Client-Server-Kommunikation mit Android

Studierender - Roman Würsch
Projektbetreuer - Beat Seeliger

HSZ-T - Technische Hochschule Zürich

Dezember 2010 bis März 2011

Zusammenfassung

In dieser Semesterarbeit wird untersucht, wie eine Kommunikation zwischen dem Android Betriebsystem und einem Serversystem funktioniert. Es wird anhand der plattformunabhängigen Kommunikationsprotokolle REST und Hessian ein Prototyp implementiert, der eine grundlegende Funktionalität bietet. Der Prototyp soll soweit ausgereift sein, damit man in der Praxis auf ihn aufbauen könnte. Zudem werden diese beiden Protokolle in einer Gegenüberstellung verglichen.

Inhaltsverzeichnis

1. Personalienblatt

Name, Vorname:	**Roman Würsch**
Adresse:	**Murhaldenweg 16**
PLZ, Wohnort:	**8057 Zürich**
Geburtsdatum:	**10.11.1980**
Heimatort:	**Emmetten NW**

Ich bestätige, dass die vorliegende Semesterarbeit "Client-Server-Kommunikation mit Android" in allen Teilen selbständig erarbeitet und durchgeführt wurde.

Ort und Datum Unterschrift

2. Aufgabenstellung

2.1. Ausgangslage

Die App-Stores der Mobil-Hersteller boomen. Die Art und Weise wie man Apps programmieren soll wird von den Mobil-Hersteller meist klar vorgegeben. Sobald jedoch eine App mit einem Server im Internet kommunizieren soll, gibt es keine Vorschriften zur Implementierung.

2.2. Ziel der Arbeit

Es soll eine Client-Server-Kommunikation am Beispiel des Android Betriebsystems gezeigt werden. Das Android Betriebsystem nimmt den Platz des Clients ein. Als Server wird ein Java EE Applikationsserver verwendet.

Folgende Ziele sollen erreicht werden:

- Es sollen zwei gängige plattformunabhängigen Arten von Daten-Kommunikation miteinander verglichen werden.
- Es solle eine detaillierte Anforderungsanalyse an einen Prototypen durchgeführt werden.
- Es soll ein Prototyp für Daten-Kommunikation implementiert werden. Der Prototyp soll beim Server Daten lesen, erstellen, aktualisieren und löschen können. Der Prototyp soll auf einem Android-Gerät lauffähig sein.

Folgende Punkte werden abgegrenzt, da es den Rahmen der Arbeit sprengen würde:

- Der Vergleich der Daten-Kommunikation soll auf zwei Protokolle beschränkt werden, die in der Praxis eingesetzt werden, Representational State Transfer (REST)[1] und Hessian[2].
- Der Prototyp des Clients wird für die Android Version 2.2 (Froyo[3]) entwickelt und soll mit dem Protokoll REST[1] und Hessian[2] implementiert werden.
- Es werden keine Umfragen, Erhebungen und Feldstudien durchgeführt.

[1] Der Begriff Representational State Transfer (mit dem Akronym REST) bezeichnet einen Softwarearchitekturstil für verteilte Hypermedia-Informationssysteme wie das World Wide Web

[2] Hessian ist ein binäres Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können.

[3] Die Android Entwickler geben den verschiedenen Versionen ihres Betriebsystems jeweils Namen von süssen Speisen FroYo ist ein Akronym für Frozen yogurt

2.3. Aufgabenstellung

- Gegenüberstellung der beiden Protokollkonzepte
- Gegenüberstellung der Einsatzgebiete der beiden Protokolle
- Prüfen ob eine Implementierung für Android möglich ist
- Testszenarien für die beiden Prototypen ausarbeiten
- Entwicklung eines Prototypen für beide Protokolle REST und Hessian
- Durchführen der definierten Tests der beiden Prototypen

2.4. Erwartete Resultate

Die erwarteten Resultate ergeben sich aus der Aufgabenstellung:

1. Es wird die Versionskontrolle auf der Basis von GIT bei Github.com verwendet.
2. Planung, Arbeitsnachweis und weitere Informationen werden im WIKI von Github.com geführt.
3. Technischer Bericht
   a) Beschreibung der Ausgangslage
   b) Konzeptionelle Gegenüberstellung von REST und Hessian
   c) ~~Gegenüberstellung der Einsatzgebiete von REST und Hessian~~[4]
   d) Definition der Anforderungen an den Prototyp
   e) Ergebnisse der Tests und Abdeckung der Testszenarios
   f) Zusammenfassung zur Entwicklung des Prototypen
4. Lauffähiger Prototyp:
   a) Es soll ein lauffähiger Prototyp auf der Basis von Android 2.2 (Froyo) für beide Protokolle gemacht werden.
   b) Der Prototyp soll bei einem Server Daten lesen, erstellen, aktualisieren und löschen können.

[4] Im Design-Review hat man beschlossen auf dieses Kapitel zu verzichten.

3. Erreichte Ziele

Es wurden alle Ziele gemäss den erwarteten Resultaten der Aufgabenstellung erreicht. Die einzelnen Punkte sind hier analog der Aufgabenstellung aufgeführt:

1. **Erreicht** Die Versionskontrolle liegt öffentlich zugänglich unter der Adresse:
   `https://github.com/sushicutta`

2. **Erreicht** Das Wiki ist öffentlich zugänglich unter der Adresse:
   `https://github.com/sushicutta/ClientServerKommunikationMitAndroid/wiki`

3. **Erreicht** Technischer Bericht
   a) **Erreicht** Beschreibung der Ausgangslage
   b) **Erreicht** Konzeptionelle Gegenüberstellung von REST und Hessian
   c) **Erreicht** Dieses Ziel wurde beim Design-Review gestrichen.
   d) **Erreicht** Definition der Anforderungen an den Prototyp
   e) **Erreicht** Ergebnisse der Tests und Abdeckung der Testszenarios
   f) **Erreicht** Zusammenfassung zur Entwicklung des Prototypen

4. **Erreicht** Lauffähiger Prototyp
   a) **Erreicht** Der Sourcecode für den lauffähigen Prototypen des Clients ist öffentlich zugänglich unter folgender Adresse:
   `https://github.com/sushicutta/ClientServerKommunikationMitAndroid`
   Der gültige Branch heisst: **restWithGlassFish**
   b) **Erreicht** Der Sourcecode für den lauffähigen Prototypen des Servers ist öffentlich zugänglich unter folgender Adresse:
   `https://github.com/sushicutta/Semesterarbeit`
   Der gültige Branch heisst: **jpa**

4. Beschreibung der Ausgangslage

Für das Informatik Diplomstudium an der Fachhochschule Zürich für Technik HSZ-T wird von den Studenten verlangt eine Semesterarbeit eigenständig zu verfassen.

