutzung der unterschiedlichen Übertragungswege, des Einsatzes der Komprimierung und dem Übertragen von mehreren Datensätzen auf einmal anstelle der Übertragung einzelner Datensätze festgestellt werden.

2 Allgemeines zu Webservices, REST und SOAP

In diesem Kapitel wird allgemein auf Webservices, REST und SOAP eingegangen, sowie jene Bereiche dieser Techniken und Technologien, auf die im weiteren Verlauf dieser Arbeit zurückgegriffen wird, näher betrachtet.

2.1 Webservices

"Web services are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP). As described by the World Wide Web Consortium (W3C), web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks" (Jendrock / Cervera-Navarro / Evans / Gollapudi / Haase / Markito / Srivathsa, 2013, Seite 21-1).

Für Webservices kommen unterschiedliche Techniken und Technologien in Frage. Neben den in dieser Arbeit verwendeten RESTful und SOAP-basierten Webservices gibt es beispielsweise auch noch XML-RPC, welches die Technik XML über HTTP verwendet, um Remote Procedure Calls zu implementieren. Für XML-RPC gibt es auch ein Projekt[3] der Apache Software Foundation.

Welche dieser Technologien, abgesehen von einer ganz speziellen Sichtweise wie in dieser Arbeit die Betrachtungsweise bzgl. mobiler Clients, zum Einsatz kommen soll, hängt naturgemäß von vielen unterschiedlichen Faktoren ab. Pautasso / Zimmermann / Leymann haben die unterschiedlichen architekturellen Aspekte für RESTful und SOAP-basierte Webservices in ihrer Arbeit eingehend untersucht.

"The main conclusion from this comparison is to use RESTful services for tactical, ad hoc integration over the Web (à la Mashup) and to prefer WS-* Web services in professional enterprise application integration scenarios with a longer lifespan and advanced QoS requirements" (Pautasso / Zimmermann / Leymann, 2008, S. 813).

3 http://ws.apache.org/xmlrpc/index.html

2.2 REST

Representational State Transfer (REST) ist ein Architektur-Stil, der von Roy T. Fielding in seiner Dissertation[4] Architectural Styles and the Design of Network-based Software Architectures eingehend untersucht wurde. Für Java ist REST im JSR-000311[5] bzw. JSR-000339[6] spezifiziert.

Von den im RFC 2616 definierten HTTP-Methoden sind besonders die Methoden GET, PUT, POST und DELETE hervorzuheben. Von diesen vier Methoden ist GET als sicher und darüber hinaus GET, PUT und DELETE als idempotent deklariert (vgl. Fielding / Gettys / Mogul / Frystyk / Masinter / Leach / Berners-Lee, 2013, S. 58 ff).

GET dient zum Abholen einer Repräsentation. Mit PUT wird eine bestehende Ressource aktualisiert bzw., wenn diese noch nicht vorhanden ist, erzeugt. Mit POST wird eine neue Ressource unter einer vom Server bestimmten URI angelegt. DELETE löscht eine Ressource (vgl. Tilkov, 2011, S. 51 ff).

In Tabelle 2.1 werden einige ausgewählte, mit Bezug auf REST häufig verwendete, Statuscodes aufgelistet.

200	OK; Die Anfrage wurde erfolgreich verarbeitet. D. h. im Falle von GET, dass im Response die Repräsentation der angeforderten Ressource enthalten ist.
201	Created; dieser Statuscode ist vor allem bei einem POST und in Verbindung mit dem Header-Field Location, das den Pfad zur neu angelegten Ressource enthält, interessant.
204	No Content; wird beispielsweise nach einem erfolgreichen DELETE gesendet.
404	Not Found; bei GET und DELETE konnte die in der URI bezeichnete

4 Online lesbar und downloadbar von http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

5 Downloadbar von http://jcp.org/aboutJava/communityprocess/final/jsr311/

6 Downloadbar von http://download.oracle.com/otndocs/jcp/jaxrs-2_0-fr-eval-spec/index.html

	Ressource nicht gefunden werden.
405	Method Not Allowed; wenn beispielsweise ein DELETE auf eine in der URI bezeichneten Ressource gesendet wird, obwohl diese Methode vom Service nicht implementiert ist.
406	Not Acceptable; beispielsweise wenn von der in der URI bezeichneten Ressource das angeforderte Repräsentationsformat nicht geliefert werden kann (z. B. wurde im Header-Field Accept application/json festgelegt, obwohl nur application/xml zur Verfügung steht).
415	Unsupported Media Type; die vom Client gesendeten Daten werden bei der verwendeten Methode nicht unterstützt (beispielsweise wird bei einem POST application/xml gesendet, obwohl nur application/json verarbeitet werden kann).

Tabelle 2.1: Ausgewählte, von REST häufig verwendete, Statuscodes (vgl. Fielding / Gettys / Mogul / Frystyk / Masinter / Leach / Berners-Lee, 2013, S. 58 ff)

2.3 SOAP

SOAP ist ein W3C-Standard, der auf der Website[7] des World Wide Web Consortiums (W3C) eingesehen werden kann. Im JSR-000224[8] sind XML-basierte, und damit auch SOAP-basierte, Webservices für JAVA spezifiziert.

Mit der Webservice Description Language (WSDL)[9] können Webservices beschrieben werden, die über SOAP-Nachrichten angesprochen werden (vgl. Tilkov, 2011, S. 146 f).

Durch die Beschreibung des Webservices mittels WSDL-Datei ergeben sich beim Entwickeln grundsätzlich zwei unterschiedliche Herangehensweisen. Einerseits kann mit dem Bottom-Up Ansatz zuerst der Java Code entwickelt und die entsprechende WSDL-Datei mit Software-Unterstützung automatisch dazu generiert werden lassen. Andererseits kann aber auch mit dem Top-Down Ansatz zuerst die WSDL-Datei ge-

7 http://www.w3.org/standards/techs/soap

8 Downloadbar von http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index4.html

9 http://www.w3.org/TR/wsdl

schrieben und mit Software-Unterstützung der Java Code automatisch dazu generiert werden lassen.

2.4 Zusammenfassung zu den Webservices, REST und SOAP

In diesem Kapitel wurde auf die Begriffe Webservice, REST und SOAP eingegangen. Darüber hinaus wurden ausgewählte Bereiche dieser Technologien, die auch in dieser Arbeit verwendet werden, näher erörtert.

In den nachfolgenden Kapiteln wird die Serverseite, soweit sie für diese Arbeit relevant ist, behandelt. Anschließend wird auf die clientseitig verwendete Hard- und Software eingegangen, sowie das Erstellen der für diese Arbeit notwendigen Testanwendungen gezeigt. Abschließend wird die Durchführung der Tests erörtert und die Testergebnisse präsentiert.

3 Die Serverseite

Um für die Tests entsprechende Webservices zur Verfügung zu haben, wurden die im Rahmen dieser Arbeit benötigten Webservices entwickelt und betrieben. In diesem Kapitel wird daher die serverseitig verwendete Software, d. h. der Anwendungsserver und der Datenbankserver, sowie die für die Erstellung der Webservices verwendete Programmiersprache und die Entwicklungswerkzeuge vorgestellt. Des Weiteren wird der für diese Arbeit ausgewählte Cloud-Provider vorgestellt und auf das Entwickeln der benötigten Webservices eingegangen.

Die Auswahl der verwendeten Programmiersprache, der Software und des Cloud-Providers erfolgte rein willkürlich und hat keine Auswirkungen auf die Messergebnisse.

3.1 Verwendete Programmiersprache, Software und die Cloud

Als Sprache für die Entwicklung der Webservices wurde Java verwendet.

Als Anwendungsserver wurde im lokalen Netzwerk der JBoss AS in der Version 7.1.1 verwendet. Diese Version ist unter der GNU Lesser General Public License (LGPL), Version 2.1[10] lizensiert. Die Software kann vom Download-Bereich der JBoss-Website[11] downgeloadet werden.

JBoss AS ist nur einer von mehreren möglichen Anwendungsservern im Bereich Java. Mögliche Anwendungsserver, deren Kompatibilität überprüft wurde, sind auf der Oracle-Website[12] aufgelistet.

Die MySQL Community Edition in der Version 5.6.11.0 wurde im lokalen Netzwerk als Datenbank-Server verwendet. Diese Version ist unter der GNU General Public

10 http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

11 http://www.jboss.org/jbossas/downloads

12 http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html

License (GPL)[13] lizensiert und kann vom Download-Bereich der MySQL-Website[14] downgeloadet werden.

Für die Entwicklung der Webservices wurde das JBoss Developer Studio in der Version 6.0.1 verwendet. Diese Entwicklungsumgebung kann von der JBoss-Website[15] downgeloadet werden und ist unter mehreren Lizenzen, wie beispielsweise der LGPL und der Eclipse Public License (EPL)[16], lizensiert. Die Verwendung dieser Entwicklungsumgebung ist insofern vorteilhaft, da alle benötigten Plug-ins bereits installiert sind und damit kein zusätzlicher Arbeitsaufwand erforderlich ist. Für die Erstellung von Webservices ist das Plug-in Forge, das unter der LGPL lizensiert ist, besonders hilfreich. Falls es nicht ohnehin bereits in der Entwicklungsumgebung als Plug-in installiert ist, kann es von der JBoss-Website[17] downgeloadet werden.

Als Cloud-Provider wurde für diese Arbeit OpenShift[18] von Red Hat gewählt. Bei diesem Anbieter können drei voneinander unabhängige Anwendungen mit jeweils 1 GB Speicherplatz kostenlos gehostet werden. Darüber hinaus können bei diesem Anbieter unter anderem auch der JBoss AS und ein MySQL-Datenbankserver genutzt werden.

3.2 Erstellung der Webservices

Das Erstellen der Webservices erfolgte zum überwiegenden Teil mit dem Plug-in Forge im JBoss Developer Studio. Des Weiteren erfolgte das Erstellen der Webservices auf Basis einer bereits bestehenden Datenbanktabelle. Der Erstellungsvorgang bis einschließlich des Objekt-relationalen Mappings (ORM) unter Nutzung von Forge unterscheidet sich bei RESTful Webservices nicht vom Erstellungsvorgang von SOAP-basierten Webservices.

---

13 http://www.gnu.org/licenses/gpl-3.0
14 http://dev.mysql.com/downloads/
15 https://devstudio.jboss.com/earlyaccess/6.0.1.GA.html
16 http://www.eclipse.org/legal/epl-v10.html
17 http://forge.jboss.org/
18 https://www.openshift.com/

Vorausgesetzt, dass in Forge die entsprechenden Plug-ins, d. h. in diesem Fall die Plug-ins hibernate-tools und jboss-as-7, bereits installiert sind, kann wie in Listing 3.1 dargestellt vorgegangen werden. Die restlichen Informationen können vom Entwickler interaktiv eingegeben bzw. die vorgeschlagenen Standardwerte mit ENTER übernommen werden. In Listing 3.1 wird nicht nur die Unterstützung für das ORM hinzugefügt, sondern auch die Java-Klassen entsprechend der in der Datenbank vorhandenen Tabellen angelegt.

```
> new-project --named MeinWebservice
    --topLevelPackage at.fhj.swd10.eibegger.bac2
> persistence setup --provider HIBERNATE --container
    JBOSS_AS7
> generate-entities
```

Listing 3.1: Webservice erstellen bis einschließlich ORM

Für einen RESTful Webservice sind noch die in Listing 3.2 gezeigten Befehle auszuführen.

```
> rest setup --activatorType APP_CLASS
> rest endpoint-from-entity --contentType
    application/json package_mit_model_classes.*
```

Listing 3.2: Befehle zum Erstellen des RESTful Webservices

Damit ist ein einfacher, lauffähiger RESTful Webservice, in diesem Fall mit Unterstützung für JSON, erstellt.

Für einen SOAP-basierten Webservice endet der Erstellungsvorgang in Forge mit dem Befehl `soap setup`. Der Rest ist manuell zu erstellen. In diesem Fall und unter Verwendung des Bottom-Up Ansatzes ist dies das Schreiben einer Klasse mit den für einen Webservice notwendigen Annotationen und der für die gewünschte

Funktionalität erforderlichen Methoden. Zum Bottom-Up Ansatz siehe auch das Kapitel 2.3. Damit diese Klasse die gleiche Funktionalität wie der oben erstellte RESTful Webservice bietet, beschränkt sich der zusätzliche Aufwand auf das manuelle Erstellen von rund 150 Zeilen Code.

