Design and implementation of an end-user programming software system to create and deploy cross-platform mobile mashups
Abstract

Significant changes in mobile computing are continuously influenced by the enhanced features of mobile devices and software applications. The release of open platforms that allow the creation of advanced mobile applications are the triggers for recent developments with regard to the topics of end-user mobile application frameworks and mobile mashup creation.

Inspired by these efforts, this thesis identifies some of the problems in this field and presents a solution for a cross-platform end-user programming software system that enables the creation of mobile mashups.

Keywords: Android, Cross-platform mobile frameworks, Google Web Toolkit, iPhone, Java, JavaScript, Mashups, Mobile mashups, Web 2.0 APIs, Web development frameworks
# Table of Contents

1 Introduction .................................................................................................................. 1  
1.1 Definition of the problem ..................................................................................... 3  
1.2 Structure of the thesis ......................................................................................... 3  
1.3 Delimitations ........................................................................................................ 4  
2 State of the art ......................................................................................................... 5  
  2.1 Literature study .................................................................................................... 5  
  2.1.1 Features of mobile mashups ......................................................................... 5  
  2.1.2 Features of an end-user programming software system ............................. 7  
  2.2 Project idea .......................................................................................................... 8  
  2.3 Web development frameworks ........................................................................... 9  
  2.3.1 Web framework decision ............................................................................. 10  
  2.4 Cross-platform mobile development ................................................................ 10  
  2.4.1 Technology .................................................................................................... 10  
  2.4.2 Cross-platform mobile framework comparison ............................................. 10  
  2.4.3 Cross-platform mobile framework decision .................................................. 11  
  2.4.4 Further developments .................................................................................... 11  
  2.5 Summary of results ............................................................................................. 12  
3 Software requirements .............................................................................................. 13  
  3.1 Mobile Mashup Scenarios ................................................................................... 13  
  3.1.1 Scenario 1 – Mobile self-presentation ....................................................... 13  
  3.1.2 Scenario 2 - Mobile learning with location aware content ....................... 14  
  3.1.3 Scenario 3 - Record sensor data and environmental sources ..................... 15  
  3.2 Stakeholders........................................................................................................ 15  
  3.3 Software system specifications .......................................................................... 16  
  3.4 Requirements analysis - Mobile Mashup Editor ............................................... 16  
  3.4.1 Functional requirements ............................................................................. 16  
  3.4.2 Non-Functional Requirements ................................................................... 17  
  3.5 Use case modelling – Mobile Mashup Editor ..................................................... 17  
  3.5.1 Use cases ..................................................................................................... 17  
  3.5.2 Use cases model ............................................................................................ 18  
  3.6 Requirements analysis - Mobile Mashup Viewer ............................................. 18  
  3.6.1 Functional Requirements .......................................................................... 19  
  3.6.2 Non-Functional Requirements ................................................................... 19  
  3.7 Use case modelling – Mobile Mashup Viewer ..................................................... 19  
  3.7.1 Use cases ..................................................................................................... 19  
  3.7.2 Use case model ............................................................................................ 20  
  3.8 Operating requirements ...................................................................................... 20  
  3.8.1 Hardware ..................................................................................................... 20  
  3.8.2 Software ....................................................................................................... 20  
4 Software analysis and design .................................................................................... 22  
  4.1 OOA class diagram - Mobile Mashup Editor ..................................................... 22  
  4.2 Scenario 1 – Mobile self-presentation ................................................................ 23  
  4.3 OOA class diagram - Mobile Mashup Viewer .................................................... 26  
  4.4 Scenario - Deploy a mashup ................................................................................. 27  
  4.5 Conclusion requirements and analysis process ................................................. 28  
  4.6 Software design .................................................................................................. 28  
  4.6.1 Mobile Mashup Editor .................................................................................. 28  
  4.6.2 Mobile Mashup Viewer ................................................................................ 31  
  4.6.3 Mobile Mashup description .......................................................................... 32  
  4.7 Interface design .................................................................................................. 33
Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Example of existing APIs taken from programmableWeb.com</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Mashup categories taken from programmableWeb.com</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Derived project idea</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Use case diagram (Mobile Mashup Editor)</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Use case diagram (Mobile Mashup Viewer)</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>OOD class diagram (Mobile Mashup Editor)</td>
<td>22</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Activity diagram 1: Create a new mashup</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Activity diagram 2: Layout the &quot;Welcome&quot; page</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Activity diagram 3: Create and layout the &quot;My pictures&quot; - Flickr page</td>
<td>25</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Activity diagram 4: Create and layout the &quot;News&quot; - Twitter page</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>OOA class diagram (Mobile Mashup Viewer)</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Activity diagram 5: Deploy a mashup</td>
<td>28</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>OOD class diagram (Mobile Mashup Editor)</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.9</td>
<td>Package structure (Mobile Mashup Editor)</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.10</td>
<td>Package structure (Mobile Mashup Viewer)</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4.11</td>
<td>Sequence diagram: Publish mashup</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.12</td>
<td>User-interface panel arrangement</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.13</td>
<td>User-interface class diagram (Mobile Mashup Editor)</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.14</td>
<td>Mashup page user-interface design</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4.15</td>
<td>Published mashups</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.16</td>
<td>Mashup details</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.17</td>
<td>Mashup pages</td>
<td>35</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Mobile Mashup Editor architecture</td>
<td>36</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Mobile Mashup Viewer architecture</td>
<td>37</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Screenshot mashup components tree</td>
<td>39</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Screenshot Upload dialog</td>
<td>40</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Screenshot user interface (Mobile Mashup Editor)</td>
<td>42</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Additional component development workflow</td>
<td>45</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>Deployment diagram of the software system</td>
<td>48</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Screenshot Pages menu</td>
<td>49</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Screenshot Pages settings</td>
<td>49</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Screenshot Layout components</td>
<td>49</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Screenshot header pane</td>
<td>50</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>Screenshot Label component configuration</td>
<td>50</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Screenshot Layout of the Welcome page</td>
<td>50</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Screenshot Layout of the Pictures page</td>
<td>51</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>Screenshot Layout of the Contact page</td>
<td>52</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>Screenshot Publish dialog</td>
<td>52</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>Published mashups</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>Mashup details</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.12</td>
<td>Welcome page</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.13</td>
<td>Pictures page</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.14</td>
<td>Contact page</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.15</td>
<td>Call number comp.</td>
<td>53</td>
</tr>
<tr>
<td>Figure 6.16</td>
<td>Published mashups</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.17</td>
<td>Mashup details</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.18</td>
<td>Welcome page</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.19</td>
<td>Pictures page</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.20</td>
<td>Contact page</td>
<td>54</td>
</tr>
<tr>
<td>Figure 6.21</td>
<td>Call number comp</td>
<td>54</td>
</tr>
</tbody>
</table>
1 Introduction