4.1. Wahl des Themas

Seit dem Siegeszug der Smartphones, welcher durch das iPhone von Apple eingeläutet wurde interessiere ich mich für mobile Geräte. Durch das Interesse an Linux und an freien und offenen Systemen bin ich bei Android als Betriebsystem gelandet. Google versucht mit Android das zu erreichen, was Apple mit dem iOS erreicht hat. Sie wollen Entwicklern eine einfach Schnittstelle für die Entwicklung neuer Software bieten. Zudem soll die Software über einen von Google gesteuerten Marktplatz zur Verfügung gestellt werden.

Die Entwicklung neuer Software für Android wird von Google mittels Tutorials gefördert. Dabei habe ich persönlich keinen Hinweis auf die Datenkommunikation zwischen Android und Fremdsystemen bekommen. Da eine solide Datenkommunikation heutzutage zu den Grundbedürfnissen in der Softwareentwicklung gehört, wollte ich die Möglichkeit zweier systemunabhängigen Methoden näher betrachten.

Bei der Auswahl der beiden Protokolle bin ich schnell auf REST gestossen, da es sich in den letzten paar Jahren zu einer populären Methode entwickelt hatte. Von Hessian als Kommunikationsprotokoll hatte ich nur schon gehört, ich wusste aber, dass es für die meisten gängigen Programmiersprachen eine Implementierung gibt, somit ist es in den Kreis meiner Auswahl gefallen.

4.2. Bewertungskriterien

Wie im Kick-off Meeting festgelegt, werden die Bewertungskriterien aus dem Bachelor Studiengang verwendet.

4.3. Sprache

Der Bericht wird in deutscher Sprache verfasst. Englische Ausdrücke werden im Context verwendet, wenn man davon ausgehen kann, dass es gängige Ausdrücke aus dem Gebiet der Informatik sind.

4.4. Projekt Termine

Die Projekt Termine wurden alle eingehalten, siehe Tabelle 4.1.

Termin	Datum	Ort
14. 12. 2010	Kick-off Meeting	Panter GmbH
12. 01. 2011	Design-Review Meeting	HSZ-T
16. 02. 2011	Abgabe der Dokumentation	HSZ-T
02. 03. 2011	Schlusspräsentation	HSZ-T

Tabelle 4.1.: Projekt Termine

4.5. Projekt Historie

In der Projekt Historie sind die wichtigsten Meilensteine ersichtlich, siehe Tabelle 4.2.

Datum	Status	Wer
06. 12. 2010	Ein Dozierender hat die Arbeit inkl. Aufgabenstellung ausgeschrieben und wartet auf einen Studierenden, der diese Arbeit durchführt	Roman Würsch
07. 12. 2010	Die Arbeit ist freigegeben	Olaf Stern
14. 12. 2010	Freigabe Kick-Off	Beat Seeliger
12. 01. 2011	Freigabe Desgin-Review	Beat Seeliger
16. 02. 2011	Abgabe der Dokumentation	Roman Würsch
02. 03. 2011	Präsentation der Arbeit	Roman Würsch

Tabelle 4.2.: Projekt Historie

4.6. Richtlinien

Folgende Dokumente mit Richtlinien der Hochschule für Technik Zürich wurden für die Semesterarbeit berücksichtigt:

- Reglement [DOS09c]
- Ablauf [DOS09a]
- Bewertungskriterien [DOS09b]

5. Konzeptionelle Gegenüberstellung von REST und Hessian

5.1. REST – Representational State Transfer

REST stammt aus der Dissertation von Roy Fielding[Fie00] in der er den Erfolg des World Wide Web (WWW) auf bestimmte Eigenschaften der verwendeten Mechanismen und Protokolle (z. B. Hypertext Transfer Protocol (HTTP)) zurückführt.

5.1.1. Was ist REST

Der Client ändert den Status mit jeder Ressource Repräsentation.

1. Ein Client referenziert eine Web Ressource über eine Uniform Resource Locator (URL).
2. Der Server lieferte eine Repräsentation der Ressource in einer vom Client akzeptierten Form, z.B. als ein Hypertext Markup Language (HTML) Dokument oder als Extensible Markup Language (XML).
3. Die vom Server geladene Ressource versetzt den Client in einen neuen Status und bietet evtl. Links zu neuen Ressourcen an.
4. Der Client lädt eine ihm neue bekannte Ressource über die URL, was wieder in einem Ressource Access endet.
5. Die vom Server geladene Ressource versetzt den Client wieder in einen neuen Status.

5.1.2. Wie funktioniert REST

- REST verwendet URL um Ressourcen zu adressieren
- REST verwendet (a) HTTP-PUT, (b) HTTP-GET, (c) HTTP-POST und (d) HTTP-DELETE Requests um Daten zu (a) kreieren, (b) lesen, (c) aktualisieren und (d) zu löschen.
- REST verwendet Standard Ressource Repräsentationen wie HTML, XML, JavaScript Object Notation (JSON), usw.
- REST verwendet HTTP Header “Content-type” wie text/html, application/xml, application/json, usw. um die Codierung der Ressourcen zu deklarieren.
- REST liefert die codierten Ressourcen aufgrund des HTTP Header “Accept”.
- REST liefert bei einem Request HTTP Statuscodes. 2xx bei einem erfolgreichen Request oder 4xx wenn der Request fehlgeschlagen ist. Es können auch weitere Statuscodes, wie 3xx oder 5xx, zurückgegeben werden, je nach Definition.

5.1.3. Wieso REST

Das WWW hat sich in den letzten Jahren bewährt. Das WWW ist skalierbar und einfach zu verstehen. Wenn die Kommunikation von Maschine zu Mensch funktioniert, dann ist sie sicher gut genug um auch von Maschine zu Maschine zu funktionieren.

Fast jede Plattform unterstützt die Standards des WWW, damit macht es eine Plattform übergreifende Kommunikation möglich.

Im WWW gibt es Standards für Security (z.B. Hypertext Transfer Protocol Secure (HTTPS)) und Caches (z.B. HTTP ETag), welche mit REST verwendet werden können.

5.2. Hessian

Hessian ist ein leichtgewichtiges Kommunikationsprotokoll, das von Caucho Technology, Inc. entwickelt wurde. Es existieren zwei Protokolldefinitionen Hessian 1.0.x und Hessian 2.0. Ich habe mich in dieser Arbeit nur mit Hessian 1.0.x beschäftigt. Hessian 2.0 bietet zum RPC-Teil (synchron) auch eine Nachrichtenbasierte Kommunikation (asynchron) an.