Darüber hinaus muss in jedem Fall noch die von Forge generierte und in Listing 3.3 gezeigte persistence.xml entsprechend der verwendeten Umgebung angepasst werden. Der Wert in Zeile 9 sollte beispielsweise von `create-drop` auf `update` geändert werden. Auch die Zeilen 10 – 12 sind den jeweiligen Erfordernissen anzupassen. Außerdem ist in Zeile 6 die tatsächlich zu verwendende Datasource anzugeben.

```
 1. <?xml version="1.0" encoding="UTF-8" standalone="no"?>
 2. <persistence xmlns="http://java.sun.com/xml/ns/persistence"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     version="2.0"
     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
     >
 3. <persistence-unit name="forge-default" transaction-
     type="JTA">
 4. <description>Forge Persistence Unit</description>
 5. <provider>org.hibernate.ejb.HibernatePersistence</provider>
 6. <jta-data-source>java:jboss/datasources/ExampleDS</jta-
     data-source>
 7. <exclude-unlisted-classes>false</exclude-unlisted-classes>
 8. <properties>
 9. <property name="hibernate.hbm2ddl.auto" value="create-
     drop"/>
10. <property name="hibernate.show_sql" value="true"/>
11. <property name="hibernate.format_sql" value="true"/>
12. <property
     name="hibernate.transaction.flush_before_completion"
     value="true"/>
13. </properties>
```

```
14. </persistence-unit>
15. </persistence>
```

Listing 3.3: von Forge generierte persistence.xml

Dem Projekt ist entweder eine Datasource-Datei, wie in Listing 3.4 gezeigt, hinzuzufügen oder in der Konfigurationsdatei des JBoss AS, der standalone.xml, der entsprechende Eintrag im Knoten `<datasources>` hinzuzufügen.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <datasources
   xmlns="http://www.jboss.org/ironjacamar/schema">
3. <datasource jndi-
   name="java:jboss/datasources/TestArticlesDS" enabled="true"
   use-java-context="true" pool-name="TestArticlesDS">
4. <connection-
   url>jdbc:mysql://127.2.183.130:3306/jaxws</connection-url>
5. <driver>mysql-connector-java-5.1.24-bin.jar</driver>
6. <pool></pool>
7. <security>
8. <user-name>MeinName</user-name>
9. <password>MeinPasswort</password>
10. </security>
11.</datasource>
12. </datasources>
```

Listing 3.4: Beispiel einer Datasource-Datei

Die in Listing 3.4 gezeigte Datasource-Datei ist in den Zeilen 4 und 5, sowie in den Zeilen 8 und 9, entsprechend der tatsächlich verwendeten Umgebung anzupassen.

Der Datenbanktreiber ist entweder in das Verzeichnis deployments des JBoss AS zu kopieren oder im Verzeichnis modules im entsprechenden Unterverzeichnis abzulegen und in der standalone.xml im Knoten `<drivers>` einzutragen. Die in Listing 3.4 gezeigte Datasource-Datei ist in Zeile 5 entsprechend anzupassen. Das

Listing 3.5 zeigt einen möglichen Eintrag in der standalone.xml im Knoten `<drivers>`.

```
1. <driver name="mysql" module="com.mysql.jdbc">
2. <xa-datasource-
     class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-
     datasource-class>
3. </driver>
```

Listing 3.5: Beispiel eines Knotens <driver> in der standalone.xml

3.3 Zusammenfassung zur serverseitig verwendeten Software, der Cloud und der entwickelten Webservices

In diesem Kapitel wurde die serverseitig verwendete Software und der ausgewählte Cloud-Provider kurz beschrieben. Die Auswahl der verwendeten Programmiersprache, der Software und des Cloud-Providers erfolgte, wie bereits eingangs erwähnt, rein willkürlich. Das Übereinstimmen der lokal verwendeten Server mit den in der Cloud genutzten Servern hat aber naturgemäß den Vorteil, dass durch das zusätzliche Hosten in der Cloud der Aufwand bei der Entwicklung und dem Deployment nur geringfügig steigt.

Des Weiteren wurde auf die Erstellung einfacher Webservices, wie sie für diese Arbeit benötigt werden, eingegangen. Dabei zeigte sich, dass aufgrund der besseren Unterstützung der verwendeten Software für RESTful Webservices der Aufwand für die Erstellung der RESTful Webservices geringfügig geringer war als der Aufwand für die Erstellung des SOAP-basierten Webservices.

4 Clientseitige Hard- und Software

In diesem Kapitel wird die bei den Tests verwendete Hardware, sowie die für das Erstellen der Testanwendungen verwendete Entwicklungsumgebung vorgestellt. Des Weiteren wird auf die Erstellung der Testanwendungen eingegangen und erläutert, wie die Testanwendungen die Messwerte ermitteln.

4.1 Hardware

Für das Durchführen der Tests wurde ein LG® P880 Optimus 4X HD verwendet. Dieses Smartphone hat einen Quad-Core-Prozessor mit 1,5 GHz, 1GB RAM, einen Akku mit 2.150 mAh und unterstützt bzgl. WLAN die Standards 802.11 b/g/n. Als Betriebssystem wird Android 4.0.3 verwendet.

4.2 Entwicklungsumgebung

Für die Entwicklung der Android-Anwendungen wurde Eclipse in der Version 4.2.2 und das Android Development Toolkit verwendet. Diese Entwicklungsumgebung ist unter der EPL lizensiert und kann vom Downloadbereich der Eclipse-Website[19] downgeloadet werden.

4.3 Die Testanwendungen

Für das Ermitteln der Werte wurden eigene Testanwendungen für die jeweiligen Arten von Webservices entwickelt.

Die Clients der RESTful Webservices wurden ohne der Nutzung von zusätzlichen Bibliotheken entwickelt. Für die Verbindung wurde die Klasse `java.net.HttpURLConnection` verwendet.

Für den Client des SOAP-basierten Webservices wurde die Bibliothek ksoap2-

19 http://www.eclipse.org/downloads/

android verwendet. Diese Bibliothek ist unter der MIT Lizenz[20] lizensiert und kann von Google Code[21] downgeloadet werden. Ein Vorteil von SOAP-basierten Webservices, die automatische Generierung von Code auf Basis der Beschreibung des Services in einer WSDL-Datei, kann hier aber nicht genutzt werden. Listing 4.1 zeigt, wie einfach die Nutzung eines SOAP-basierten Webservices für einen Client, der die JavaSE Java Runtime Edition (JRE) zur Verfügung hat, ist.

Mit wsimport bzw. wsconsume, bei Verwendung von JBoss AS, kann beispielsweise mit dem folgenden Befehl der benötigte Code generiert werden.

```
wsconsume.bat --source=src http://localhost/articles?wsdl
```

Nach Importieren des generierten Codes in ein Projekt kann dann, wie in Listing 4.1 gezeigt, der Webservice genutzt werden um beispielsweise ein Objekt abzurufen.

```
ArticlesService service = new ArticlesService();
Articles articles = service.getArticlesPort();
Article article = articles.get(42);
```

Listing 4.1: Zugriff auf einen Webservice unter Verwendung von mittels wsconsume generiertem Code

Im Gegensatz zum Beispiel in Listing 4.1 musste bei Android und der Verwendung der ksoap2-android-Bibliothek die WSDL-Datei manuell analysiert werden, um die erforderlichen Operationen, Parameter, etc. zu ermitteln. Der in Listing 4.2 gezeigte Code-Ausschnitt implementiert die gleiche Funktionalität unter Verwendung der ksoap2-android-Bibliothek wie der Code in Listing 4.1 unter Verwendung des generierten Codes.

```
SoapObject request = new SoapObject(NAMESPACE, GET);
request.addProperty("id",42);
SoapSerializationEnvelope envelope = new
```

20 http://opensource.org/licenses/mit-license.php

21 http://code.google.com/p/ksoap2-android/

```
        SoapSerializationEnvelope(SoapEnvelope.VER10);
    envelope.setOutputSoapObject(request);
    HttpTransportSE transport = new HttpTransportSE(uri);
    transport.call(NAMESPACE + GET, envelope);
    if(envelope.bodyIn instanceof SoapObject)
        SoapObject so = (SoapObject)envelope.bodyIn;
```

Listing 4.2: Zugriff auf einen Webservice unter Verwendung von ksoap2-android

Während in Listing 4.1 das Objekt bereits im entsprechenden Typ vorliegt, muss in Listing 4.2 das SoapObject noch entsprechend ausgewertet bzw. mit den darin enthaltenen Informationen ein Objekt vom gewünschten Typ erzeugt werden. Listing 4.3 zeigt das Ermitteln der im SOAP-Objekt enthaltenen Informationen.

```
SoapObject article =
    (SoapObject)so.getProperty("return");
String articlename =
    article.getPrimitivePropertySafelyAsString("artic
    lename");
```

Listing 4.3: Zugriff auf Felder in einem Objekt vom Typ SoapObject

Zum Vergleich dazu zeigt der Code-Ausschnitt in Listing 4.4, wie ein RESTful Webservice genutzt werden kann, um ein JSON-Objekt zu erhalten und von diesem Objekt ein Feld auszulesen. Für das Abrufen von Daten wird GET verwendet. Für das Anlegen, Ändern und Löschen von Daten wird, wie in Kapitel 2.2 geschildert, POST, PUT und DELETE verwendet.

```
URL url = new URL(uri);
HttpURLConnection urlConn =
    (HttpURLConnection)url.openConnection();
urlConn.setRequestProperty("Accept","application/json
    ");
```

```
int statusCode = urlConn.getResponseCode();
if(statusCode == HttpURLConnection.HTTP_OK){
    InputStream istream = new
        BufferedInputStream(urlConnection.getInputStr
        eam());
    JSONObject json = new JSONObject(new
        Scanner(istream).useDelimiter("\\A").next();
    String articlename =
        json.getString("articlename");
}
```

Listing 4.4: Abrufen eines JSON-Objekts von einem RESTful Webservice und Zugriff auf ein Feld des abgerufenen Objekts

In Listing 4.4 steht `HTTP_OK` für den Statuscode 200. Die für diese Arbeit entwickelten Testanwendungen zeigen den Statuscode teilweise, je nach der verwendeten Methode, auch am Ende eines Testdurchlaufs an. Für eine Liste mit ausgewählten, im Bereich REST häufig verwendeten, Statuscodes siehe auch Tabelle 2.1.

Trotz des zusätzlichen Aufwands bei der Nutzung des SOAP-basierten Webservices war der Aufwand für die Entwicklung dieses Clients nur geringfügig größer als der Aufwand für die Entwicklung der Clients für die RESTful Webservices.

Abbildung 4.1 zeigt eine dieser Testanwendungen. Diese Testanwendungen implementieren einerseits alle Methoden, um die CRUD-Operationen, die der Webservice anbietet, nutzen zu können. Andererseits ist auch die für das Ermitteln der Messwerte notwendige Funktionalität implementiert. Damit können unabhängig von der Entwicklungsumgebung oder sonstigen Hilfsmitteln die Messungen durchgeführt werden. Nach jedem Testdurchlauf werden die ermittelten Werte unterhalb der Schaltflächen angezeigt. Zusätzlich können die ermittelten Werte in einer csv-Datei gespeichert werden, um die Werte später analysieren zu können.

Abbildung 4.1: Testanwendung

4.3.1 Ermitteln der CPU-Nutzung

Für das Ermitteln der CPU-Nutzung wurden zwei verschiedene Ansätze gewählt.

Einerseits bieten die Klassen `android.os.Process` und `android.os.Debug` Methoden zum Bestimmen der CPU-Zeit des jeweiligen Prozesses bzw. Threads, wie die nachfolgenden beiden Code-Zeilen zeigen (vgl. Android-Reference Debug, 2013 und Android-Reference Process, 2013).

```
long cputime = android.os.Process.getElapsedCpuTime();
long tcputime = android.os.Debug.threadCpuTimeNanos();
```

Andererseits können Informationen auch aus der Datei stat aus dem Verzeichnis proc ausgewertet werden.

Die Datei stat enthält unter anderem die Werte für utime und stime in Clock Ticks (vgl. Kerrisk, 2013).

Die Werte utime und stime werden beispielsweise auch beim Ausführen des Befehls top ausgewertet. Der Code-Ausschnitt in Listing 4.5 zeigt das Ermitteln dieser Werte, so wie es in den Testanwendungen erfolgt.