Mobile computing is becoming the largest Internet platform (ITU, 2009); the mobile Internet is used more than ever before. Devices are getting more powerful and have turned the ordinary mobile phone into small computers.

“Mobile cellular has been the most rapidly adopted technology in history. Today it is the most popular and widespread personal technology on the planet, with an estimated 4.6 billion subscriptions globally by the end of 2009. In 2009, more than a quarter of the world’s population are using the Internet” (ITU, 2009).

Background

With the release of the iPhone in the year February 2007 the company Apple paved the way for a new kind of mobile devices. The biggest innovations were the new way of user interaction through a big multi-touch screen and the combination of advanced hardware with novel software solutions. Especially the Mobile Safari web browser termed in (Maximilien, 2008) as a “revolutionary Web browser …” that “… has opened the flood gate on how Web content can be accessed and delivered”, has to be emphasized.

Beside the revolutionary iPhone device, the release of a software development kit (SDK) for the device platform made this product even more usable and successful. The iPhone SDK, which was released in March 2008, enables third-party developers to create advanced mobile applications that can access and use the hardware features of the mobile device. The App Store, the central platform to provide and download theses applications, contains today (22.01.2010) more than 125,000 applications.

“Apple's success with the App Store has prompted other players to focus on devices that can enable third-party developers to easily bring applications and services to mobile phones.” (Leach, 2009). The most competitive one, the Android platform, was invented by the company Google and released on the first mobile device in October 2008. Google released also a SDK for the Android platform. The Android Market contains by now (22.01.2010) more than 20,000 applications.

Today big manufacturers like HTC and Sony Ericsson have released devices with the Android platform. A statistic provided by (Leach, 2009) says “we expect Android shipments to reach 72 million units by 2014, representing 18% of the market and overtaking shipments of Windows Mobile”. This statistic is a forecast on the growing market for powerful mobile devices. GPS sensors, connection with the Internet and a camera can be expected as standard features of future mobile devices. The market for mobile application market will grow in coordination with this market. The more powerful these devices will get, the more powerful their applications will become. Nowadays, these applications provide an optimal collaboration between the device and the use of its hardware features.

Challenges of mobile application development

In mobile development, developers have to be aware of the restrictions in the mobile phone hardware. Namely a small screen, limited user input possibilities, low memory space and slower processors. By releasing SDKs the platforms inventors provide development tools that are optimized for the particular platform and device hardware. The drawback of these powerful SDKs is that they are tied to the platform and use different programming languages as for example Object-C for the iPhone, Java for Android or C++ for Symbian devices. To implement applications that shall work on
different platforms, developers need therefore knowhow in different programming
languages and have to implement their solutions for every platform individually.

Another recent topic in the mobile development domain is that end-users without any
programming knowledge have no possibilities to develop mobile applications. The
recently published report (Bosch, 2009) emphasized that “no successful end-user
programming software ecosystems exist yet for the mobile domain”.

By integrating the feature to create cross-platform mobile applications in such a system
it would not only be useful for end-users, it would also become useful for mobile
application developers. With using a cross-platform developing system, developers
have not the need to be aware off all different programming languages that are required
by the platforms any more.

Changes and challenges of the Web
The success of the Web 2.0 has changed the way people use the Internet. Everybody
can easily provide and use the digital content of the Internet. Services that store this
content and information about events, places, relationships, media, messaging, geo
locations etc. have grown rapidly. Most of these services have made their data and
functionalities accessible through an Application Programming Interface (API). APIs
exist for almost all services that are part of the Web 2.0.

The use of APIs can make application development easier and faster, because all the
common methods and functionalities that developers may need are done within the
APIs. Developers do not have to implement them, but can reuse them. The advantage of
Web APIs is that they are provided over the Internet and therefore easily accessible for
developers. The success of the Web APIs has evolved the concept of mashups. A
Mashup uses APIs from different service providers and integrate them to create new,
innovative, useful applications and services. The statistic of the mashup resource
programmableWeb shows examples of Web API providers and the percentage of usage
of their particular API within the 4597 (as of 22.01.2010) registered mashup
applications (Figure 1.1).

![Pie chart showing API usage](http://example.com/mashup-charts.png)

*Figure 1.1 - Example of existing APIs taken from programmableWeb.com (22.01.2010)*

Desktop mashups have shown the “power of integrating Web data and process to
create novel and situational Web applications” (Maximilien, 2008). However, many
desktop-mashup implementations, especially the location-based, would be more
meaningful on a mobile device. Maximilien (2008) also suggested “linking mobile
devices with Web content is leading the way to a shift in how humans communicate,
interact, and socialize”.

A second statistic of programmableWeb shows the categorization of existing
mashups (Figure 1.2). Over one third are geo-based mashups, these mashup would
become more powerful with the knowledge of the current location of a user. By
knowing the current position these mashups, and also mashups of the categories travel,
shopping and social, could providing nearby information to the user when and where they are meaningful and appropriate for them.