5.2.1. Was ist Hessian

Hessian ist ein binäres Netzwerkprotokoll, dass für Remote Procedure Call (RPC) und Datenkommunikation zwischen Computersystemen verwendet werden kann. Hessian wird üblicherweise über HTTP übertragen. Hessian definiert keine Interface Definition Language (IDL) oder ein externes Schema wie man das aus Simple Object Access Protocol (SOAP) oder Common Object Request Broker Architecture (CORBA) kennt. Die Schnittstelle (sprich Methoden Interfaces) des Servers muss beim Client bekannt sein.

5.2.2. Was beinhaltet Hessian

Hessian definiert hauptsächlich zwei Dinge.

1. Wie ein Remote Procedur Call ausgeführt wird. Dabei wird definiert, wie eine Methode auf einem Objekt aufgerufen wird und wie die dazugehörigen Argumente übergeben werden. Zudem wird auch definiert, wie der Reply der Methode aussieht. Zudem werden im Fehlerfall sogenannte Faults zurückgegeben, was nichts anderes ist als eine Exception.

2. Die Serialisierung von Daten. Es werden neun primitive, zwei kombinierte und spezielle Konstrukte unterstützt. Bei den primitiven Datentypen handelt es sich um folgende:
   - boolean
   - 32-bit int
   - 64-bit long
   - 64-bit double
   - 64-bit date
   - UTF8-encoded string
   - UTF8-encoded xml
   - raw binary data
   - remote objects

Kombinierte Konstrukte sind:

- list für Listen und Arrays
- map für Objects und Hashtabellen

Die speziellen Konstrukte sind:

- null für null values
- ref Referenzen auf ein object in einer list oder map

5.2.3. Wieso Hessian

Als binäres Protokoll ist Hessian insbesondere für die Versendung von Binärdaten geeignet. Binäre Protokolle wie RMI und Hessian sind darüber hinaus wesentlich performanter als XML basierte Protokolle, wie z.B. SOAP oder XML-RPC.

Hessian wurde von den Erfindern für verschiedene Programmiersprachen portiert, damit macht es eine Plattform übergreifende Kommunikation möglich.

5.3. Gegenüberstellung

In der Tabelle 5.1 werden die wichtigsten Eigenschaften verglichen.

Aspekt	REST	Hessian
Standard	Kein Standard, verwendet existierende Standards wie RFC2616 HTTP 1.1	Kein Standard, wurde von Caucho Technology, Inc entwickelt
Ressource Adressierung	Jede Ressource hat ihre eigene URL	Indirekt über Hessian Funktionen
URL	Wird verwendet für die Adressierung der einzelnen Ressourcen	Wird für den Hessian Endpoint (Servlet) verwendet
Error handling	Server Statuscodes 4xx und 5xx im Response	Hessian Fault
Daten Repräsentation	Alle Encodings von HTTP definiert, wie (Text, HTML, XML, JSON, usw.)	Serialisierung gemäss Hessian Protokoll
HTTP	PUT, GET, POST, DELETE sind auf Funktionen gemappt	POST wird für den Transport verwendet
State	Stateless, serverseitig wird nichts gespeichert	Stateful, jede Hessian Funktion ist ein Teil einer definierten Applikation
Interface	HTTP PUT, GET, POST, DELETE	Keine IDL wie bei CORBA, das Interface muss aber bekannt sein.
Transaktionen	Nicht direkt unterstützt, eine Transaktion könnte über Ressourcen abgebildet werden.	Transaktionskontext kann im Header übergeben werden
Asynchrone Kommunikation	Nicht direkt unterstützt.	Hessian 2.0 bietet diese Möglichkeit

Tabelle 5.1.: Gegenüberstellung von REST und Hessian

5.4. Zusammenfassung

Da ich mich nun tiefer mit beiden Protokollen beschäftigt habe, möchte ich noch meinen subjektiven Standpunkt vertreten.

5.4.1. REST ist "cool", aber der Teufel steckt im Detail

REST ist in aller Munde. Seit gut drei Jahren höre ich immer wieder, wie einfach dass eine Schnittstelle mit REST zu implementieren ist. Das mag auch sein, wenn man sich an einen Standard wie JSR-000311[Mic08] hält, und den Server, wie auch den Client, nach diesem Schema umsetzt. Leider aber sieht es schon anders aus, wenn der Server in Ruby on Rails[Wik11c] geschrieben wurde und der Client in Java implementiert wird. Diese Erfahrung habe ich gemacht, als ich das im Rahmen der Semesterarbeit versucht habe. Die Unterschiede zeigen sich in den Finessen, wie zum Beispiel ein HTTP-Responsecode aussieht, wenn ich auf die URL einer ehemals publizierten Ressource, welche mit Ruby on Rails bereitgestellt wurde, zugreife, die aber schon einmal gelöscht wurde. Da kann es durchaus vorkommen, dass dann nicht ein Responsecode 404 (Not Found), wie erwartet, sondern ein Responsecode 406 (Not Acceptable) zurückgegeben wird. Zudem kommen optionale Headerfelder, die in einem HTTP-Request oder HTTP-Response Objekt verpackt werden können, und so weiter. Wie man hier sieht, gibt es viele verschiedene Arten, wie man ein RESTful Webservice auslegen kann.

Wenn man demnach versucht eine plattformunabhängige Kommunikationsschnittstelle zur Verfügung zu stellen, oder eine solche anzubinden, muss man alle möglichen Fälle von Ressourcezugriffen untersuchen, und auch deren Fehlerfälle alle korrekt abarbeiten, nur so kann man gewährleisten, dass man eine "coole" Schnittstelle gebaut hat.

Natürlich gibt es auch die Vorteile, welche nicht vernachlässigt werden dürfen. Da sehe ich die weit aus Grössten in der Repräsentation der Daten und der Verständlichkeit der einzelnen Zugriffe über einzelne URL's. Hinzu kommt, dass namhafte Firmen wie Amazon oder Yahoo bereits Schnittstellen zu ihren Systemen über REST bereitstellen, was den Akzeptanzfaktor in einem möglichen Projekt enorm erhöht.