```
RandomAccessFile raf = new RandomAccessFile("/proc/"
    + Integer.toString(pid) + "/stat", "r");
String statline = raf.readLine();
String[] values = statline.split("[ ]+");
long utime = Long.valueOf(values[13]);
long stime = Long.valueOf(values[14]);
```

Listing 4.5: Ermitteln von utime und stime aus der Datei stat

4.3.2 Ermitteln des Speicherbedarfs

Für das Feststellen des Speicherbedarfs wurden drei Werte ermittelt. Der Durchschnittswert der Proportional Set Size (PSS) wurde aus den Werten zu Beginn des Testlaufs, während des Testlaufs nach jedem ausgewerteten Datensatz und am Ende des Testlaufs berechnet. Des Weiteren wurde der Maximalwert der Resident Set Size (RSS) und der VmSize am Ende des Testdurchlaufs festgestellt.

Das Ermitteln der Werte erfolgte einerseits mittels der Klasse `android.os.Debug` und deren Methode `getPss()` (vgl. Android-Reference Debug, 2013).

Andererseits wurden die Werte für RSS und VmSize aus der Datei `/proc/[pid]/status` ausgewertet bzw. da die Maximalwerte dieser Werte erforderlich sind, die Werte für VmHWM und VmPeak ausgelesen (vgl. Kerrisk, 2013).

Der Code-Ausschnitt in Listing 4.6 zeigt, wie das Ermitteln des Werts für VmPeak aus der Datei status in den Testanwendungen erfolgt.

```
BufferedReader reader = new BufferedReader(new
    InputStreamReader(new FileInputStream("/proc/" +
    Integer.toString(pid) + "/status")), 2048);
while((line = reader.readLine()) != null){
    line = line.trim();
    if(line.startsWith("VmPeak")){
      String[] values = line.split("[ \t]+");
      int ret = Integer.valueOf(values[1]);
      break;
    }
}
```

Listing 4.6: Ermitteln von VmPeak aus der Datei status

4.3.3 Ermitteln der übertragenen Datenmenge

Für das Ermitteln der übertragenen Datenmenge wurde die Klasse `android.net.TrafficStats` und deren Methoden `getMobileTxBytes()`, `getMobileRxBytes()`, `getUidTxBytes(uid)`, `getUidRxBytes(uid)`, `getTotalTxBytes()`, `getTotalRxBytes()` verwendet (vgl. Android-Reference TrafficStats, 2013).

Zwar können auch bzgl. der übertragenen Datenmengen die entsprechenden Dateien aus dem Verzeichnis proc ausgewertet werden, da aber alle benötigten Daten mittels der Klasse `android.net.TrafficStats` ermittelt werden können, wurde hier auf ein direktes Auslesen von Dateien verzichtet.

4.3.4 Ermitteln des Stromverbrauchs

Auf den Stromverbrauch einer Anwendung hinsichtlich eines bestimmten

Subsystems, wie CPU oder Arbeitsspeicher, kann aufgrund der verwendeten Algorithmen geschlossen werden, wenn für die einzelnen Anweisungen der Algorithmen entsprechende Benchmarks vorhanden sind (vgl. Krintz/Wen/Wolski, 2004, S. 224 ff).

Abbildung 4.2 zeigt das Entleeren des Akkus bei der Ausführung der verschiedenen Anweisungen für die Ermittlung der jeweiligen Benchmarks (vgl. ebd., S. 225 f).

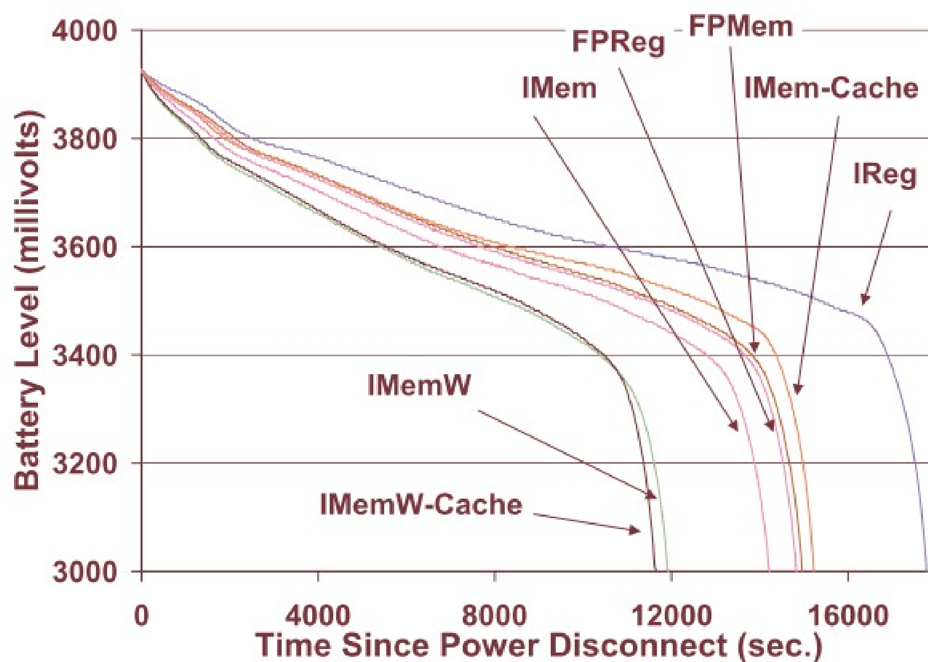

Abbildung 4.2: Comparison of battery drain rates for our benchmarks (Krintz/Wen/Wolski, 2004, S. 224)

Um den Stromverbrauch einer Anwendung bestimmen zu können, müssen bei der obigen Vorgehensweise aber für alle bei der Ausführung der Anwendung ausgeführten Anweisungen und den jeweils beteiligten Subsystemen entsprechende Benchmarks vorhanden sein. D. h. gegebenenfalls nicht nur für die Subsysteme CPU und Arbeitsspeicher, sondern auch Speichermedien, Teile die für das Versenden und Empfangen von Daten benötigt werden, Sensoren, etc.

Aufgrund der Schwierigkeit des Ermittelns aller erforderlichen Benchmarks aller

beteiligter Subsysteme wurde in dieser Arbeit der Stromverbrauch mittels eines Objekts vom Typ `android.content.Intent`, das als Broadcast-Receiver für den Empfang der entsprechenden Werte mittels der Klasse `android.content.Context` registriert wurde, festgestellt (vgl. Android-Reference Context, 2013).

Auf eine fortlaufende Verarbeitung der entsprechenden Werte wurde aber verzichtet und stattdessen die Werte zu Beginn und am Ende des Testlaufs abgefragt, wie der Code-Ausschnitt in Listing 4.7 zeigt.

```
IntentFilter filter = new
    IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent bs = context.registerReceiver(null, filter);
int scale =
    bs.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
int level =
    bs.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int voltage =
    bs.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1);
```

Listing 4.7: Ermitteln der Werte für Batterie-Scale, Batterie-Level und Batterie-Voltage mittels der Klasse android.content.Intent als Broadcast-Receiver

Um eine Gegenprobe zu haben und auch eine zeitnahe Messung sicherzustellen, wurden zusätzlich zur obigen Vorgangsweise die Werte auch direkt aus dem Verzeichnis `/sys/class/power_supply/battery` ermittelt, wie der Code-Ausschnitt in Listing 4.8 darstellt.

```
RandomAccessFile rafc = new
    RandomAccessFile("/sys/class/power_supply/battery
    /capacity", "r");
int levelF = Integer.valueOf(rafc.readLine());
```

```
RandomAccessFile rafv = new
    RandomAccessFile("/sys/class/power_supply/battery
    /voltage_now", "r");
int voltageF = Integer.valueOf(rafv.readLine());
```

Listing 4.8: Ermitteln der Werte für capacity und voltage aus dem Verzeichnis /sys/class/power_supply/battery

Es ist allerdings anzumerken, dass das Ermitteln des Stromverbrauchs bei der in dieser Arbeit verwendeten Vorgangsweise ein Problem darstellt. Einerseits kann der Stromverbrauch über das Ermitteln der Werte, wie oben dargestellt, nur hinsichtlich des gesamten Gerätes und nicht hinsichtlich einer bestimmten Anwendung bestimmt werden. Dementsprechend muss auf Auswirkungen anderer Anwendungen geachtet bzw. können diese nie ganz ausgeschlossen werden. Andererseits wird die Restkapazität nur in ganzen Prozent angegeben und ist damit für einen kurz laufenden Test nicht fein genug abgestuft. Auch das Heranziehen der Spannung des Akkus und damit der Spannungsabfall zwischen Beginn und Ende eines Tests stellt ein Problem dar, da der Spannungsabfall nicht linear erfolgt, wie auch aus der Abbildung 4.2 ersichtlich ist.

Aufgrund der oben genannten Probleme ist ein Ermitteln des Stromverbrauchs grundsätzlich nur bei lang laufenden Tests möglich und die genannten Probleme sind beim Interpretieren der ermittelten Werte entsprechend zu berücksichtigen.

4.3.5 Ermitteln der Dauer des Testdurchlaufs und der Datenübertragung

Für das Messen einer Zeitdauer bietet Android verschiedene Klassen an. In dieser Arbeit wurde die Methode `elapsedRealtime()` der Klasse `android.os.SystemClock` verwendet (vgl. Android-Reference SystemClock, 2013).

Die Dauer des Testdurchlaufs und der Datenübertragung wird mittels der oben genannten Methode zu Beginn und am Ende des Testdurchlaufs bzw. zu Beginn und am Ende der Datenübertragung festgestellt.

4.4 Zusammenfassung zur clientseitigen Hard- und Software

In diesem Kapitel wurde die clientseitig verwendete Hardware und die für das Erstellen der Client-Anwendungen verwendete Entwicklungsumgebung vorgestellt.

Es wurde auf die Entwicklung der Client-Anwendungen eingegangen und gezeigt, dass bei SOAP-basierten Webservices der Entwicklungsaufwand geringfügig höher ist als bei RESTful Webservices. Darüber hinaus wurde für die Nutzung des SOAP-basierten Webservices eine zusätzliche Bibliothek verwendet. Dementsprechend ergibt sich ein leichter Vorteil für RESTful Webservices gegenüber SOAP-basierten Webservices.

Es wurde auch gezeigt, wie das Ermitteln der erforderlichen Messwerte erfolgt. Dabei ist anzumerken, dass das Ermitteln des Stromverbrauchs problematisch ist. Dies ist auch beim Interpretieren der ermittelten Werte für den Stromverbrauch zu berücksichtigen.

5 Testaufbau, Testabläufe und Messergebnisse

In diesem Kapitel wird die Durchführung der Tests dargelegt und die Messergebnisse präsentiert.

5.1 Testaufbau

Für die Tests wurden Daten in einer Datenbank-Tabelle mit der in Listing 5.1 gezeigten Definition gespeichert.