![Mashup categories taken from programmableWeb.com (22.01.2010)](image)

Figure 1.2 - Mashup categories taken from programmableWeb.com (22.01.2010)

In the desktop world different Mashup editors exist to provide end-user programming solutions. With tools like Intel Mash Maker, Dapper, or Yahoo Pipes the creation of customized mashups for end-user is already possible today, but none of these tools is meant for creating mobile mashups. The most of created mashups can run inside a mobile web browser. However not as native mobile applications and without capabilities for access and use the hardware features of the mobile device. Yahoo! Pipes provide a solution to create mashups optimized for the use on the iPhone, but also these mashups work only in the web browser of the device.

1.1 Definition of the problem

The aim of the thesis is to explore ways to solve the presented drawbacks in the mobile application development and mobile mashup domain, such as developers have to be aware of the restrictions in the mobile phone hardware, SKDs are tied to platforms and therefore developers have to know different programming languages and implement applications for every platform individually, a missing end-user programming solution and the lack in possibilities to create mashups optimized for mobile usage. Therefore an approach shall be developed based on the question presented below:

> Which are the requirements for the design and development of an end-user programming software system that supports the creation and deployment of cross-platform mobile mashups?

In order to answer this question the related sub-questions can be formulated as follows:

a. What are the most suitable, currently available technologies to implement such a software system?

b. How should the architecture for such a system look like?

c. What are the potential benefits of using this software system for mobile development?

1.2 Structure of the thesis

The thesis starts with an overview of latest work and research projects in the topic of mobile application development and mobile mashups. From identified requirements and features in this overview the project idea is derived and presented. Within the chapter “State of the art” an evaluation of suitable technologies for realising the project ideas is conducted. This evaluation concludes with the introduction of the technologies and frameworks that will be used for the implementation. The Software Engineering process
is documented in the chapters “Software requirements”, “Software analysis and design”. The “Implementation” chapter presents the system architecture and highlights important parts of the prototypical implementation of the software system. The main text of the thesis ends with the test and exploration of the developed prototype. The conclusion contains a discussion on the achieved results and an outlook of future work.

1.3 Delimitations

The focus of the prototype implementation is not on the general mashup concept to combine as many different Web API services as possible, but rather to create a basis, extensible software system.

Aspects of privacy and trust are not considered in the software system. Based on the used cross-platform mobile framework, the mobile mashups at the current version can only be deployed on the platforms iPhone and Android.

For the purposes of the thesis the potential benefits of the system are illustrated with user scenarios and not through an evaluation with users.
2 State of the art

This section contains a summary of the current research works and articles in the topics related to mobile computing, mobile application development, mobile mashups and visual programming. Derived from this study the project idea is presented. Subsequent new technologies that could be used to realize the project idea are explored.

2.1 Literature study

The literature study was conducted to explore the state of the art in mobile computing and mobile mashups and to identify the features of the envisioned end-user programming software system. Different activities were carried out that included search and analysis of scientific papers, technical reports and books in this field. The section below describes those identified features.

2.1.1 Features of mobile mashups

This section contains the description of usage scenarios and features for mobile mashups, identified in the literature study.

Mobile mashup scenarios

The papers describe different mobile mashup scenarios as meaningful on a mobile device. Four of these scenarios are presented below:

1. The first scenario uses the current location data provided by the device to send or get location-based information. “The mobile phones of the teens are able to broadcast location data that would allow parents to keep a watch on the routes taken as well as allowing the teens to be alerted of friends from their Facebook profile who are near…” (Maximilien, 2008). Combining the location information of a person with information about this person, for example from his/her Facebook profile, enables a new way mobile commerce. For example a cinema service could send information about currently playing movies in the genre the person likes, when the person is nearby a cinema that is listed in this service.

2. A second scenario describes a mobile mashup that could help to improve health and social care system in countries in the third world. Based on the fact that a lack of doctors exists in these countries and that they have to “… move about the countryside to provide care to the most needy based on regular rounds of care”. A scenario could be: “Doctors could use mobile devices to record health information and keep tab on patients thereby allowing them to address a vast number of patients than would normally possible.” (Maximilien, 2008).

3. The third scenario describes the creation and providing of location-aware content “Another service could enable a user to leave virtual Post-its, photos, or videos for friends to discover as they get near them.” (Jensen et al., 2008).

4. The last scenario focuses on content creation and real time providing of these contents “… users publish their vacation travel experiences in real time for family, friends, and the general public to react to, by continually uploading georeferenced content.” (Jensen et al., 2008).

The described scenarios use the location information provided by the mobile device and allow mobile content creation. Accesses to the device sensor data, the device content as well as creation of (location-aware) content through the use of the device components (e.g. camera), are therefore key features for successful mobile mashups.
Location information
The importance of the location information has been suggested by Nokia Report (Nokia, 2009) were they describe it as information that has “the most potential” for new services. “That’s because, once this is known, there are so many other pieces of data that can be inferred contextually. The weather we’re expecting for example” or “…the language that we’re speaking”.

Mobile sensing
A goal of mobile or ubiquitous computing as described in (Battestini et al., 2009) is to ”enable devices to sense changes in their environment and automatically adapt to these changes based on user needs and preferences ”.

Sheth (2009) explains that sensing is already easily possible today. The data people create and “the use of Internet- or Web-enabled mobile devices to upload this data” allows “these devices to act as sensors”. He explained the term citizens sensor network that refers to a network of people who “actively observe, report, collect, analyze, and disseminate information via text, audio, or video messages.” This “combination of human-in-the-loop sensing, Web 2.0, and mobile computing has led to the emergence of several citizen sensor networks.” (Sheth, 2009).

Furthermore he emphasized “two significant developments in mobile computing” that helped to enable these kind of networks first the “….enhanced features such as GPS capability and cameras became a standard part of most mobile devices” and second that “large companies created open mobile operating systems, such as Apple’s OS X for the iPhone and Google’s Android” (Sheth, 2009).

Content sharing
Microblogging - “in which user share short messages and pictures, typically over the Web” describes the author of (Sheth, 2009) as “particular interest of citizen sensors”. The integration and use of microblogging services like Twitter and other social network services like Facebook has to be possible within the mobile applications.