Wie die Schnittstelle eines möglichen Produktservices aussehen würde, möchte ich anhand einer Grafik 5.1 zeigen:

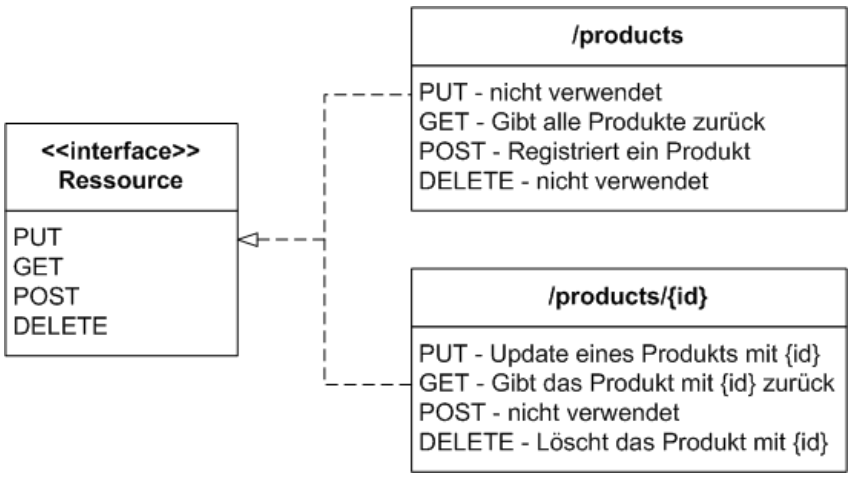

Abbildung 5.1.: Ein Produktservice mit REST bereitgestellt

5.4.2. Hessian ist leichtgewichtig, aber leider kein Standard

Ganz nach dem KISS-Prinzip[Wik11b] wurde dieses Protokoll geschaffen. Es soll so einfach wie möglich anzuwenden sein, und das ist es auch. Für den Serverteil, reicht ein Interface, und die ausprogrammierte Klasse ist auch schon in ein paar Zeilen Code geschrieben. Dann noch die nötige Konfiguration für den Servlet-Container. Die ganze Magie, der Method-Invocation und Serialisierung der Daten, wird über das Package com.caucho.hessian.* abgewickelt, davon sieht man als Softwareentwickler nichts mehr. Der Client ist wird wie der Server auch gleich kurzerhand realisiert. Die aktuelle Version 4.0.7 ist als JAR File gerade einmal 383 Kilobyte gross und beinhaltet alle Klassen um als Server oder als Client zu fungieren. Ein Beispiel Code in Java könnte in etwa so aussehen:

Das Interface:

```
package example;

public interface Basic {

  public String hello();

}
```

Die ausprogrammierte Klasse:

```
package example;

import com.caucho.hessian.server.HessianServlet;

public class BasicServiceEndpoint extends HessianServlet implements Basic {

  @Override
  public String hello() {
    return "Hello World";
  }

}
```

Dann fehlt nur noch die Konfiguration für den Servlet-Container im web.xml, und schon kann über die URL http://www.examplehost.com/BasicService auf den Server zugegriffen werden.:

```
<web-app>
  <servlet>
    <servlet-name>BasicService</servlet-name>
    <servlet-class>example.BasicServiceEndpoint</servlet-class>
    </servlet>
  <servlet-mapping>
      <servlet-name>BasicService</servlet-name>
      <url-pattern>/BasicService</url-pattern>
  </servlet-mapping>
</web-app>
```

Der Client kommt auch schön schlank daher:

```
package example;

import com.caucho.hessian.client.HessianProxyFactory;

public class BasicClient {

  public static void main(String []args) throws Exception {

    String url = "http://www.examplehost.com/BasicService";

    HessianProxyFactory factory = new HessianProxyFactory();
    Basic basic = (Basic) factory.create(Basic.class, url);

    System.out.println(basic.hello());

  }

}
```

Als Nachteil sehe ich den geringen Bekanntheitsgrad von Hessian und zudem, dass Hessian kein Standard ist. In einem grösseren Entwicklungsprojekt, dürfte ein Webservice wohl mit SOAP oder einer ähnlichen Technik umgesetzt werden, da es sich dabei effektiv um einen Standard handelt. Zudem dürfte es komplizierter werden, wenn man Datentypen verwendet, die im Protokoll nicht für die Serialisierung vorgesehen sind.

5.5. Fazit

Für kleine Projekte, oder Projekte bei denen die Grösse des ausgelieferten Pakets eine Rolle spielen, würde ich auf Hessian setzen. Durch die Leichtgewichtigkeit und die Einfachheit der Implementierung hat man mit Hessian ein mächtiges Werkzeug.

Bei einem Projekt, wo man nicht nur eine Kommunikation von Maschine zu Maschine bauen soll, sondern auch eine Schnittstelle für Menschen bereitstellt, würde ich auf REST setzten. Da man die Repräsentation gleich für maschinentaugliche, wie auch für browsertaugliche (was ja von Menschen verwendet wird) Formate bereitstellen kann. Sprich, die Ressourcen in XML und gleich in HTML zu präsentieren, macht da keinen grossen Unterschied mehr. Zudem fällt eine Dokumentation der Schnittstelle fast komplett weg, oder wenn wirklich nötig, kann die Dokumentation der Schnittstelle gleich als eigene Ressource über eine URL zugänglich gemacht werden.

6. Definition der Anforderungen an den Prototyp

Die Definition der Anforderungen an die Prototypen werden mit der Technik der User Stories[Wel99b] gemacht.

6.1. Was sind User Stories

User Stories beschreiben Anforderungen an eine Software in einer für Jedermann verständlichen Sprache. Es sollen keine technischen Details genannt, sondern viel eher Eigenschaften erläutert werden die jeder versteht. Oft werden User Stories als Aktionen in einem Graphical User Interface (GUI) verfasst. Aus den User Stories kann man die jeweiligen Akzeptanztests[Wel99a] ableiten.

Eine User Story sollte nicht mehr als drei Sätze haben. Das kommt daher, dass eine User Story nie zu komplex sein darf. Wenn man mehr als drei Sätze für die Beschreibung einer User Story braucht, sollte diese in mehrere kleinere User Stories aufgeteilt werden, welche genug simpel sind, um wiederum in drei Sätzen beschrieben zu werden.

Jeder User Story wird mit einer eindeutigen Nummer versehen: US-{User Story Nummer}

6.2. Anforderungen an den Prototyp

Im folgenden werden die Anforderungen an den Prototypen spezifiziert. Die Anforderungen werden aus zwei Blickwinkeln definiert. Aus der Sicht eines Anwenders und aus der Sicht eines Informatikstudenten. Diese beiden Sichtweisen werde jeweils ich vertreten, da ich mich davon abgegrenzt habe, irgendwelche Erhebungen durchzuführen.

Nachfolgend ist die Rede von Datenobjekten. Bei einem Datenobjekt wie es hier aufgeführt wird, handelt es sich um ein Plain Old Java Object (POJO).

6.3. Aus der Sicht des Anwenders

Der Anwender stellt ein Mensch dar, der nur die Sicht auf die Clientapplikation hat. Normalerweise hat ein Anwender keinerlei technische Hintergründe. Die definierten User Stories aus der Sicht des Anwenders sind in der Tabelle 6.1 ersichtlich.