```
CREATE TABLE IF NOT EXISTS `mydb`.`article` (
`id` INT NOT NULL AUTO_INCREMENT ,
`last_update` TIMESTAMP NOT NULL DEFAULT CURRENT_TI-
    MESTAMP ON UPDATE CURRENT_TIMESTAMP ,
`articlename` VARCHAR(45) NULL ,
`hashvalue` VARCHAR(128) NULL ,
`hashvaluebase64` VARCHAR(128) NULL ,
`picturebase64` MEDIUMTEXT NULL ,
PRIMARY KEY (`id`) )
ENGINE = InnoDB DEFAULT CHARSET=utf8$$;
```

Listing 5.1: SQL-Skript zum Anlegen der Tabelle

Für das Generieren der Testdaten wurden die im Anhang A.1 und im Anhang A.2 dargestellten Python-Skripte verwendet. Das erste Python-Skript erstellt Bilder mit zufälligen Pixel-Werten mit einer vorher definierten Größe. Das zweite Python-Skript erstellt Base64-codierte Zeichenketten von den generierten Bildern, berechnet den SHA256-Hashwert des Bildes und der Base64-codierten Zeichenkette und speichert diese Daten in der Datenbanktabelle. Auf diese Weise ist es einfach möglich, viele unterschiedliche Datensätze mit einer bestimmten Größe zu erstellen.

Die Werte in den Datenbankfeldern hashvaluebase64 und picturebase64 können am

Client nicht nur dazu verwendet werden, die fehlerfreie Übertragung zu überprüfen, sondern durch das Arbeiten mit den Daten am Client kann der Unterschied zwischen XML und JSON bzgl. der Ressourcenanforderungen festgestellt werden.

Für die in dieser Arbeit durchgeführten Testdurchläufe wurden Bilder in drei verschiedenen Größen generiert. Aufgrund der Größe der Bilder enthielten die vier Textfelder in der Datenbank Zeichenketten mit einer Gesamtlänge von 641, 10.349 und 40.212 Zeichen. Diese Zeichen zusammen mit der Zahl und dem Timestamp, sowie den zusätzlichen Daten entsprechend der Datenformate bzw. der Art des Webservices ergibt die in Tabelle 5.1 und Abbildung 5.1 gezeigten Größen bei der Übertragung als einzelner Datensatz bzw. die in Tabelle 5.2 und Abbildung 5.2 gezeigten Größen bei der Übertragung von 100 Datensätzen in Form einer Liste.

	SOAP	**XML**	**JSON**
klein	1.025	894	751
mittel	10.733	10.602	10.459
groß	40.595	40.464	40.321

Tabelle 5.1: Größe eines einzelnen Datensatzes in Byte

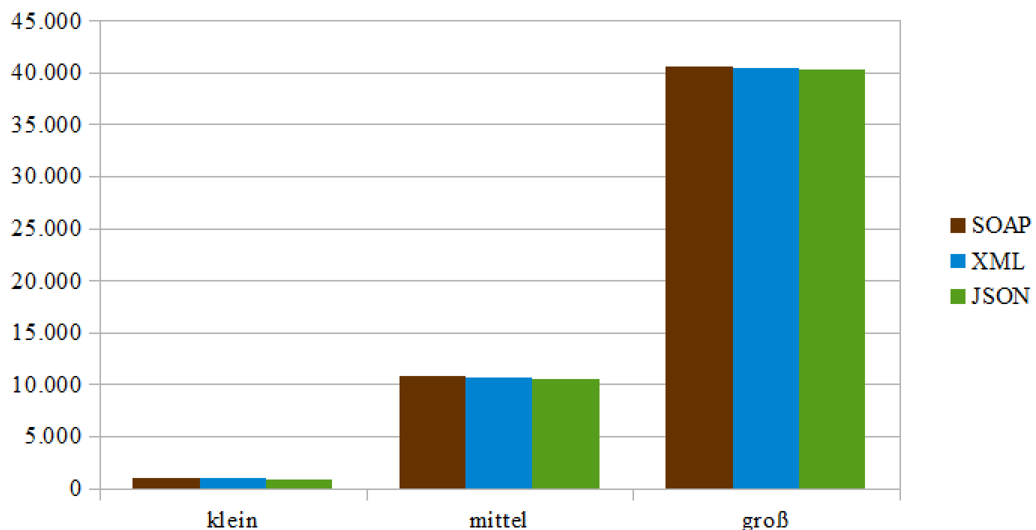

Abbildung 5.1: Größe eines einzelnen Datensatzes in Byte

	SOAP	XML	JSON
klein	83.896	83.980	75.201
mittel	1.054.696	1.054.780	1.046.001
groß	4.040.896	4.040.980	4.032.201

Tabelle 5.2: Größe von 100 Datensätzen in Form einer Liste in Byte

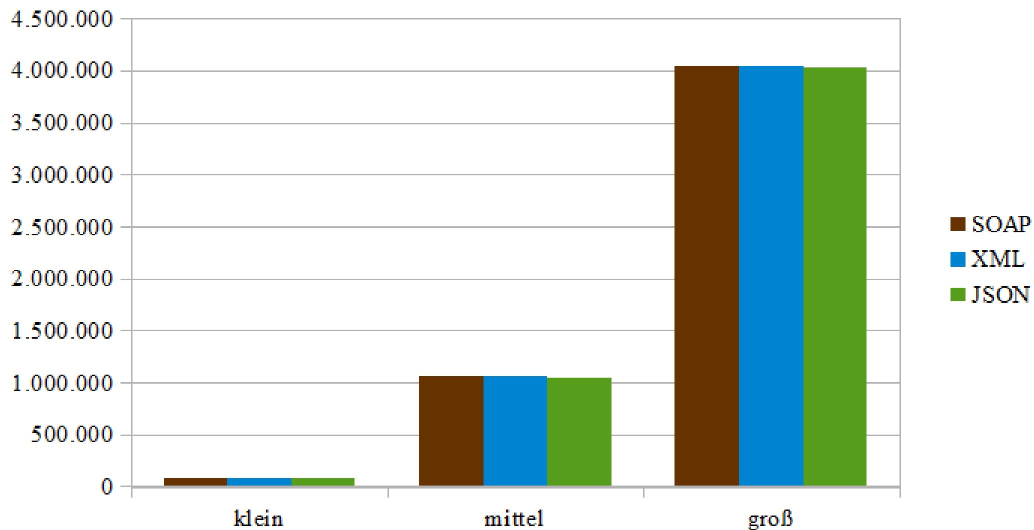

Abbildung 5.2: Größe von 100 Datensätzen in Form einer Liste in Byte

Die in dieser Arbeit verwendeten Webservices in Verbindung mit der in Listing 5.1 gezeigten Tabelle ergaben den in Listing 5.2, Listing 5.3 und Listing 5.4 gezeigten Response beim Abruf eines einzelnen Datensatzes. Daraus ist auch der Unterschied zwischen den verwendeten Technologien bzgl. der Datengröße ersichtlich. Die Felder hashvalue, hashvaluebase64 und picturebase64 werden aus Gründen der Übersichtlichkeit jeweils gekürzt dargestellt.

```
<soap:Envelope xmlns:soap="http://schemas.xml-
    soap.org/soap/envelope/">
<soap:Body>
<ns2:getResponse xmlns:ns2="http://ws.bac2.ei-
    begger.fhj.at/">
```

```
<return>
<articlename>image4910.bmp</articlename>
<hashvalue>1002d40033721c09b66ba695...</hashvalue>
<hashvaluebase64>896464b0609c...</hashvaluebase64>
<id>3210</id>
<lastUpdate>2013-08-02T13:45:47-04:00</lastUpdate>
<picturebase64>Qk0AAAAAAAAADYA...</picturebase64>
</return>
</ns2:getResponse>
</soap:Body>
</soap:Envelope>
```


Listing 5.2: Response des SOAP-basierten Webservices beim Abruf eines einzelnen Datensatzes


```
<?xml version="1.0" encoding="UTF-8"
    standalone="yes"?>
<article>
<articlename>image4910.bmp</articlename>
<hashvalue>1002d40033721c09b66ba695...</hashvalue>
<hashvaluebase64>896464b0609c...</hashvaluebase64>
<id>3210</id>
<lastUpdate>2013-08-06T14:32:11-04:00</lastUpdate>
<picturebase64>Qk0AAAAAAAAADYA...</picturebase64>
</article>
```


Listing 5.3: Response des RESTful Webservices mit XML beim Abruf eines einzelnen Datensatzes


```
{
"id":3210,
"lastUpdate":1375468053000,
"articlename":"image4910.bmp",
```

```
"hashvalue":"1002d40033721c09b66ba695...",
"hashvaluebase64":"896464b0609c...",
"picturebase64":"Qk0AAAAAAAAADYA..."
}
```

Listing 5.4: Response des RESTful Webservices mit JSON beim Abruf eines einzelnen Datensatzes

Für den Request hin zu einem RESTful Webservice reicht zum Abrufen von Daten ein einfacher HTTP-GET-Request aus. Bei der Nutzung des SOAP-basierten Webservices muss ein entsprechendes XML gesendet werden. Dieses XML ist in Listing 5.5 abgebildet und die zu übertragenden Datenmengen erhöhen sich dementsprechend.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xml-
    soap.org/soap/envelope/" xmlns:ws="http://ws.-
    bac2.eibegger.fhj.at/">
<soapenv:Header/>
<soapenv:Body>
<ws:get>
<id>3210</id>
</ws:get>
</soapenv:Body>
</soapenv:Envelope>
```

Listing 5.5: Request eines einzelnen Datensatzes von einem SOAP-basierten Webservice

5.2 Tests bzgl. der Übertragungswege und -methoden

Bei diesen Tests lag der Fokus auf den Unterschieden zwischen den einzelnen Übertragungswegen, d. h. auf den für mobile Clients typischen Szenarien:

- Der mobile Client greift über das Mobilfunknetz auf einen Service in der Cloud zu.

- Der mobile Client ist mit einem WLAN verbunden und der Zugriff auf den Webservice erfolgt über das Internet.
- Der mobile Client ist mit einem WLAN verbunden und der Server befindet sich in der selben Organisation im LAN.

Des Weiteren sollten einerseits die Auswirkungen von komprimiert vs. unkomprimiert und andererseits mehrere Datensätze auf einmal vs. einzelner Datensätze mehrmals festgestellt werden.

Aufgrund der Länge der einzelnen Durchläufe konnte auch der Stromverbrauch erhoben werden.

Für die Übertragung im Mobilfunknetz stand eine Übertragungsrate von 21 Mbit/s für den Download und 5 Mbit/s für den Upload zur Verfügung. Als Signalstärke wurde -81 dBm am Smartphone angezeigt.

Für das Szenario WLAN mit Internetzugriff über Kabel wurden die Tests am Campus der FH Joanneum in Kapfenberg durchgeführt und die dort vorhandene Infrastruktur benutzt.

Im Szenario WLAN und der Server befindet sich in der selben Organisation im LAN wurde als zentraler Netzwerkknoten ein WLAN-Router verwendet, der die Standards 802.11 b/g/n unterstützt.

Je Übertragungsweg und je Webservice wurden nacheinander folgende Daten abgerufen:

- fünfmal 100 kleine Datensätze als Liste unkomprimiert
- fünfmal je 100 kleine Datensätze einzeln unkomprimiert
- fünfmal 100 kleine Datensätze als Liste komprimiert
- fünfmal je 100 kleine Datensätze einzeln komprimiert

- fünfmal 100 mittlere Datensätze als Liste unkomprimiert
- fünfmal je 100 mittlere Datensätze einzeln unkomprimiert
- fünfmal 100 mittlere Datensätze als Liste komprimiert
- fünfmal je 100 mittlere Datensätze einzeln komprimiert
- fünfmal 100 große Datensätze als Liste unkomprimiert
- fünfmal je 100 große Datensätze einzeln unkomprimiert
- fünfmal 100 große Datensätze als Liste komprimiert
- fünfmal je 100 große Datensätze einzeln komprimiert

Jeweils vor dem Test je Übertragungsweg und Webservice wurde der Akku voll aufgeladen, d. h. bis zur entsprechenden Meldung am Smartphone geladen.

Für das Feststellen der Übertragungsdauer wurde die Zeit vom Versenden der Anforderung bis zum vollständigen Erhalt der angeforderten Daten und das gegebenenfalls notwendige Dekomprimieren gemessen.

Am Server wurde die Zeit für die Verarbeitung der Anforderung protokolliert und bei der entsprechenden Auswertung von der am Client ermittelten Zeit abgezogen. Damit wurde der Einfluss des Servers auf die gemessene Zeit am Client weitestgehend berücksichtigt.

Entsprechend den drei Webservices, die für diese Tests genutzt wurden, wurden drei Anwendungen entsprechend dem Kapitel 4.3 verwendet.

Tabelle 5.3 und Abbildung 5.3 bzw. Tabelle 5.4 und Abbildung 5.4 zeigen die ermittelten Werte für die Übertragungsdauer in Sekunden je Übertragungsweg und aufgeschlüsselt nach der Größe der Datensätze und ob sie Einzeln oder als Liste übertragen wurden. In Tabelle 5.3 und Abbildung 5.3 wird die am Server protokollierte Verarbeitungszeit entsprechend berücksichtigt. Besonders deutlich geht

bei diesem Test der Unterschied zwischen den Übertragungswegen, sowie der Unterschied zwischen der Übertragung von mehrmals einzelnen Datensätzen gegenüber mehreren Datensätzen auf einmal, hervor.