Cross-platform applications
(Chaisatien and colleagues (2009) describe the ideas for a “Web-based Mashup Tool”. They mention the disadvantage of most of today’s mobile application when they say “Most mobile applications are bound with device specification and with the web services they use…”. Cross-platform working applications can avoid this disadvantage.

Privacy and trust
Beside the advantages these new kind applications can provide to the users the important aspects; privacy and trust have to be considered as well. As explained in the report (Battestini et al., 2009) “… Next-generation applications for mobile devices are typically user-centric and personal. They process and exchange personal user data or information about the user’s context. Therefore it is important that adequate mechanisms are in place to protect the user’s privacy and avoid the misuse of the system … a user-centric privacy and trust framework for controlling the access to user data is required …”. The authors of (Jensen et al., 2008) name these aspects also when they say that a service platform “… should support basic aspects of services such as authentication, security, and privacy …”. 
Derived from the described scenarios, articles and papers, the Table 2.1 summarizes all identified suggested features for mobile mashups:

<table>
<thead>
<tr>
<th>Features of mobile mashups</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The device sensor data and content is accessible and can be used. The use of the location information shall allow the creation of location-aware content.</td>
</tr>
<tr>
<td>2</td>
<td>Microblogging, social networks and other Web 2.0 services can be used to enable mobile sensing through the possibilities of sending, receiving and sharing data.</td>
</tr>
<tr>
<td>3</td>
<td>The mobile applications are not bound on device specification and work cross-platform.</td>
</tr>
<tr>
<td>4</td>
<td>Authentication, security and privacy have to be ensured.</td>
</tr>
</tbody>
</table>

*Table 2.1 - Summarization of features for mobile mashups*

### 2.1.2 Features of an end-user programming software system

The following section contains the description of features for an end-user programming software system to create mobile mashups.

**End-user requirements**

Bosch (2009) characterizes the target group for an end-user solution when he writes that they have a “good domain understanding, but no computer science or engineering degree. Hence the basics of computing and programming cannot be assumed to be understood by this group…” Jensen and colleagues (2008) demand that an end-user software system shall make “it equally easy for ordinary Internet users to create such services.” The creation shall “not be limited to professional computer programmers. Rather, all those who can contribute to Facebook and You-Tube should be able to create mobile services”. Hence a solution shall provide a task-oriented, graphical user interface with a graphical representation of the mobile application components. It shall also works with the What You See Is What You Get (WYSIWYG) techniques to prove that the requirement to “understand with minimal instruction and to make application creation intuitive and requiring minimal understanding by the end-user” (Bosch, 2009) is fulfilled. Another way to accomplish this is to “enable programmers to create ‘service templates’ that nonprogrammers can fill in to configure and create services.” (Jensen et al., 2008).

**Component-based solution**

“End-user programming often is concerned with creative composition of pre-created building blocks, rather than the creation of fundamentally new functionalities …“ for that reason Bosch (2009) suggests that the software system provides “pre-created components that can be combined.” Trevor (2008) introduces the Yahoo! Pipes, as an end-user service platform to create mashup. Pipes provide modules with “high-level data-processing functions such as geocoding locations in feed items or translating from one language to another. “

**Share mashups**

The following two papers point out that a sharing of the created application or services to other users are also necessary. “Providing effective ways of sharing applications in ways that allows other customers to adopt, adjust and use these solutions” (Bosch, 2009). “A successful ecosystem for user-generated mobile services includes the ability to share services. Creators of services must be able to advertise them to other users who can then subscribe.” (Jensen et al., 2008).
Visual programming

A successful end-user programming software system shall use concepts and techniques from visual programming as the way for adding and combining the components of a mashup. Trevor (2008) explains the features of Yahoo! Pipes, he emphasizes that “developers of all capabilities can use Pipes’ Web-based visual programming environment to combine data sources and user input into mashups without having to host or write any code.” To create and combine a mashup the developer can “drag modules from a toolbox … onto the Pipes canvas.” The author explains also that these modules can be “configured directly on the canvas” (Trevor, 2008).

Derived from the described features, the Table 2.2 summarizes all identified suggestions for an end-user programming software system to create mobile mashups:

<table>
<thead>
<tr>
<th>Features of an end-user programming software system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. No programming knowledge is required to create mobile mashups.</td>
</tr>
<tr>
<td>2. The software system provides its functionalities as components that can be configured.</td>
</tr>
<tr>
<td>3. The created mashups can be shared with other users.</td>
</tr>
<tr>
<td>3. The software system for creating these mashups works with visual computing concepts and technologies.</td>
</tr>
</tbody>
</table>

Table 2.2 - Summary of the features for an end-user programming software system

2.2 Project idea

Derived from the identified features on the software system, the Figure 2.1 shows the developed project idea. It is presented as a comparison of today’s desktop mashup domain with the added new mobile mashup domain ideas that will be explored within the thesis.

Figure 2.1 – Derived project idea

Desktop mashups allow the combination of different Web APIs services to new applications that are usually published on a web server and run within a standard web browser.
To provide enhanced customization features, a mobile mashup will be combined from different component types. *Service components* are components that provide core functionalities for a mashup like the access to important Web APIs and/or combination of them. *Layout components* are simple components that are used to present information and content like text or images. *Device components* are components to access the advanced features of today’s mobile device functionalities such as: GPS sensor data, photo camera, and access to the device content. *Additional components* provide the extensible part of the software system. The design and implementation of the solution shall make it possible to develop and integrate new mashup components into it. These components will be developed upon the needs of the users or to integrate new technologies for mashups.

As other mashup editor solutions, the editor for creating the mobile mashups shall provide a graphical user interface that enables to combine the mashup components using the *What You See Is What You Get* (WYSIWYG) technique.

From the editor, the mashups can be published to a service directory on a web server to make them accessible for users. A mashup viewer on the mobile device provides access to the remote mashup service directory. The user utilizes the viewer to subscribe to mashups with their mobile devices. In difference to the desktop domain the mobile mashups will be deployed and run as native applications on the device and not within the web browser. This is necessary to fulfil the requirement to access the sensor data and content of mobile devices within a mashup. Only native applications provide this access that is not possible within the mobile web browser. The deployed applications have the layout, content and functionalities that were defined by the author of the mashup.