User Story	Beschreibung
US-1	Ich als Anwender will ein hochwertiges GUI.
US-2	Ich als Anwender will meine Applikation überall verwenden können.
US-3	Ich als Anwender will eine stabile Applikation.
US-4	Ich als Anwender will ein ausgewähltes Datenobjekt vom Server an mein Android Gerät übertragen und darstellen können.
US-5	Ich als Anwender will eine komplette Liste von Datenobjekten vom Server an mein Android Gerät übertragen und darstellen können.
US-6	Ich als Anwender will ein ausgewähltes Datenobjekt auf dem Server löschen können.
US-7	Ich als Anwender will ein ausgewähltes Datenobjekt auf dem Server editieren können.
US-8	Ich als Anwender will ein neues Datenobjekt an den Server übertragen können. Dieses Datenobjekt soll für zukünftige Zugriffe auf dem Server gespeichert werden.
US-9	Ich als Anwender will über Übertragungsfehler informiert werden.

Tabelle 6.1.: User Stories eines Anwenders

6.4. Aus der Sicht des Informatikstudenten

Der Informatikstudent stellt ein Mensch dar, der sowohl die Sicht auf den Client, wie auch auf den Server der Applikation hat. Der Informatikstudent versucht durch technische Hilfsmittel die Anforderungen eines Anwenders zu erfüllen.

6.4.1. Clientapplikation

Die definierten User Stories zum Client aus der Sicht des Informatikstudent sind in der Tabelle 6.2 ersichtlich.

User Story	Beschreibung
US-10	Ich als Informatikstudenten will, dass die Clientapplikation für Geräte mit der Android Version 2.2 lauffähig ist.
US-11	Ich als Informatikstudenten will, dass die Clientapplikation unabhängig von der Serverapplikation ausgeliefert werden kann.

Tabelle 6.2.: User Stories eines Informatikstudenten zum Client

6.4.2. Serverapplikation

Die definierten User Stories zum Server aus der Sicht des Informatikstudent sind in der Tabelle 6.3 ersichtlich.

User Story	Beschreibung
US-12	Ich als Informatikstudent will, dass die Serverapplikation auf einem Java EE 6 fähigen GlassFish Applikationsserver läuft.
US-13	Ich als Informatikstudent will, dass die Serverapplikation sowohl Requests über das REST Protokoll wie auch über das Hessian Protokoll bearbeiten kann.
US-14	Ich als Informatikstudent will, dass Datenobjekte auf der Serverapplikation gespeichert werden können. Diese Daten sollen nur solange gespeichert sein, wie der Server läuft.
US-15	Ich als Informatikstudent will, dass mehrere Clientapplikationen gleichzeitig mit der Serverapplikation bedient werden können.
US-16	Ich als Informatikstudent will, dass die Serverapplikation auf dem Internet zugänglich ist.
US-17	Ich als Informatikstudent will, dass die Serverapplikation immer verfügbar ist.
US-18	Ich als Informatikstudent will, dass die Serverapplikation unabhängig von der Clientapplikation ausgeliefert werden kann.

Tabelle 6.3.: User Stories eines Informatikstudenten zum Server

6.5. Priorisierung aller User Stories

Da es sich bei dieser Arbeit und eine Semesterarbeit handelt, kann nicht auf jede Anforderung eingegangen werden. Die User Stories werden aus meiner Sicht priorisiert. Für jede Story, die in dieser Semesterarbeit nicht umgesetzt wird, werde ich eine kurze Begründung dazu abgeben.

6.5.1. Aus der Sicht des Anwenders

Die Priorisierung der User Stories aus der Sicht des Anwenders sind in der Tabelle 6.4 ersichtlich.

User Story	Umsetzung	Priorisierung
US-1	Nein	—
US-2	Ja	mittel
US-3	Ja	tief
US-4	Ja	hoch
US-5	Ja	hoch
US-6	Ja	hoch
US-7	Ja	hoch
US-8	Ja	hoch
US-9	Ja	tief

Tabelle 6.4.: Priorisierung der Anwender User Stories

6.5.2. Aus der Sicht des Informatikstudenten

Die Priorisierung der User Stories aus der Sicht des Informatikstudenten sind in der Tabelle 6.5 ersichtlich.

User Story	Umsetzung	Priorisierung
US-10	Ja	mittel
US-11	Ja	tief
US-12	Ja	mittel
US-13	Ja	hoch
US-14	Ja	hoch
US-15	Ja	tief
US-16	Nein	—
US-17	Nein	—
US-18	Ja	tief

Tabelle 6.5.: Priorisierung der Informatikstudent User Stories

6.6. Stories die nicht umgesetzt werden

- User Story US-1: Es wurde auf die Umsetzung eines hochwertigen GUI's für den Android Client verzichtet, da es den Aufwand einer Semesterarbeit übersteigt.
- User Story US-16: Der Server soll nicht über das Internet zugänglich sein, da eine lauffähige Instanz eines GlassFish Servers, der von überall her im Internet zugänglich ist, Geld kosten würde. Ich bin nicht bereit für eine Semesterarbeit einen solchen Hosting-Vertrag abzuschliessen.
- User Story US-17: Der Server kann nicht immer verfügbar sein, da die Instanz des GlassFish Servers auf meinen Notebook laufen wird.

6.7. Planung der User Stories die umgesetzt werden

Es werden alle User Stories mit der Priorität hoch und mittel umgesetzt. Die Reihenfolge der Umsetzung wird von mir festgelegt. Ich versuche die Reihenfolge so festzulegen, damit es ein Sinnvoller Ablauf in der Entwicklung der einzelnen User Stories gibt.

Falls die Zeit reicht, werden auch noch die tief priorisierten Users Stories umgesetzt. Es wird keine Reihenfolge für tief priorisierte User Stories festgelegt. Die Reihenfolge der Umsetzung darf somit frei gewählt werden.

Es wird folgende Reihenfolge für die Umsetzung definiert:

6.7.1. Priorität hoch

1. User Story US-14
2. User Story US-13
3. User Story US-4
4. User Story US-5
5. User Story US-8
6. User Story US-6
7. User Story US-7

6.7.2. Priorität mittel

8. User Story US-12
9. User Story US-10
10. User Story US-2

6.8. Testszenario

Aus den priorisierten User Stories können nun Akzeptanztests abgeleitet werden. Eine Userstory ist dann fertig, wenn sie technisch umgesetzt wurde und die dazu definierten Akzeptanztests abgenommen wurden. Jeder Akzeptanztest wird mit einer eindeutigen Nummer versehen: T-{User Story}.{Testnummer}

6.8.1. Auflistung der Akzeptanztests

Gemäss den hoch und mittel priorisierten User Stories ergeben sich folgende Akzeptanztests:

T-14.1 Es soll ein Datenobjekt auf dem Server gespeichert werden. Die Referenz des Datenobjekts soll man sich merken. Es soll das Datenobjekt welches man sich gemerkt hat über die ID wieder gelesen werden. Es soll die Gleichheit der beiden Datenobjekte geprüft werden.