	WLAN	WLAN und Kabel	Mobilfunk
Liste – kleiner DS	3	20	64
Einzeln – kleiner DS	37	713	1.066
Liste – mittlerer DS	14	82	121
Einzeln – mittlerer DS	55	790	1.191
Liste – großer DS	18	81	73
Einzeln – großer DS	107	1.175	1.773

Tabelle 5.3: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste und Verarbeitungszeit am Server berücksichtigt

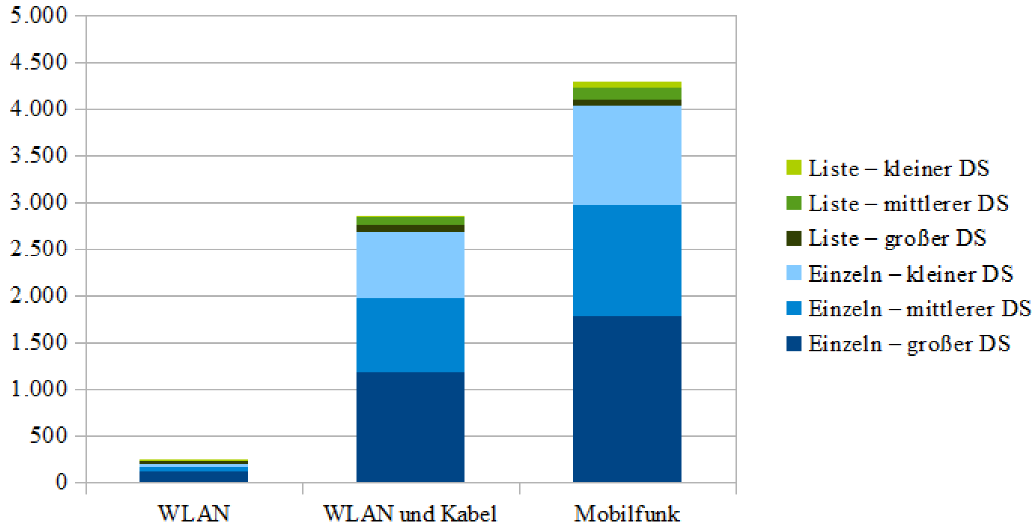

Abbildung 5.3: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste und Verarbeitungszeit am Server berücksichtigt

	WLAN	WLAN und Kabel	Mobilfunk
Liste – kleiner DS	4	25	65
Einzeln – kleiner DS	48	755	1.088
Liste – mittlerer DS	17	114	189
Einzeln – mittlerer DS	66	820	1.208
Liste – großer DS	54	312	308
Einzeln – großer DS	126	1.209	1.803

Tabelle 5.4: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste, ohne Berücksichtigung der Verarbeitungszeit am Server

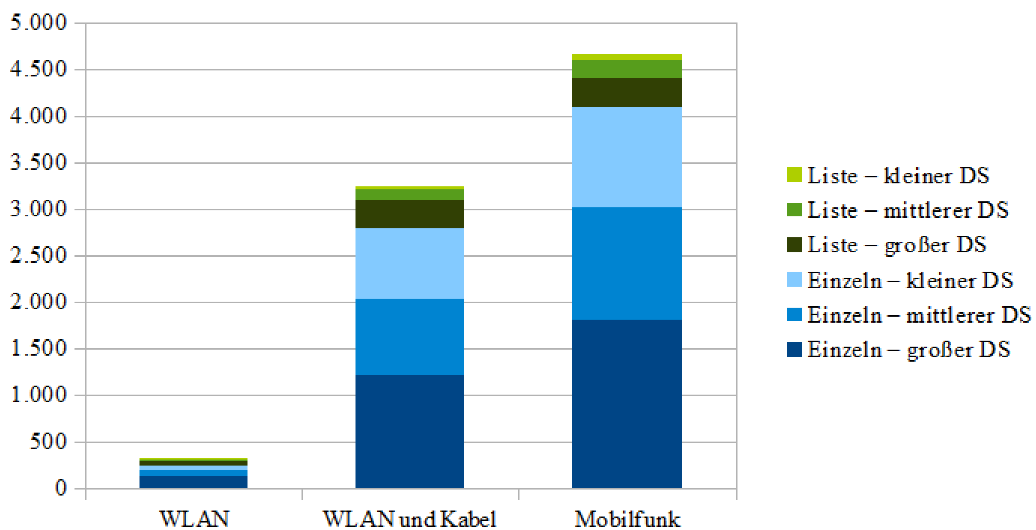

Abbildung 5.4: Übertragungsdauer in Sekunden, einzelne Datensätze vs. Liste, ohne Berücksichtigung der Verarbeitungszeit am Server

Tabelle 5.5 und Tabelle 5.6 zeigen die ermittelten Werte für die Übertragungsdauer in Millisekunden je Übertragungsweg und Webservice und aufgeschlüsselt nach komprimiert oder unkomprimiert. In Tabelle 5.5 wird überdies die am Server protokollierte Verarbeitungszeit berücksichtigt. Im Vergleich zur Übertragung von einzelnen Datensätzen mehrmals vs. mehreren Datensätzen auf einmal wirkt sich die Kompri-

mierung nicht so stark auf die Übertragungsdauer aus bzw. hatte im WLAN eine negative Auswirkung.

	WLAN	WLAN und Kabel	Mobilfunk
SOAP komprimiert	37.562	581.252	947.689
SOAP unkomprimiert	35.469	609.395	1.002.530
REST + XML komprimiert	47.366	386.330	565.151
REST + XML unkomprimiert	37.392	389.917	598.081
REST + JSON komprimiert	44.602	420.154	566.576
REST + JSON unkomprimiert	30.723	473.362	607.389

Tabelle 5.5: Übertragungsdauer in Millisekunden, komprimiert vs. unkomprimiert und Verarbeitungszeit am Server berücksichtigt

Abbildung 5.5 zeigt die ermittelten Werte für die Übertragungsdauer in Sekunden je Übertragungsweg und aufgeschlüsselt nach komprimiert oder unkomprimiert, wobei die am Server mitprotokollierte Verarbeitungszeit berücksichtigt ist.

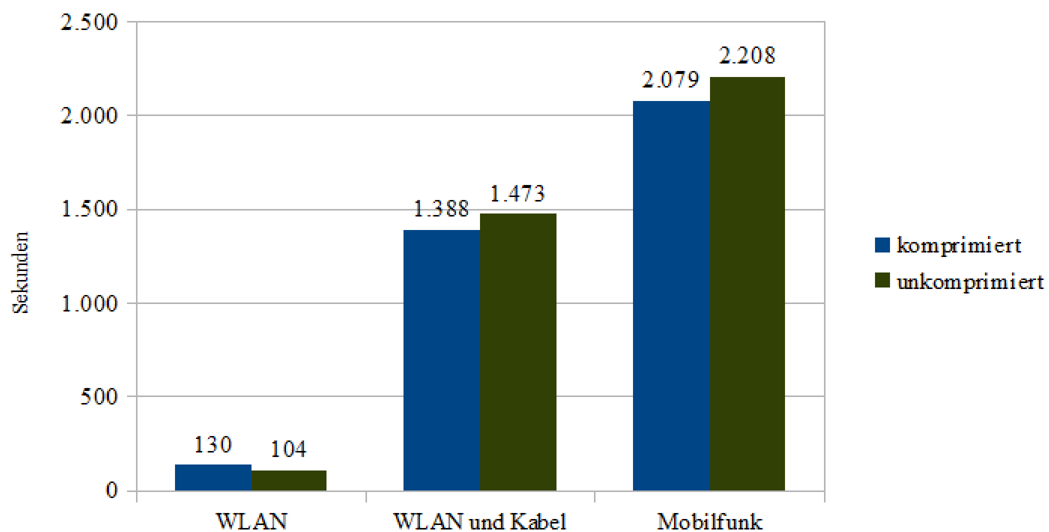

Abbildung 5.5: Übertragungsdauer in Sekunden, komprimiert vs. unkomprimiert und Verarbeitungszeit am Server berücksichtigt

	WLAN	WLAN und Kabel	Mobilfunk
SOAP komprimiert	53.808	634.493	1.005.144
SOAP unkomprimiert	51.327	658.547	1.064.540
REST + XML komprimiert	60.958	418.514	623.560
REST + XML unkomprimiert	46.613	428.351	672.184
REST + JSON komprimiert	58.595	521.153	619.350
REST + JSON unkomprimiert	43.092	573.663	675.857

Tabelle 5.6: Übertragungsdauer in Millisekunden, komprimiert vs. unkomprimiert, ohne Berücksichtigung der Verarbeitungszeit am Server

In Abbildung 5.6 werden die ermittelten Werte für die Übertragungsdauer in Sekunden je Übertragungsweg und aufgeschlüsselt nach komprimiert oder unkomprimiert gezeigt.

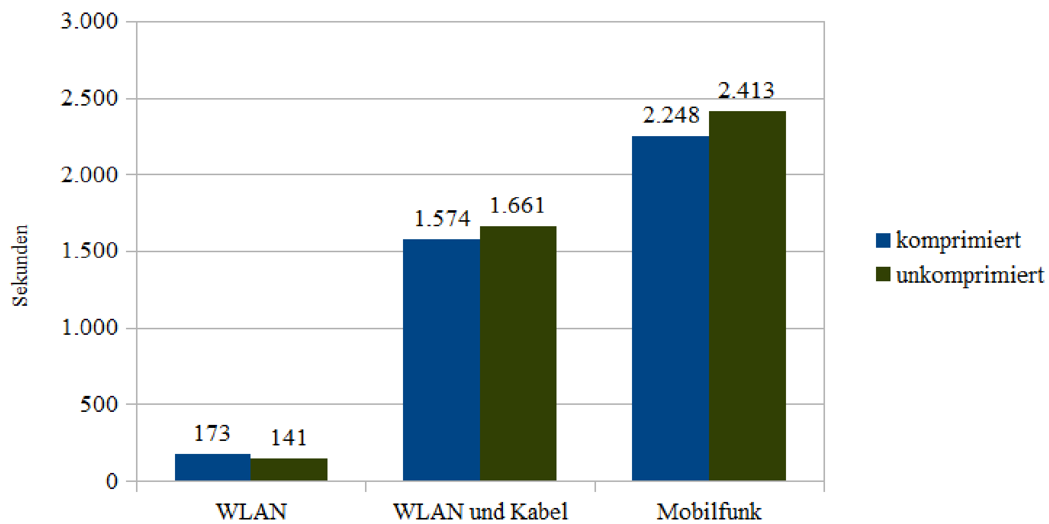

Abbildung 5.6: Übertragungsdauer in Sekunden, komprimiert vs. unkomprimiert, ohne Berücksichtigung der Verarbeitungszeit am Server

Tabelle 5.7 zeigt die ermittelten Werte für das gesendete und empfangene Datenvolumen in Byte je Übertragungsweg und Webservice und aufgeschlüsselt nach komprimiert oder unkomprimiert. Die Auswirkung der Komprimierung auf das übertragene Datenvolumen ist, im Gegensatz zur Auswirkung auf die Übertragungsdauer, erheblich.

	WLAN	**WLAN und Kabel**	**Mobilfunk**
SOAP komprimiert	40.500.410	40.464.151	40.463.564
SOAP unkomprimiert	53.210.855	53.342.331	53.354.089
REST + XML komprimiert	40.319.695	39.841.536	39.838.142
REST + XML unkomprimiert	52.490.390	52.629.702	52.627.417
REST + JSON komprimiert	40.083.005	39.664.406	39.648.477
REST + JSON unkomprimiert	52.159.645	52.291.482	52.286.568

Tabelle 5.7: Datenvolumen in Byte, komprimiert vs. unkomprimiert

Abbildung 5.7 zeigt die ermittelten Werte für das gesendete und empfangene Datenvolumen in KB je Übertragungsweg und aufgeschlüsselt nach komprimiert oder unkomprimiert.

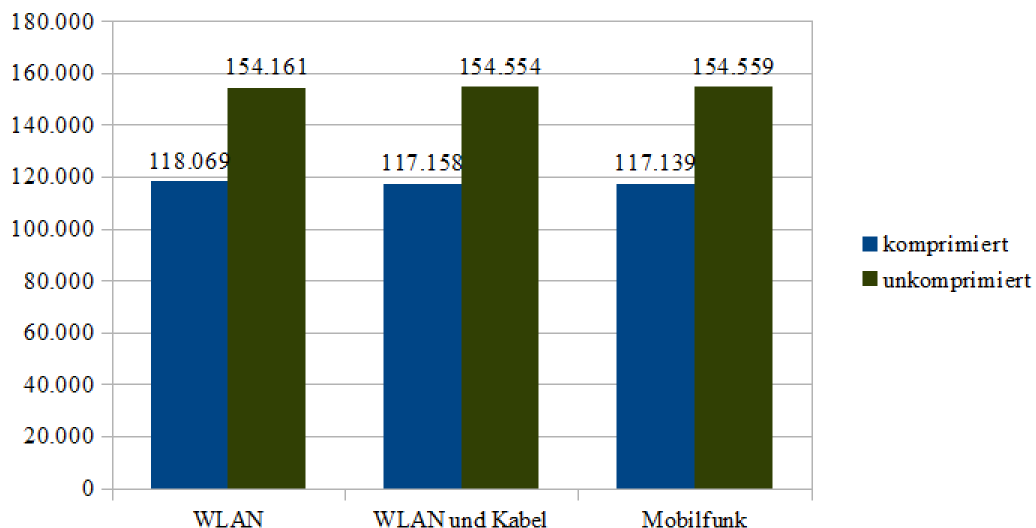

Abbildung 5.7: Datenvolumen in KB, komprimiert vs. unkomprimiert

Aufgrund der Länge der einzelnen Testdurchläufe wurde auch der Stromverbrauch ermittelt. Tabelle 5.8 zeigt die durchschnittlich vorhandene Restkapazität des Akkus in % am Ende der Testdurchläufe je Übertragungsweg und Webservice.