### 2.3 Web development frameworks

Web applications have considerable advantages such as cross-platform running, usage via a standard web browser, users do not need to install an application and updates have only to be deployed on the web server to provide them to the users. Therefore the editor to create the mobile mashups shall be implemented as a web application.

The editor is targeted to end users and shall therefore provide an enhanced graphical user interface, behave like applications the user is already familiar with and not require any additional installations to use it. Rich Internet Applications (RIA), are web applications that appear and behave like desktop applications. Today several frameworks to develop RIA exist. The main web development frameworks currently existing are *Adobe Flex*, *Microsoft Silverlight* and *Google Web Toolkit* (*GWT*). The Table 2.3 shows a comparison of these frameworks and their features:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Adobe Flex</th>
<th>Microsoft Silverlight</th>
<th>Google Web Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open source</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Platform independent development tools</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Additional installations necessary</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Programming language</td>
<td>Action-Script</td>
<td>.Net languages</td>
<td>Java</td>
</tr>
</tbody>
</table>

*Table 2.3 – Web development framework comparison (last update: 11.01.2010)*

**Legend:**  
Y – feature supported  
N – feature not supported
2.3.1 Web framework decision
The framework Google Web Toolkit generates web applications based on JavaScript, HTML and CSS and requires just a web browser with enabled JavaScript to work with the application. Web applications based on Adobe Flex or Silverlight have the drawback that the user has to install a proprietary plug-in to be able to work with them. The further development of the software system shall be possible for developers on all platforms for that reason the development framework has to be platform independent. This applies for the Adobe Flex and GWT framework. Microsoft Silverlight development tools are only available for the Windows platform.

For that reasons GWT was chosen as framework for the development of the mashup editor. GWT has also the advantages that it is free of charge, open source and uses the programming language Java that enables to develop complex web application as an object oriented Java project. It is also the fact to mention that Google uses GWT to develop its own applications like Google docs or Google mail. Through the popularity of these tools, it can be assumed that a wide range of users is familiar with the handling and user-interface of these kinds of applications.

2.4 Cross-platform mobile development
This section contains a research within the field of cross-platform mobile frameworks and development. The decision to use a cross-platform framework comes from the intention to not limit the created mobile mashups to a certain device or platform.

Since the success of the iPhone and the Android platform, the market of mobile cross-platform frameworks is increasing. The outcomes of the technical review pointed out that by 2009 three promising frameworks were released, PhoneGap, Titanium Mobile and Rhomobile. The productivity and functionalities of these three frameworks will be explored in the following sections.

2.4.1 Technology
Cross-platform frameworks allow developers to implement mobile applications in a familiar programming language with the advantage that an application has to be implemented just once and can then be deployed on many platforms. The framework compiler compiles the language of the framework into native code of the different platforms. Most of these frameworks for the mobile domain use the technologies JavaScript, HTML and CSS to implement and layout mobile applications. Web developers can therefore implement mobile application without learning a new language like Object-C for the iPhone or Java for the Android platform.

2.4.2 Cross-platform mobile framework comparison
This section contains the results of the tests of the three cross-platform mobile frameworks PhoneGap, Rhomobile and Titanium Mobile. The test is conducted to ascertain which framework has the potential to fulfil the requirements for the viewer application of the software system. The test results and detailed information about used technologies, the license model, supported devices and features are documented in the Appendix C – Cross-platform mobile frameworks. A summary of the test results is given in Table 4. The Table 2.4 provides an overview over the tested frameworks, their currently supported devices and devices features.
### Table 2.4 - Framework comparison (last update: 11.01.2010)

<table>
<thead>
<tr>
<th>Framework</th>
<th>PhoneGap</th>
<th>Titanium</th>
<th>Rhomobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature / Device</td>
<td>I A B S W</td>
<td>I A B S W</td>
<td>I A B S W</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>Y Y Y Y *</td>
<td>Y Y * * *</td>
<td>N N N N N</td>
</tr>
<tr>
<td>Camera</td>
<td>Y Y Y Y *</td>
<td>Y Y * * *</td>
<td>Y Y Y Y Y</td>
</tr>
<tr>
<td>Contact support</td>
<td>Y Y Y Y *</td>
<td>Y Y * * *</td>
<td>Y Y Y Y Y</td>
</tr>
<tr>
<td>Geo Location</td>
<td>Y Y Y Y *</td>
<td>Y Y * * *</td>
<td>Y Y Y Y Y</td>
</tr>
<tr>
<td>Local DB</td>
<td>Y Y N N *</td>
<td>Y Y * * *</td>
<td>Y Y Y Y Y</td>
</tr>
<tr>
<td>Photo Gallery</td>
<td>N N N N N</td>
<td>Y Y * * *</td>
<td>N N N N N</td>
</tr>
<tr>
<td>Push Notification</td>
<td>N N N N N</td>
<td>Y N * * *</td>
<td>Y N Y N N</td>
</tr>
<tr>
<td>Sound</td>
<td>Y Y Y Y Y</td>
<td>Y Y * * *</td>
<td>N N N N N</td>
</tr>
<tr>
<td>Vibration</td>
<td>Y Y Y Y *</td>
<td>Y Y * * *</td>
<td>N N N N N</td>
</tr>
<tr>
<td>Video</td>
<td>N N N N N</td>
<td>Y Y * * *</td>
<td>N N N N N</td>
</tr>
</tbody>
</table>

**Legend:**
- I – iPhone
- A – Android
- B – Blackberry
- S – Symbian
- W – Windows Mobile
- Y – feature supported
- N – feature not supported
- * – feature not or not fully supported yet, but on the roadmap

#### 2.4.3 Cross-platform mobile framework decision

The decision for one of the frameworks is based on the supported technologies, features, the roadmap for the further development and the provided support.