T-13.1 Es soll irgendein REST Request auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden.

T-13.2 Es soll irgendein Hessian Request auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden.

T-4.1 Es soll ein REST HTTP-GET Request mit der Referenz auf eine bekannte Ressource von Prototypen aus auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung HTTP-Response-Message 200 geprüft werden.

T-4.2 Es soll ein Hessian

```
Product get(Long id)
```

Request mit der Referenz auf eine bekannte Ressource auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden.

T-5.1 Es soll ein REST HTTP-GET Request auf die Listenansicht einer bekannte Ressource, vom Prototypen aus, auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung HTTP-Response-Message 200 geprüft werden.

T-5.2 Es soll ein Hessian

```
Product [] allProducts()
```

Request auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden.

T-8.1 Es soll ein REST HTTP-POST Request auf die Listenansicht mit einer neuen Ressource, vom Prototypen aus, auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung HTTP-Response-Message 201 geprüft werden. Die registrierte Ressource soll über ein REST GET Request wieder geladen und verglichen werden.

T-8.2 Es soll ein Hessian

```
Product register(Product product)
```

Request mit einer neuen Ressource auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden. Die registrierte Ressource soll über ein Hessian

```
Product get(Long id)
```

wieder geladen und verglichen werden.

T-6.1 Es soll ein REST HTTP-DELETE Request auf eine bestehende Ressource, vom Prototypen aus, auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung HTTP-Response-Message 200 geprüft werden. Die gelöschte Ressource soll über ein REST HTTP-GET Request wieder geladen werden. Es soll eine HTTP-Response-Message 404 zurückgeliefert werden.

T-6.2 Es soll ein Hessian

```
Product delete(Long id)
```

Request auf eine bestehende Ressource auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden. Die gelöschte Ressource soll über ein Hessian

```
Product get(Long id)
```

wieder geladen werden. Dabei soll eine EntityNotFoundException zurückgeworfen werden.

T-7.1 Es soll eine bestehende Ressource genommen und verändert werden. Auf die bestehende Ressource soll ein REST HTTP-PUT Request vom Prototypen aus auf den Server abgesetzt werden. Es soll auf ein erfolgreiche Rückmeldung HTTP-Response-Message 200 geprüft werden. Die geänderte Ressource soll über ein REST HTTP-GET Request wieder geladen und verglichen werden.

T-7.2 Es soll eine bestehende Ressource genommen und verändert werden. Es soll ein Hessian

```
Product update(Long id, Product product)
```

Request mit der ID der bestehenden Ressource auf dem Server abgesetzt und auf ein erfolgreiche Rückmeldung geprüft werden. Die gelöschte Ressource soll über ein Hessian

```
Product get(Long id)
```

wieder geladen und verglichen werden.

T-12.1 Die Serverapplikation soll auf einem GlassFish Application Server, welcher Java EE 6 kompatibel ist, deployed werden. Es soll geprüft werden, ob die Applikation läuft.

T-10.1 Die Clientapplikation soll auf einem Android 2.2 kompatiblen Gerät installiert werden. Es soll geprüft werden, ob die Applikation läuft.

T-2.1 Die Clientapplikation soll über das Handynetzwerk mit dem Server kommunizieren. Wenn das funktioniert, kann die Applikation von überall her benutzt werden.

7. Ergebnisse der Tests und Abdeckung des Testszenarios

Da es sich hiermit um eine Semesterarbeit handelt wird nicht auf ein vollständiges Testing eingegangen. Das definierte Testszenario soll wenn möglich mit Unittests abgedeckt werden. Ein Test kann auch manuell ausgeführt und auf einen Erfolg geprüft werden.

Unittests in Java werden normalerweise mit JUnit durchgeführt. Für das Java EE Projekt kann JUnit verwendet werden. Für das Android Betriebsystem wird eine Test-Engine mit dem SDK mitgeliefert, mit der man Unittests definieren kann.

Manuelle Tests werden vom Studenten persönlich durchgeführt und überprüft und nach Treu und Glauben für korrekt oder fehlgeschlagen erklärt.

7.1. Testplan

Für das definierte Testszenario wird ein Testplan aufgestellt, dabei wird die Art und Weise der Durchführung, ob es sich um eine Unittest oder um einen manuelle Test handelt, aufgelistet. Zudem wird für jeden Test das Datum der Durchführung und das Ergebnis gezeigt. Der Testplan ist in der Tabelle 7.1 ersichtlich.

Akzeptanztest	Durchgeführt	Wann	Ergebnis
T-14.1	Manuell	08.02.2011	Erfolgreich
T-13.1	Manuell	08.02.2011	Erfolgreich
T-13.2	Unittest	08.02.2011	Erfolgreich
T-4.1	Manuell	08.02.2011	Erfolgreich
T-4.2	Unittest	08.02.2011	Erfolgreich
T-5.1	Manuell	08.02.2011	Erfolgreich
T-5.2	Unittest	08.02.2011	Erfolgreich
T-8.1	Manuell	08.02.2011	Erfolgreich
T-8.2	Unittest	08.02.2011	Erfolgreich
T-6.1	Manuell	08.02.2011	Erfolgreich
T-6.2	Unittest	08.02.2011	Erfolgreich
T-7.1	Manuell	08.02.2011	Erfolgreich
T-7.2	Unittest	08.02.2011	Erfolgreich
T-12.1	Manuell	08.02.2011	Erfolgreich
T-10.1	Manuell	08.02.2011	Erfolgreich
T-2.1	Manuell	08.02.2011	Erfolgreich

Tabelle 7.1.: Testergebnisse des Testszenarios

7.2. Abdeckung des Testszenarios

Da alle definierten Akzeptanztest, welche aus den User Stories abgeleitet wurden, am 08.02.2011 erfolgreich durchgeführt wurden, ist das Testszenario zu 100% abgedeckt.

Im Testszenario wurden die User Stories, welche tief priorisiert wurden, nicht berücksichtigt, da es sich dabei um optionale Akzeptanztests handelt.

8. Zusammenfassung zur Entwicklung des Prototypen

Da es sich bei dieser Semesterarbeit auch um eine praktische Arbeit handelt, werde ich in diesem Kapitel eine Zusammenfassung zur geleisteten Entwicklung geben. Ich werde auf die Nachvollziehbarkeit eingeben, zudem werde ich mich zu den verwendeten Tools äussern und die Zusammenfassung wird in der Form eines Erfahrungsberichts geschrieben.