	WLAN	WLAN und Kabel	Mobilfunk
SOAP	98	94	86
REST + XML	97	94	87
REST + JSON	99	94	89

Tabelle 5.8: Durchschnittlich vorhandene Restkapazität des Akkus in %

Abbildung 5.8 zeigt die durchschnittlich vorhandene Restkapazität des Akkus in % am Ende der Testdurchläufe je Übertragungsweg. Trotz der in Kapitel 4.3.4 dargelegten Probleme lässt die große Differenz zwischen den einzelnen Werten doch eine entsprechende Schlussfolgerung zu.

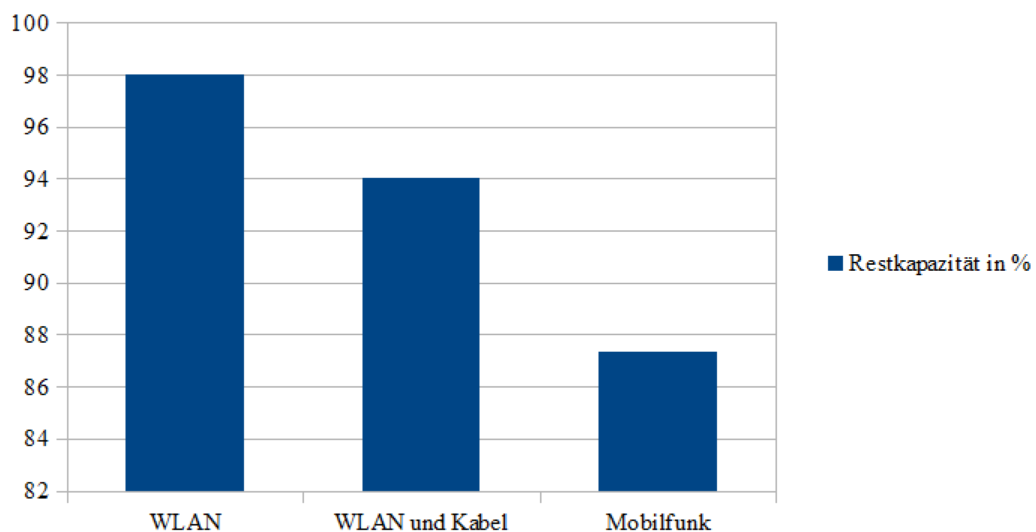

Abbildung 5.8: Durchschnittlich vorhandene Restkapazität des Akkus in %

Der Trend, der sich aus den in Abbildung 5.8 gezeigten Werten ableiten lässt, wird auch durch die Arbeit von Balasubramanian / Balasubramanian / Venkataramani bestätigt.

“In 3G, nearly 60% of the energy is tail energy, which is wasted in high-power states after the completion of a typical transfer. In comparison, the ramp energy spent in switching to this high-power state before the transfer is small. The tail and ramp energies can be amortized over frequent successive transfers, but only if the transfers occur within Tail time of each other” (Balasubramanian / Balasubramanian / Venkataramani, 2009, Seite 286).

“In WiFi, the association overhead is comparable to the tail energy of 3G, but the data transfer itself is significantly more efficient than 3G for all transfer sizes. When the scan cost is included, WiFi becomes inefficient for small sized transfers compared to GSM, but is still more energy efficient than 3G“ (ebd.).

“Our measurements ... confirm that the transmission energy consumed by WiFi is significantly smaller than both 3G and GSM, especially for large transfer sizes” (ebd., Seite 281).

Auch auf die Auswirkung der Art und Weise, wie die Daten übertragen werden, d. h. in diesem Fall ob einzelne Datensätze mehrmals oder mehrere Datensätze auf einmal übertragen werden, wird in der Arbeit von Balasubramanian et al. eingegangen.

“We find that the energy consumption is intimately related to the characteristics of the workload and not just the total transfer size, e.g., a few hundred bytes transferred intermittently on 3G can consume more energy than transferring a megabyte in one shot” (ebd., Seite 280).

5.3 Tests bzgl. SOAP vs. REST+XML vs. REST+JSON

Die nachfolgenden Tests dienten zum Feststellen des Unterschieds im Hinblick auf die Beanspruchung der CPU und des Speichers zwischen SOAP-basierten Webservices einerseits und RESTful Webservices mit XML bzw. JSON andererseits. Außerdem sollten die Unterschiede bzgl. der Ausführungszeit festgestellt werden.

Bei jedem einzelnen Testdurchlauf wurden 100 Datensätze entweder einzeln oder als Liste gesamt, unkomprimiert oder komprimiert, abgerufen und ausgewertet. Nach jedem einzelnen Testdurchlauf wurde die Anwendung geschlossen und mit einer anderen Anwendung fortgefahren. Diese Testdurchläufe wurden insgesamt fünfmal durchgeführt und der gesamte Ablauf sowohl für kleine als auch mittlere und große Datensätze durchgeführt.

Dementsprechend wurden insgesamt je Anwendung und Datensatzgröße 500 Datensätze in Listenform unkomprimiert, 500 Datensätze als einzelne Datensätze unkomprimiert, 500 Datensätze in Listenform komprimiert und 500 Datensätze als einzelne Datensätze komprimiert abgerufen und ausgewertet.

Es wurden wiederum drei Anwendungen entsprechend dem Kapitel 4.3 für das Abrufen und Verarbeiten der Datensätze verwendet. Für die Verbindung zum Server hin wurde WLAN genutzt.

Zum Bestimmen des Ausmaßes der CPU-Nutzung wurden die Werte für utime und stime in Clock Ticks ermittelt. Tabelle 5.9 und Abbildung 5.9 zeigen diese beiden Werte bereits zusammengezählt. Die Werte lassen auf keinen klaren Vorteil für die eine oder andere Technologie schließen. Es ergibt sich nur ein unerheblicher Vorteil von SOAP gegenüber REST und von REST+JSON gegenüber REST+XML.

	SOAP	REST+XML	REST+JSON
DS klein, Liste unkomprimiert	736	727	727
DS klein, Einzeln unkomprimiert	1.262	1.247	1.194
DS klein, Liste komprimiert	739	737	739
DS klein, Einzeln komprimiert	1.233	1.263	1.282
DS mittel, Liste unkomprimiert	852	860	874
DS mittel, Einzeln unkomprimiert	1.328	1.249	1.308
DS mittel, Liste komprimiert	890	976	999
DS mittel, Einzeln komprimiert	1.365	1.494	1.377
DS groß, Liste unkomprimiert	1.324	1.335	1.314
DS groß, Einzeln unkomprimiert	1.736	1.729	1.668
DS groß, Liste komprimiert	1.458	1.766	1.769
DS groß, Einzeln komprimiert	1.945	2.188	2.148

Tabelle 5.9: utime + stime in Clock Ticks, Liste vs. Einzeln, komprimiert vs. unkomprimiert

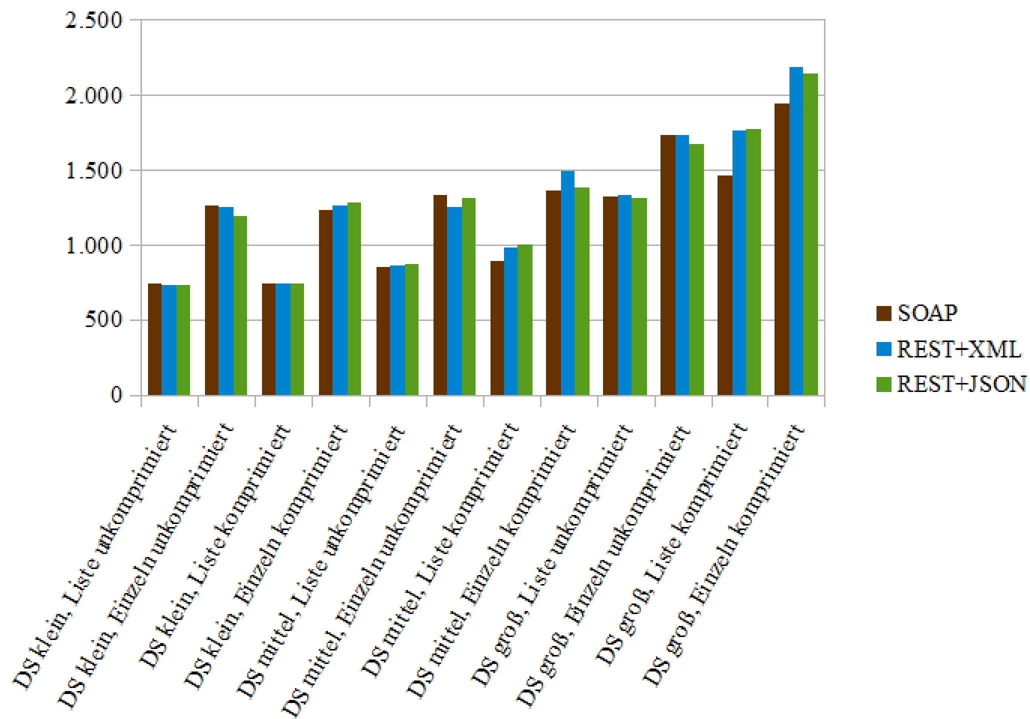

Abbildung 5.9: utime + stime in Clock Ticks, Liste vs. Einzeln, komprimiert vs. unkomprimiert

Für das Feststellen des Speicherbedarfs wurde einerseits der Durchschnittswert der Proportional Set Size während der Testdurchläufe und andererseits der Durchschnitt der jeweiligen Höchstwerte für die Resident Set Size und VmSize der einzelnen Testdurchläufe ermittelt. Tabelle 5.10 und Abbildung 5.10 zeigen die Werte für die Proportional Set Size, Tabelle 5.11 und Abbildung 5.11 die Werte für die Resident Set Size und Tabelle 5.12 und Abbildung 5.12 die Werte für VmSize jeweils in KB. Dabei zeigt sich ein leichter Vorteil von REST gegenüber SOAP und ein unerheblicher Vorteil von JSON gegenüber XML.