**PhoneGap** is not currently a suitable solution because of its lack of documentation and tutorials. **Titanium Mobile** and **Rhomobile** worked very well in the first tests. The **Rhomobile** framework does not have support for several multimedia technologies. For this reason, this framework was discarded as multimedia technologies can provide multiple functionalities to mobile mashup applications.

The **Titanium Mobile** framework offers the most features, but currently it provides support only for *iPhone* and *Android*. The roadmap of the framework development contains the other important mobile platform **RIM Blackberry**, **MS Windows Mobile**, **Symbian** and **Palm Pre**. This framework offers also the best support in documentation, tutorials and sample projects.

For that reasons the **Titanium Mobile** framework shall be used as the technology for the implementation of the software component that deploys the mobile mashups. The framework is based on web standards and web technologies therefore the component will be implemented as a mobile web application, using HTML, CSS, JavaScript and Ajax.

#### 2.4.4 Further developments

Mobile development and the new technologies of cross-platform mobile frameworks are an increasing market that is still under development. During the thesis work **Sony Ericsson**, in cooperation with **PhoneGap**, released a cross-platform mobile framework, the **Sony Ericsson WebSDK**. This framework has not been evaluated in detail. It supports the platforms **Android** and **Symbian** and works, like **Titanium Mobile** and **PhoneGap**, with the technologies **HTML**, **CSS** and **JavaScript**. This release proves again the growing market of these frameworks. The fact that big companies like **Sony Ericsson** invest in these kinds of frameworks proves that the technology is working.
2.5 Summary of results

The literature study has shown the requirements for today’s mobile domain. Summing up the access to device sensor data and content, use of Web 2.0 services and have to work cross-platform. The developed project idea covers these requirements. To realize this idea different technologies were explored and the following decisions were made. An editor application will be provided to enable end-users to create mobile mashups within a graphical user interface. This editor shall be implemented as web application with the Goggle Web Toolkit framework. These mobile mashups are provided to mobile users within a viewer application. The viewer shall be implemented with the technologies JavaScript, HTML and CSS and the mobile cross-platform framework Titanium Mobile.
3 Software requirements

This chapter contains the documentation of the software requirements specification. The results of this specification are the requirements and use case model for the software system.

The chapter starts with a description of possible usage scenarios for the software system. These scenarios were identified on the basis of project papers and in interviews with project members. The requirements for the prototype implementation shall define a standard, extensible solution for the software system. It also caters for the features identified in all scenarios. Features specific to single scenarios are not considered here.

Because the development of the software system is a feasibility study, the non-functional requirements are restricted to the technologies to use and will not consider requirements regarding portability, reliability, performance etc.

Method

The software requirements and use case modelling approach follows the guidelines and recommendations of the Unified Process (UP) described in the book “UML 2 and the Unified Process” (Arlow, J. et al., 2005), chapter three to five.

3.1 Mobile Mashup Scenarios

In addition to the scenarios presented in the literature study this section describes in detail three usage scenarios for mashups and their creation with the mashup editor. These scenarios will be used to reveal the advantages of the solution and to identify the requirements and use cases for the software system.

3.1.1 Scenario 1 – Mobile self-presentation

The first scenario is a simple scenario to point out the usage of the software system with its basic components and functionalities. For that reason this scenario shall be realizable with the prototype implementation of the system.

Users can utilize the mashup editor to create a mobile application to present themselves, their business, company or interests. The author of the mashup can be a private person, or a person in a business.

In a mashup for realizing a self-presentation the author combines different components. To present static information and to layout the mashup, components like labels, pictures, videos and web links can be used. To provide up-to-date information the author can connect a blog or news feed to his/her mashup. An author can choose between different Service components making it possible to provide information from Web 2.0 services like a Google map, a news feed or a Flickr photo gallery. Device components can be used to add contact features: like direct call, sending short messages, or email.

A private author can create a mashup to provide for example up-to-date information about his current vacation trip to his friends and family. The scenario with the mashup editor can be described as follows: The author creates a new mashup with different pages, one for news, one for pictures and a third one for a map. On the news page the author adds the “Twitter feed” component and configures it for his/her personal Twitter feed. With this component friends can receive message about the author’s current travel experiences. On the second page the author adds and configures a “Flickr photo gallery” component. This component is used to share the pictures the author has taken and uploaded to Flickr. On the map page the author adds the “Google map” component to visualize his/her current position on a map.
A business author can create a mashup to present his/her business and products to customers. The scenario for a business user can be described as follows: The author creates and layouts different pages, for example an introduction page, a news page and a contact page. On the introduction page the author adds Layout components like labels, pictures and videos to introduce his/her business. On the news page the author adds and configure the “Twitter feed” component for his/her business Twitter feed. With this component he can provide announcements and news about products, developments and specials. On the contact page he adds Layout components with his contact data and the Device components “Call” and “E-Mail” to provide direct contact feature to his/her customers. The author can also add and layout additional pages with advertisement for his products or services.

To make the mashup accessible the authors have to publish them with the “Publish” functionality of the mashup editor. Depending on whether the mashup has a private or business background, the users for this kind of mashups can be friends and relatives, or customers who will use the mobile application to receive general and up-to-date information.

3.1.2 Scenario 2 - Mobile learning with location aware content
A scenario in the area of mobile learning is a mobile mashup with location aware content. With this mashup it is possible to realize a treasure hunting game based on customized geo-tasks (Spikol et al., 2008).

The author of the mashup is a teacher who wants to create a geo-task game to teach his students in orienteering with mobile devices. The students are the players of the game. The mashup editor enables authors to create different, customized games as often as they want. To build a game, the author can define the game tasks and their instructions on different mashup pages. For the layout and to define the instruction of the game he/she can use Layout components like labels, images, videos, or a Service component for a map. With the build-in map feature the author can define the geo location where a task should appear to the player. He can also define instructions that are shown, when the player is not or not yet at the geo location of a task. The author can add different Service or Device components to provide data collection and feedback features. For data collection he/she can add a component to use the internal photo camera to take pictures or a component to record the GPS data. Feedbacks are possible by adding Service components for uploading taken pictures to a specified Flickr account, or sending messages in a Twitter or RSS feed.