8.1. Nachvollziehbarkeit

Die Nachvollziehbarkeit ist aufgrund des Versionierungssystems GIT[Wik11a] gesichert. In der Network-Ansicht von github kann man das inkrementelle Wachstum der Sourcen anhand der versionierten Commits verfolgen, auch kann man sehen, dass ich alles selbständig entwickelt habe.

8.2. Verwendete Tools

Da der Server- und der Client-Prototyp beide auf JAVA basieren habe ich die Entwicklung, teils auf meinem Heimrechner, teils auf meinem Notebook, gemacht. Auf meinem Heimrechner läuft das Betriebsystem Windows XP, auf meinem Notebook Mac OS X. Ich werde hier nur auf die Tools eingehen, welche ich auf meinem Heimrechner, mit Windows XP, verwendet habe.

8.2.1. Für das Server- und das Clientprojekt

- Windows XP Service Pack 3, von Microsoft, als Betriebsystem
- JAVA Java Development Kit (JDK) 1.6.0_22, von Oracle, als Laufzeitumgebung
- www.github.com, für die Verwaltung des Sourcecodes
- Eclipse Helios Service Release 1, Build id: 20100917-0705, als Entwicklungsumgebung

8.2.2. Für das Serverprojekt

- Oracle GlassFish Server 3.0.1, als Java EE Applikationsserver
- Oracle GlassFish Server Plugin for Eclipse, für die Verwaltung des Applikationsservers
- Rest Client 2.3.3 von http://code.google.com/p/rest-client/, zur Überprüfung der Schnittstelle
- Dynamischer Domain-Name-Service von no-ip.com, um den GlassFish Server auch von ausserhalb zu erreichen
- Derby DB für die Persistenz der Daten, wird mit dem GlassFish Server mitgeliefert

- Java Persistence API (JPA) 2.0 für die Abstraktion des Persistenzzugriffs, wird mit dem GlassFish Server mitgeliefert
- Enterprise JavaBeans (EJB) 3.1, für die Verwaltung von Enterprise Beans, wird mit dem GlassFish Server mitgeliefert
- Jersey, für den REST Webservice, wird mit dem GlassFish Server mitgeliefert
- Java Architecture for XML Binding (JAXB), für das Objectmapping von JSON oder XML zu POJO und umgekehrt, wird mit dem GlassFish Server mitgeliefert
- Java Server Faces 2.0, für die Repräsentation von Daten in HTML
- JUnit 4.8.2, für die Unittests
- Hessian 4.0.7, für den Hessian Webservice

8.2.3. Für den Client

- Android SDK r08, für die Android Version 2.2 Entwicklung
- Android Development Tools Plugin for Eclipse, für die Integration des Android SDK in die Entwicklungsumgebung
- Jackson 1.6.2, für das Objectmapping von JSON zu POJO und umgekehrt
- hessdroid 0.8.1, von http://code.google.com/p/hessdroid/, für den Hessian Webservice

8.2.4. Für die Dokumentation

- TeXnicCenter 1.0 Stable Release, für die LaTeX Entwicklung
- MiKTeX 2.9, für das erstellen von PDF-Dokumenten aus LaTeX

8.3. Erfahrungsbericht

Die detaillierte Umsetzung ist im Sourcecode nachzuvollziehen. Ich beschränke mich hier auf einen groben Überblick.

8.3.1. REST Client

Als erstes habe ich mich an den REST Client auf Android gewagt. Ich wollte mich so schnell wie möglich in das Thema Android einarbeiten. Dabei habe ich die Entwicklung gegen einen bestehenden RESTful Webservice auf Ruby on Rails Basis gemacht.

Schnell habe ich festgestellt, dass ich auf der Androidplattform kein JAXB Framework zur Verfügung habe, auch existierte keine Möglichkeit die Referenz Implementierung für einen REST Client von JAVA, namens Jersey, in die Androidplattform einzubinden. Das Problem dabei war die Dalvik Virtual Machine[Wik10], auf die die Androidplattform aufbaut. Sie akzeptiert nur ein Teil aller Java Packages. Wenn eine Abhängigkeit zu einem Java Package besteht, das nicht unterstützt wird, muss man einen anderen Weg finden, um zum Ziel zu kommen.

Für das Objektmapping und die Kommunikation mit einer REST Schnittstelle habe ich mich dann für die Standardkomponenten der Androidplattform entschieden. Dies sind die Klassen:

- org.json.JSONArray und org.json.JSONObject
- org.apache.http.impl.client.DefaultHttpClient

Nun habe ich noch ein kleines GUI für die Ansteuerung der REST Funktionen gebaut.

8.3.2. REST Server

Nach einer erfolgreichen Umsetzung des Clients habe ich mich an den Serverteil gemacht. Ich habe die REST Schnittstelle strikte nach dem Konzept von Jersey Implementiert. Bei der Lösung habe ich mich vom Buch "Real World Java EE Patterns"[Bie09] inspirieren lassen. Zudem habe ich einen Persistenzlayer für den Zugang zur Datenbank geschaffen. Um die REST Funktionen einfach zu testen, habe ich auch ein kleines GUI auf Basis von Java Server Faces 2.0 programmiert. In der Grafik 5.1 ist ersichtlich wie die REST Funktionen anzusprechen sind.

Die serverseitige Konfiguration beschränkt sich dabei auf eine Javaklasse, in der der Einstiegspunkt zum REST Service definiert ist. Diese Klasse sieht so aus:

```
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("")
public class RESTServlet extends Application {

}
```

Der GlassFish Applikationsserver erkennt über die Annotation @ApplicationPath(""), dass REST Requests an die Jersey implementierung weitergereicht werden sollen, sofern eine existiert.

8.3.3. Hessian Server

Den Hessian Service habe ich ganz nach der Anleitung der Hessian Dokumentation[CT11] implementiert. Der Service beschränkt sich auf eine Klasse namens ProductRegistrationServiceEndpoint welche von der Klasse HessianServlet ableitet und das Interface der Schnittstelle implementiert, siehe Grafik 8.1. Da ich bereits ein Persistenzlayer zur Datenbank geschaffen habe, konnte ich diesen wieder verwenden, um Datenobjekte zu persistieren, zu aktualisieren, zu löschen oder zu laden.

Die Konfiguration ist mit zwei Einträgen im web.xml File gemacht, siehe Grafik 8.2.

8.3.4. Hessian Client

Für den Hessian Client, der wiederum auf der Androidplattform laufen soll, funktionierte die Hessian Library nicht sofort. Mit der Unterstützung von Google bin ich auf das Projekt Hessdroid gestossen. Dieses Projekt zielt darauf ab, die Hessian Sourcen soweit anzupassen, damit sie auf der Dalvik Virtual Machine zum Laufen kommen. Ich habe mich daran gemacht, den bestehenden Hessian Service anzubinden. Wesentlich gibt es nur zwei Knackpunkte die zu erwähnen sind.