	SOAP	REST+XML	REST+JSON
DS klein, Liste unkomprimiert	10.678	10.016	9.974
DS klein, Einzeln unkomprimiert	12.100	11.189	11.115
DS klein, Liste komprimiert	10.588	9.912	9.923
DS klein, Einzeln komprimiert	12.152	10.990	10.941
DS mittel, Liste unkomprimiert	12.859	10.520	10.629
DS mittel, Einzeln unkomprimiert	14.010	10.917	11.091
DS mittel, Liste komprimiert	12.905	10.673	10.654
DS mittel, Einzeln komprimiert	13.931	11.019	11.034
DS groß, Liste unkomprimiert	20.489	11.062	10.957
DS groß, Einzeln unkomprimiert	14.407	11.262	11.284
DS groß, Liste komprimiert	20.609	11.065	10.986
DS groß, Einzeln komprimiert	14.205	11.318	11.251

Tabelle 5.10: Durchschnittswert der PSS in KB während der Testdurchläufe

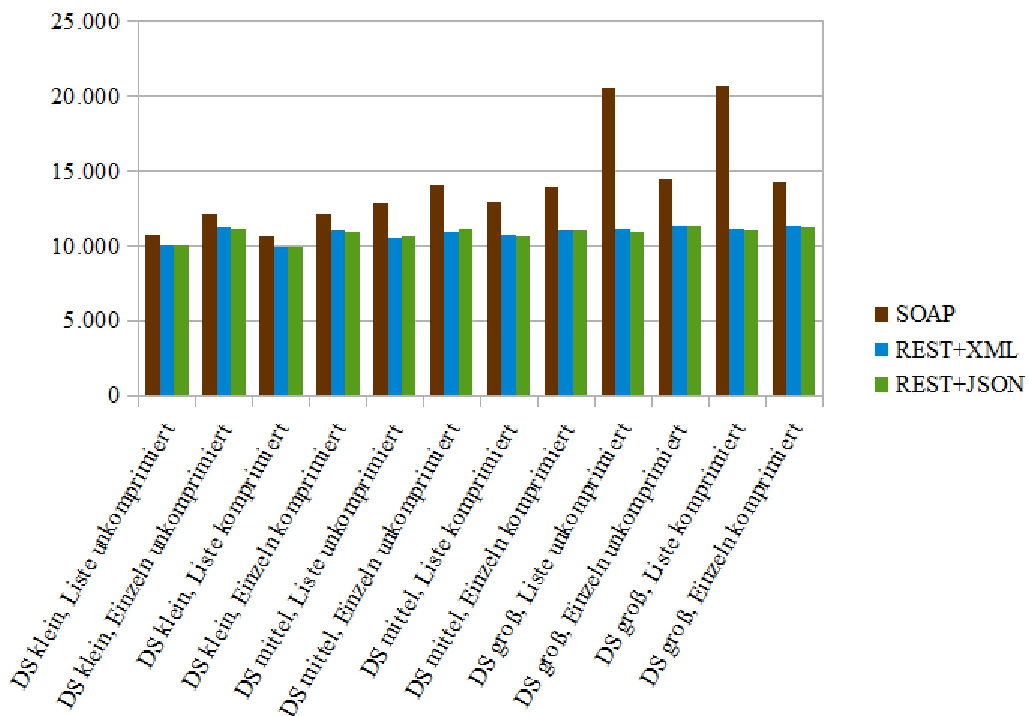

Abbildung 5.10: Durchschnittswert der PSS in KB während der Testdurchläufe

	SOAP	REST+XML	REST+JSON
DS klein, Liste unkomprimiert	51.585	51.124	51.185
DS klein, Einzeln unkomprimiert	53.540	52.556	52.553
DS klein, Liste komprimiert	51.518	50.966	51.058
DS klein, Einzeln komprimiert	53.876	52.560	52.288
DS mittel, Liste unkomprimiert	53.761	51.566	51.778
DS mittel, Einzeln unkomprimiert	63.720	51.946	52.372
DS mittel, Liste komprimiert	53.732	51.810	51.632
DS mittel, Einzeln komprimiert	62.480	52.090	52.029
DS groß, Liste unkomprimiert	61.528	51.946	51.814
DS groß, Einzeln unkomprimiert	62.224	52.368	52.183
DS groß, Liste komprimiert	61.830	51.990	51.884
DS groß, Einzeln komprimiert	62.620	52.310	52.138

Tabelle 5.11: Durchschnitt der jeweiligen Höchstwerte der RSS in KB

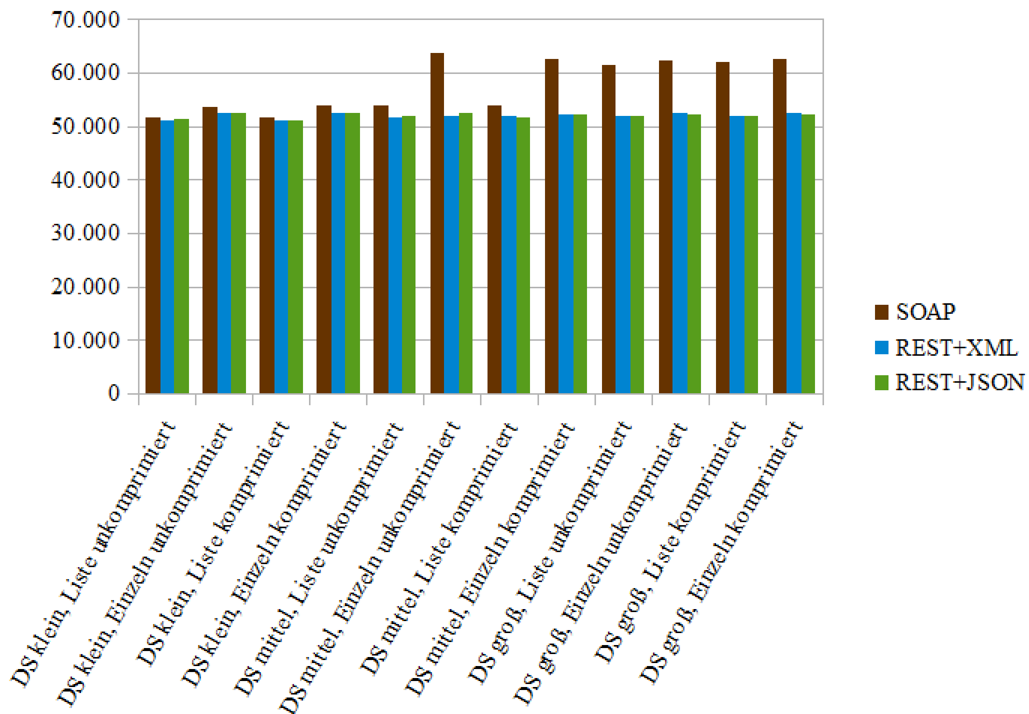

Abbildung 5.11: Durchschnitt der jeweiligen Höchstwerte der RSS in KB

	SOAP	REST+XML	REST+JSON
DS klein, Liste unkomprimiert	488.969	488.186	487.583
DS klein, Einzeln unkomprimiert	489.108	488.786	487.820
DS klein, Liste komprimiert	487.902	487.349	487.362
DS klein, Einzeln komprimiert	489.095	487.702	487.730
DS mittel, Liste unkomprimiert	487.683	487.311	487.331
DS mittel, Einzeln unkomprimiert	488.022	488.492	488.530
DS mittel, Liste komprimiert	487.678	487.325	488.159
DS mittel, Einzeln komprimiert	488.037	488.107	488.705
DS groß, Liste unkomprimiert	487.907	487.946	487.325
DS groß, Einzeln unkomprimiert	489.062	488.704	487.663
DS groß, Liste komprimiert	488.526	487.320	487.333
DS groß, Einzeln komprimiert	489.076	487.657	488.084

Tabelle 5.12: Durchschnitt der jeweiligen Höchstwerte der VmSize in KB

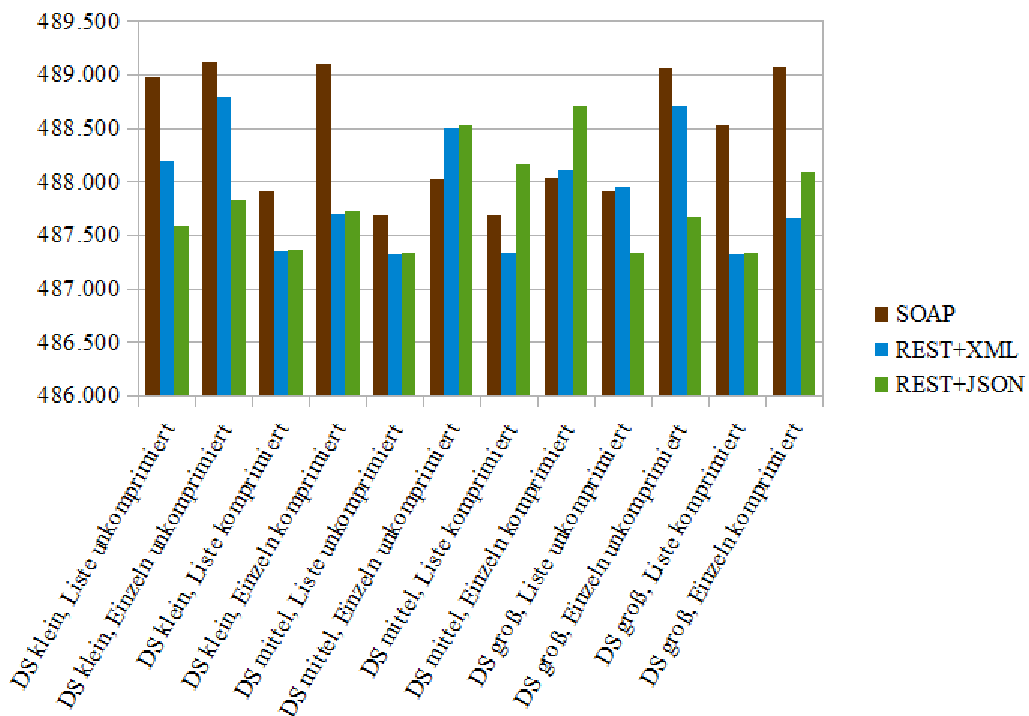

Abbildung 5.12: Durchschnitt der jeweiligen Höchstwerte der VmSize in KB

Abschließend zu den Tests noch die Darstellung der gesamten Ausführungsdauer, d. h. der Dauer vom Aufbau des Requests, Versenden des Requests, Empfangen des Response und Verarbeitung der empfangenen Daten. Tabelle 5.13 und Abbildung 5.13 zeigt diese Werte in Millisekunden. Diese Werte zeigen einen ähnlichen Verlauf, der auch bei utime + stime beobachtet werden konnte. Es ergibt sich auch hier nur ein unerheblicher Vorteil von SOAP gegenüber REST und von REST+JSON gegenüber REST+XML.

	SOAP	**REST+XML**	**REST+JSON**
DS klein, Liste unkomprimiert	7.958	7.875	7.889
DS klein, Einzeln unkomprimiert	18.239	18.925	17.988
DS klein, Liste komprimiert	7.898	7.913	8.093
DS klein, Einzeln komprimiert	17.724	18.455	18.247
DS mittel, Liste unkomprimiert	10.295	10.267	10.539
DS mittel, Einzeln unkomprimiert	20.360	19.848	20.248
DS mittel, Liste komprimiert	10.104	11.320	11.891
DS mittel, Einzeln komprimiert	20.261	22.870	21.068
DS groß, Liste unkomprimiert	17.540	18.245	18.625
DS groß, Einzeln unkomprimiert	28.165	28.717	27.490
DS groß, Liste komprimiert	18.538	22.565	22.780
DS groß, Einzeln komprimiert	31.430	33.920	32.976

Tabelle 5.13: Ausführungsdauer in Millisekunden

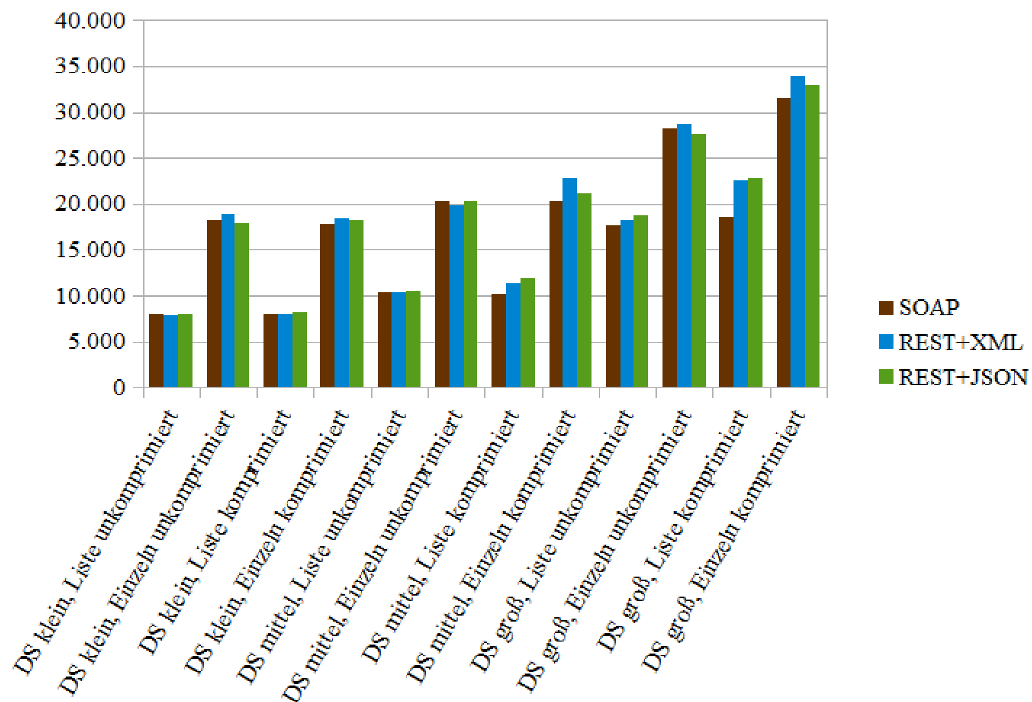

Abbildung 5.13: Ausführungsdauer in Millisekunden

5.4 Zusammenfassung zu den Tests

In diesem Kapitel wurden der Testaufbau und die Testabläufe erörtert, sowie die Testergebnisse präsentiert.