The player will discover the defined instructions for a task when he enters the defined geo location with his mobile device. Depending on which feature the author decided to use, there are different ways to evaluate if the player fulfilled the tasks of the game. For this evaluation it is possible to use the data generated by the mobile device and the player.

One possibility to implement this scenario could be that the mashup records the GPS route of the players by using the data from the GPS sensor in the mobile device and storing them in a local database. Players could take pictures at the location of a task that will be automatically tagged with the current geo location and stored on the device. If the players have an Internet connection, they can give live feedbacks into the defined news feed service like RSS, or Twitter. The players can also upload taken pictures to an added and configured Flickr service.

Back in the classroom the students can review the game results by showing the recorded GPS route and geo tagged pictures on a map or showing the user generated data in the news feeds on a web site.
3.1.3 Scenario 3 - Record sensor data and environmental sources

A third scenario is based on a research project of the Center for Learning and Mobile Technologies (CeLeKT) at Linneaus University. The projects focus is on environmental research with the aim to “implement and research a new paradigm for fostering high school students learning in teams for environmental science” (Vogel et al., 2010).

One learning activity in this project is the Water quality activity that shall help students to learn by using new learning technologies. Students are working in teams to ascertain the water quality from several measurements in different water probes and at different pond locations. To derive the water quality a SPARK science learning system is used that provides sensor data for the different water quality indicators like dissolved oxygen, temperature, pH, and salt content. The student teams collect these data and take notes of the activity and results for their water probes. They also make pictures of the surrounding. Back in the covered area students write a report based on the collected data including their theories and reflections.

The mashup editor scenario in order to meet the requirements of this project, could be as follows: In the editor the author of the mashup defines and provides different pages with Service components that provide forms with input fields for the data the students have to collect in their activity, for example measurements from the sensors, results and notes. The author activates the mobile device features GPS recording, and camera for this mashup. With the GPS recording, the mashup can store the location where the students are taking the water probes. The camera feature is used to document the surrounding in pictures. The author connects a service to the mashup that will be used to receive the collected data from the different teams, for later visualization and evaluation of the results. A possible service to connect is the Pachube platform, a platform to store and share sensor and environment data.

The scenario for the student teams using the mashup on a mobile device could be presented as: Each team will have a mobile device with the environmental science mashup. Within the mashup, the team fills out the input fields with their collected data, notes and results. With the mobile device the participants take pictures of the surroundings that are automatically tagged with the current geo location. All the data including their current GPS location are stored in a local database.

Not as a part of the mashup, but as a feature of the mobile device, students can record their discussions with the build-in voice recorder to evaluate them later. Back in the classroom, the students can use the mashup to view the stored data for writing their reports. Was a service like Pachube connected, this service can be used to visualize the results. To do so each team can send their collected data over an Internet connection from the mashup to the service.

3.2 Stakeholders

The planned software system is flexible in the type of mashups that can be created with it. It provides a generic, extensible solution to create customized mashups. The described scenarios reveal that the system can be used in different application areas, (private, business or education) and a wide range of mobile computing. For these reasons, the stakeholders for the system are not limited to a special target group.
3.3 Software system specifications
Based on the scenario descriptions, the project idea and the technologies to use, the software system is specified as follows: it consists of two applications, an editor and a viewer. The editor shall allow the creation of multi-page mashups within a web application. This editor will be referred to as Mobile Mashup Editor (MME).

The viewer shall run on the mobile device as a cross-platform native application. The viewer shall deploy the mashups on the mobile device and provide the access to the device features and content. This viewer will be referred to as Mobile Mashup Viewer (MMV). The requirements analysis and use case modelling will be documented separately for the editor and the viewer.

Mashup description
As exemplified in the project idea in chapter 2 “State of the art”, mobile mashups are no standard web applications. They have to run as native applications on the mobile device to have access to the device sensor data and content. For that reason the Mobile Mashup Editor creates not a web application but rather a description of the created mashup. This description contains the mashup structure and settings, its pages, components and component settings.

The Mobile Mashup Viewer shall load the description and deploy the mobile application out of it. To describe the mobile mashup the Extensible Markup Language (XML) shall be used.

3.4 Requirements analysis - Mobile Mashup Editor
This section contains the requirements model for the prototype of the editor. The requirements are classified as functional and non-functional and will be presented in a narrative form. The detailed requirements catalogue is presented in Appendix A.

3.4.1 Functional requirements
The functional requirements define what the system shall do. For the editor the following functional requirements have been identified:

Create mashups
The editor shall allow creating mashups for mobile usage (FR-E1).

Mashup components
The editor shall contain different components an author can use to customize his mashup (FR-E2). These components are templates without any functionality. They are used to define the functionalities and layout of the mashup.

Mashup layout and content customization
The editor shall provide the possibility to create mashups with multiple pages and multiple components. To define the layout of components on a mashup page, their position shall be assignable and, depending on the component type, they shall be resizable (FR-E3).

Configuration
Depending on the component type, some components need additional configurations that the editor shall allow to set (FR-E4). For example a label component needs a text that shall be displayed or a service component needs login data that should be used for this service.
Publish mashup
The editor shall publish mashup descriptions for the access on a mobile device.

Graphical User Interface (GUI)
The editor shall work with the “What You See Is What You Get” (WYSIWYG) concept. The layout and functionalities the author defines in the GUI of the editor will be identical to the generated application on the mobile device (FR-E6).

3.4.2 Non-Functional Requirements
For the editor the following non-functional requirements have been identified:

Implementation
Following the technology evaluation in chapter 2 “State of the art”, the editor shall be implemented as a web application (NF-E1). The web development framework Google Web Toolkit shall be used for the implementation (NF-E2).

3.5 Use case modelling – Mobile Mashup Editor
As a “complementary way of eliciting and documenting requirements” (Arlow, J. et al., 2005) this section contains the artefacts of the use case modelling process for the editor. Each use case is described in a narrative form. As the results of the Unified Process activity “Detail a use case”, templates with the detailed specifications of the defined use cases are presented in Appendix B.