1. Die HessianProxyFactory musste nach meinen Erkenntnissen auf die Hessian Protokoll Version 1.0 gestellt werden.
2. In der ProxyFactory Create Methode muss Thread.currentThread().getContextClassLoader() für den ClassLoader verwendet werden.

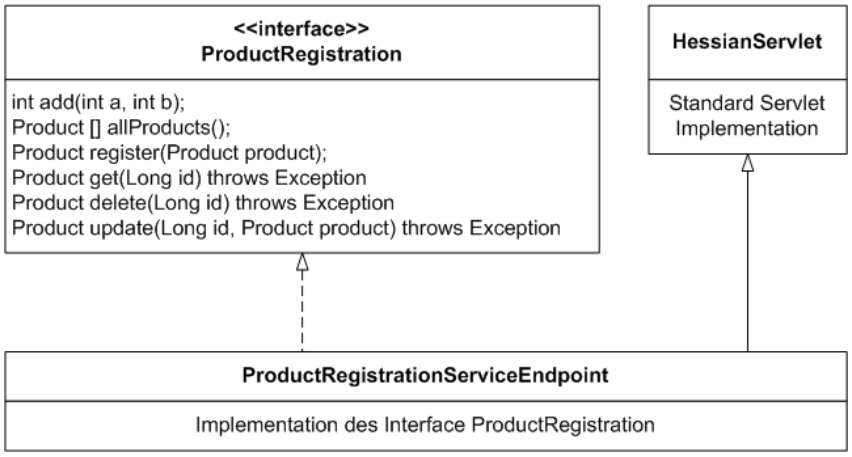

Abbildung 8.1.: Ein Produktservice mit Hessian bereitgestellt

web.xml

```
<web-app>

  <servlet>
    <servlet-name>
      ProductRegistrationService
    </servlet-name>
    <servlet-class>
      ProductRegistrationServiceEndpoint
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>
      ProductRegistrationService
    </servlet-name>
    <url-pattern>
      /ProductRegistrationService
    </url-pattern>
  </servlet-mapping>

</web-app>
```

Abbildung 8.2.: Konfiguration für den Hessian Service

9. Reflektion

Für mich hat von Anfang an der Schwerpunkt der Semesterarbeit immer auf der Entwicklung der Prototypen gelegen. Der Grund dafür war, dass ich davon am meisten profitieren konnte. Ich habe mich mit zwei Techniken vertraut gemacht, welche ich nun in der Praxis einsetzten kann. Zudem habe ich meine LaTeX Kenntnisse gravierend vertiefen können.

Rückblickend war es schade, dass ich diese Arbeit zu diesem Zeitpunkt schreiben musste, da mein Sohn Linus im Dezember zur Welt gekommen ist, und ich insgesamt über 100 Stunden vor meinen Rechnern verbracht habe, um alles zu vollenden. Natürlich liegt die Schuld voll und ganz bei mir selber, da ich viel früher mit der Semesterarbeit hätte beginnen können.

Schlussendlich war die Arbeit in meinen Auge ein voller Erfolg, da ich alle Ziele erreicht und viel gelernt habe, und dank der unkomplizierten Art und Weise von Beat Seeliger, in der Betreuerrolle, mein Bestes zum Vorschein gebracht habe. Ich bin glücklich, dass es vorbei ist, und bereit für die Diplomarbeit.

A. Abkürzungsverzeichnis

CORBA Common Object Request Broker Architecture
EJB Enterprise JavaBeans
GUI Graphical User Interface
HTML Hypertext Markup Language
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
IDL Interface Definition Language
JAXB Java Architecture for XML Binding
JDK Java Development Kit
JPA Java Persistence API
JSON JavaScript Object Notation
POJO Plain Old Java Object
REST Representational State Transfer
RPC Remote Procedure Call
SOAP Simple Object Access Protocol
URL Uniform Resource Locator
WWW World Wide Web
XML Extensible Markup Language

B. Abbildungsverzeichnis

C. Tabellenverzeichnis

D. Literaturverzeichnis

[Bie09] Adam Bien. *Real World Java EE Patterns*. Lulu.com, Juni 2009.

[CT11] Inc Caucho Technology. Hessian Dokumentation. `http://hessian.caucho.com/`, 2011. [Online; 15. Februar 2011].

[DOS09a] Studienleiter Informatik Dr. Olaf Stern. Ablauf semesterarbeit. `https://ebs.hsz-t.ch/files/ebs_files/Reglemente/Kreditsystem/Semesterarbeit/Ablauf-Semesterarbeit_Studiengang-Informatik-der-HSZ-T_V1.4.pdf`, Juni 2009.

[DOS09b] Studienleiter Informatik Dr. Olaf Stern. Bewertungskriterien semesterarbeit. `https://ebs.hsz-t.ch/files/ebs_files/Reglemente/Kreditsystem/Semesterarbeit/Bewertungskriterien-Semesterarbeit_V1.xls`, Januar 2009.

[DOS09c] Studienleiter Informatik Dr. Olaf Stern. Reglement semesterarbeit. `https://ebs.hsz-t.ch/files/ebs_files/Reglemente/Kreditsystem/Semesterarbeit/Reglement-Semesterarbeit_Studiengang-Informatik-der-HSZ-T_V3.7.pdf`, Oktober 2009.

[Fie00] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`, 2000. [Online; 16. Februar 2011].

[Mic08] Sun Microsystems. JAX-RS: The Java™ API for RESTful Web Services. `http://www.jcp.org/aboutJava/communityprocess/final/jsr311/index.html`, 2008. [Online; 14. Februar 2011].

[Wel99a] Don Wells. Acceptance Tests. `http://www.extremeprogramming.org/rules/functionaltests.html`, 1999. [Online; 27. Dezember 2010].

[Wel99b] Don Wells. User Stories. `http://www.extremeprogramming.org/rules/userstories.html`, 1999. [Online; 27. Dezember 2010].

[Wik10] Wikipedia. Dalvik Virtual Machine. `http://de.wikipedia.org/wiki/Dalvik_Virtual_Machine`, 2010. [Online; 15. Februar 2011].

[Wik11a] Wikipedia. GIT. `http://de.wikipedia.org/wiki/Git`, 2011. [Online; 15. Februar 2011].

[Wik11b] Wikipedia. KISS. `http://de.wikipedia.org/wiki/KISS-Prinzip`, 2011. [Online; 14. Februar 2011].

[Wik11c] Wikipedia. Ruby on Rails. `http://de.wikipedia.org/wiki/Ruby_on_Rails`, 2011. [Online; 14. Februar 2011].