Es wurde gezeigt, dass bzgl. der zu übertragenden Datenmengen die Verwendung von REST gegenüber SOAP bzw. die Verwendung von JSON gegenüber XML aufgrund des schlankeren Formats vorteilhaft ist.

Bei der Übertragungsdauer wurde gezeigt, dass die Datenübertragung im Mobilfunknetz erheblich langsamer gegenüber einer Datenübertragung im WLAN ist. Besonders stark wirkt sich aus, ob die erforderliche Anzahl an Datensätzen einzeln oder in einem Vorgang als Liste übertragen werden. Komprimierung wirkt sich bzgl. der

Übertragungsdauer weniger aus, aber es kann damit das zu übertragende Datenvolumen wesentlich reduziert werden.

Bezüglich des Stromverbrauchs hatte die Datenübertragung im Mobilfunknetz ebenfalls negative Auswirkungen.

Hinsichtlich der Beanspruchung der CPU konnte kein klarer Vorteil für die eine oder andere Technologie festgestellt werden. Das Gleiche gilt auch hinsichtlich der Ausführungsdauer.

In Bezug auf den Speicherbedarf zeigte sich ein leichter Vorteil von REST gegenüber SOAP und ein unerheblicher Vorteil von JSON gegenüber XML.

6 Zusammenfassung

In dieser Arbeit wurde ein Vergleich zwischen SOAP-basierten und RESTful Webservices hinsichtlich des Ressourcenbedarfs, d. h. der CPU-Nutzung, des Speicherbedarfs und des Stromverbrauchs, am Client gezogen. Außerdem wurde auch auf die Ausführungsdauer, die von den unterschiedlichen Clients benötigt wurde, geachtet. Dabei beschränkt sich diese Arbeit auf Clients, die mobile Geräte mit dem Android-Betriebssystem sind.

Des Weiteren wurden bei den durchgeführten Tests unterschiedliche Übertragungswege, die typisch für mobile Geräte sind, in Betracht gezogen. Hinsichtlich der Datenübertragung wurden auch die Auswirkungen der Art und Weise wie die Daten downgeloadet werden, d. h. komprimiert oder nicht komprimiert, als einzelne Datensätze oder als eine Liste von Datensätzen, auf die Übertragungsdauer untersucht. Das unterschiedliche Datenvolumen bei den einzelnen Arten von Webservices bzw. bei den Datenformaten XML und JSON wurde ebenfalls untersucht. Das Feststellen des Stromverbrauchs erfolgte im Zuge dieser Tests.

Die Unterschiede bzgl. des Entwicklungsaufwands wurden ebenfalls beachtet.

Bei den durchgeführten Tests konnte kein nennenswerter Vorteil für die eine oder andere Technologie bzgl. der CPU-Nutzung und der Ausführungsdauer festgestellt werden.

In anderen Bereichen, wie dem Speicherbedarf oder der übertragenen Datenmengen stellte sich die Verwendung von JSON gegenüber XML bzw. REST gegenüber SOAP, wenn auch nur in geringem Maße, als vorteilhaft heraus.

Wesentliche Unterschiede ergaben sich aber aufgrund unterschiedlicher Übertragungswege. Das Mobilfunknetz war bei den durchgeführten Tests erheblich langsamer gegenüber den anderen Übertragungswegen.

Des Weiteren stellte sich die richtige Art und Weise des Datendownloads, d. h. möglichst alle benötigten Daten mit einer Anforderung abzurufen, als der entscheidende Faktor hinsichtlich der Übertragungsdauer heraus. Ein Komprimieren der übertragenen Daten, vor allem bei einer Übertragung mittels des Mobilfunknetzes, kann die Übertragungsdauer nochmals verkürzen und vor allem die übertragenen Datenmengen erheblich reduzieren.

Auch hinsichtlich des Stromverbrauchs stellte sich die Nutzung des Mobilfunknetzes für die Datenübertragung als negativ heraus.

Das Entwickeln der Client-Anwendungen hat gezeigt, dass grundsätzlich eine Nutzung von SOAP-basierten Webservices auch von Android-Geräten aus möglich ist. Der geringfügig höhere Implementierungsaufwand fällt in Anbetracht anderer Aufwände bei der Softwareentwicklung kaum ins Gewicht.

Alles in allem hat das Entwickeln der Client-Anwendungen und das Durchführen der Tests gezeigt, dass RESTful Webservices mit JSON für Neuentwicklungen zu favorisieren sind. Der Unterschied zu den anderen Technologien ist aber nicht so beachtlich, sodass ein Weiterverwenden von bereits bestehenden Services auch von Android-Clients aus durchaus in Erwägung zu ziehen ist. Um eine gute Performance der Client-Anwendungen zu erreichen, sollte vor allem auf die Art und Weise des Datendownloads geachtet werden, d. h. möglichst die Daten bei verfügbarer WLAN-Verbindung und möglichst alle benötigten Daten auf einmal downzuloaden und diese gegebenenfalls zu komprimieren.

7 Literaturverzeichnis

Android-Reference Context: Reference for android.content.Context class. Stand 13.09.2013.
URL: http://developer.android.com/reference/android/content/Context.html

Android-Reference Debug: Reference for android.os.Debug class. Stand 13.09.2013.
URL: http://developer.android.com/reference/android/os/Debug.html

Android-Reference Process: Reference for android.os.Process class.
Stand 13.09.2013.
URL: http://developer.android.com/reference/android/os/Process.html

Android-Reference SystemClock: Reference for android.os.SystemClock class.
Stand 13.09.2013.
URL: http://developer.android.com/reference/android/os/SystemClock.html

Android-Reference TrafficStats: Reference for android.net.TrafficStats class.
Stand 13.09.2013.
URL: http://developer.android.com/reference/android/net/TrafficStats.html

Balasubramanian, Niranjan / Balasubramanian, Aruna / Venkataramani, Arun: Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. ACM, New York, USA, 2009. Seiten 280 – 293.
URL: http://doi.acm.org/10.1145/1644893.1644927

Fielding, R. / Gettys, J. / Mogul, J. / Frystyk, H. / Masinter, L. / Leach, P. / Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. Juni, 1999.
Stand 15.09.2013. URL: http://tools.ietf.org/html/rfc2616

Jendrock, Eric / Cervera-Navarro, Ricardo / Evans, Ian / Gollapudi, Devika / Haase, Kim / Markito, William / Srivathsa, Chinmayee: The Java EE 7 Tutorial. August 2013.
URL: http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

Kerrisk, Michael: Linux Programmer's Manual. PROC(5). Stand 08.08.2013. URL: http://man7.org/linux/man-pages/man5/proc.5.html

Krintz, Chandra / Wen, Ye / Wolski, Rich: Application-level Prediction of Battery Dissipation. In: Proceedings of the 2004 international symposium on Low power electronics and design. ACM, New York, USA, 2004. Seiten 224 – 229. URL: http://doi.acm.org/10.1145/1013235.1013292

Pautasso, Cesare / Zimmermann, Olaf / Leymann, Frank: RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In: Proceedings of the 17th international conference on World Wide Web. ACM, New York, USA, 2008. Seiten 805 – 814.
URL: http://doi.acm.org/10.1145/1367497.1367606

Tilkov, Stefan: REST und HTTP. Einsatz der Architektur des Web für Integrationsszenarien. Heidelberg, Deutschland, 2011. ISBN: 978-3-89864-732-8

Anhang

A.1 Python-Skript zum Erstellen der Bilder

```
# Basierend auf:
# http://pseentertainmentcorp.com/smf/index.php?topic=2034.0
# Datum: 2013-07-13
# Zweck: Erstellen von Bildern im BMP-Format
#        mit zufälligen Pixelwerten
#
import struct, random

default_bmp_header = {'mn1':66,
                      'mn2':77,
                      'filesize':0,
                      'undef1':0,
                      'undef2':0,
                      'offset':54,
                      'headerlength':40,
                      'width':200,
                      'height':200,
                      'colorplanes':0,
                      'colordepth':24,
                      'compression':0,
                      'imagesize':0,
                      'res_hor':0,
                      'res_vert':0,
                      'palette':0,
                      'importantcolors':0}

def bmp_write(header, pixels, filename):
    header_str = bytes()
    header_str += struct.pack('<B', header['mn1'])
    header_str += struct.pack('<B', header['mn2'])
    header_str += struct.pack('<L', header['filesize'])
    header_str += struct.pack('<H', header['undef1'])
    header_str += struct.pack('<H', header['undef2'])
    header_str += struct.pack('<L', header['offset'])
    header_str += struct.pack('<L', header['headerlength'])
    header_str += struct.pack('<L', header['width'])
    header_str += struct.pack('<L', header['height'])
    header_str += struct.pack('<H', header['colorplanes'])
    header_str += struct.pack('<H', header['colordepth'])
```

```
    header_str += struct.pack('<L', header['compression'])
    header_str += struct.pack('<L', header['imagesize'])
    header_str += struct.pack('<L', header['res_hor'])
    header_str += struct.pack('<L', header['res_vert'])
    header_str += struct.pack('<L', header['palette'])
    header_str += struct.pack('<L', header['importantcolors'])
    outfile = open(filename, 'wb')
    outfile.write(header_str + pixels)
    outfile.close()

def row_padding(width, colordepth):
    byte_length = width*colordepth/8
    padding = int((4-byte_length)%4)
    padbytes = bytes()
    for i in range(padding):
        x = struct.pack('<B',0)
        padbytes += x
    return padbytes

def pack_color(red, green, blue):
    return struct.pack('<BBB',blue,green,red)

def createbmps():
    counter = 0;
    while counter < 1:
        header = default_bmp_header
        header["width"] = 150
        header["height"] = 50

        def rand_color():
            x = random.randint(0,255)
            return x

        pixels = bytes()
        for row in range(header['height']-1,-1,-1):
            for column in range(header['width']):
               r = rand_color()
               g = rand_color()
               b = rand_color()
               pixels += pack_color(r, g, b)
            pixels += row_padding(header['width'],
header['colordepth'])

        filename = 'image' + str(counter) + '.bmp'
        bmp_write(header, pixels, filename)
```

```
        counter += 1

if __name__ == '__main__':
    createbmps()
```

A.2 Python-Skript zum Erstellen der Hashwerte und Befüllen der Datenbank

```
# Author: Bernhard Eibegger
# Datum: 2013-07-15
# Zweck: Dateien Base64 codieren, SHA256-Hashwerte berechnen
#        und Daten in eine MySQL-Datenbank schreiben
#
import base64, binascii, hashlib, mysql.connector, time
from mysql.connector import errorcode

counter = 4001
filename = r'C:\MyTestData\images\image'

config = {
  'user': 'MyUser',
  'password': 'MyPassword',
  'host': '127.0.0.1',
  'database': 'mydb',
  'raise_on_warnings': True
}
try:
    conn = mysql.connector.connect(**config)
    cursor = conn.cursor()
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Wrong credentials!")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exists!")
  else:
    print(err)
while counter < 5001:
    filenamefull = filename + str(counter) + '.bmp'
    fin = open(filenamefull,'rb')
    imgfile = fin.read()
    fin.close()
    imgbase64 = binascii.b2a_base64(imgfile)
    imgbase64 = imgbase64.decode('utf-8')
```

```
    imgbase64 = imgbase64[:-1]
    imgbase64enc = imgbase64.encode('utf-8')
    hashimg = hashlib.sha256(imgfile).hexdigest()
    hashstring = hashlib.sha256(imgbase64enc).hexdigest()
    try:
        cursor.execute('INSERT INTO
article(articlename,hashvalue,hashvaluebase64,picturebase64) VALUES
("%s","%s","%s","%s")' % ('image' + str(counter) + '.bmp', hashimg,
hashstring, imgbase64))
        conn.commit()
    except mysql.connector.Error as err:
        print(err)
        conn.rollback()
    #
    counter += 1
# End while
#
conn.close()
```