3.5.1 Use cases
Based on the identified functional requirements the following use cases can be derived:

Creation of mashups
The author can use the editor to create new mashups as often as he wants.

Defined Use case
- UC-E1: Create a new mobile mashup

Customization of the mashup
The author can customize the layout and content of mashups for his own needs. To fulfil the multiple pages requirement of the scenarios, he can add and delete pages. To define the content the author can add provided components to and delete components from the pages. To define the layout the author can position the component via drag-and-drop and resize them on the page.

Defined Use cases
- UC-E2: Add a new page to the mashup
- UC-E3: Deletes a page from the mashup
- UC-E4: Add a new component to a mashup page
- UC-E5: Select a component that is on a mashup page
- UC-E6: Delete a selected component from a mashup page
- UC-E7: Position a selected component
- UC-E8: Resize a component on the mashup page

Configuration of mashup components
The author can set and save properties for single mashup pages. The prototype will just contain simple page properties like the page name. In the Scenario 2 location awareness settings for pages are described, this are page properties that are planned for the future
development. For components that need additional configuration the author can set and save them.

*Defined Use cases*
- UC-E9: Configure a selected component
- UC-E10: Set the properties for the mashup pages

**Publish a mashup**
To make the mashup accessible for users on their mobile devices, the author can publish the mashup description to a web server.

*Defined Use case*
- UC-E11: Publish a mashup

### 3.5.2 Use cases model
The use case model shown in Figure 3.1 contains the basic use cases that will be realized in the prototype of the Mobile Mashup Editor. The Author, the identified actor who interacts directly with the system, uses the Mobile Mashup Editor to create new mobile mashups, to customize, configure and publish them. In a more product-oriented version of the system other use cases like Save mashup and Load mashup would be of course important. More details regarding future functionalities can be found in the section “Future work”.

*Figure 3.1 - Use case diagram (Mobile Mashup Editor)*

#### 3.6 Requirements analysis - Mobile Mashup Viewer
As presented in the editor requirements section, this section contains the requirements model for the prototype of the viewer. The requirements are separated by their classification as functional and non-functional requirements and will be presented in a narrative form. The detailed requirements catalogue is presented in Appendix A.
3.6.1 Functional Requirements
For the viewer the following functional requirements have been identified:

Provide access to published mashups
The viewer shall provide the user with a list containing all published mashups. (FR-V1).

Load a mashup
The viewer shall allow the user to select one of the published mashups (FR-V2) and load it into the viewer (FR-V3).

Generate mobile application
The viewer shall generate a mobile application with the layout and components as defined in the loaded mashup description (FR-V4). The viewer contains the functional implementation of the components that the editor offers as template components for a mashup. The functionalities a mobile application can have depend therefore on the components that are implemented in the viewer.

3.6.2 Non-Functional Requirements
According to the technology evaluation in chapter 2 “State of the art”, the viewer will be implemented as a cross-platform mobile application (NF-V1) with the framework Titanium Mobile (NF-V2). Currently this framework supports the platforms Android OS and iPhone OS, so the viewer shall work on both of these operating systems (NF-V3).

3.7 Use case modelling – Mobile Mashup Viewer
This section contains the artefacts of the use case modelling process for the viewer. Each use case is described in a narrative form. As the results of the Unified Process activity “Detail a use case”, templates with the detailed use case specifications are presented in Appendix B.

3.7.1 Use cases
Based on the identified functional requirements the following use cases can be derived:

Browse published mashups
The user can browse through a list of mashups that are currently published on the web server.
defined use case
- UC-V1: Browse through published mashups

Deploy mashup
The user can select a mashup from the list. This will load the description of the mashup from the web server into the viewer. Based on this description the viewer generates a mobile application and integrating all components with their defined layout and settings.
defined use case
- UC-V2: Deploy a published mashup

Use mobile application
The user can use the generated application like a native application within the viewer. The features and usage of the application depends on the mashup the authors created and on which components have been added to it. For that reason this use case is dynamic and cannot be defined in more details.
**Defined use case**
- UC-V3: Use mobile application

### 3.7.2 Use case model
The use case model shown in Figure 3.2 contains the basic use cases that will be realized in the prototype of the *Mobile Mashup Viewer*. The MashupUser, the identified actor who interacts directly with the system, utilizes the viewer to work with published mobile mashups as mobile applications.

![Use case diagram (Mobile Mashup Viewer)](image)

**Figure 3.2 - Use case diagram (Mobile Mashup Viewer)**

### 3.8 Operating requirements
To run the software system consisting of the editor and viewer the following hardware and software requirements have to be met.

#### 3.8.1 Hardware

**Editor**
A standard personal computer is needed as a web server to host the *Mobile Mashup Editor* web application. The server has to be connected to the Internet. To use the editor, an author needs a standard computer that is connected to the Internet.

**Viewer**
The *Mobile Mashup Viewer* requires a mobile device with the operating system *Android OS* or *iPhone OS*.

#### 3.8.2 Software

**Editor**
The editor is deployed in a Java Glassfish application server. GWT was successfully tested on a server with Linux, Ubuntu version 9.10 as its Operating System and Java Glassfish version v2.1.1, therefore these configurations are recommended.

On the client site the author needs one of the recommended web browsers with enabled JavaScript:
- Firefox 1.0, 1.5, 2.0 and 3.0
- Internet Explorer 6 and 7
- Safari 2 and 3
- Opera 9.0
**Viewer**
The viewer requires:
- IPhone OS version 3.0 or higher
- Android OS version 1.5 or higher
4 Software analysis and design

This chapter contains the documentation of the analysis workflow. The workflow is based on the requirements model that was created in the chapter 3 Software requirements. The result of the analysis is the Analysis model that describes what the software system needs to do.

Method
The software analysis approach follows the guidelines and recommendations of the Unified Process (UP) described in the book “UML 2 and the Unified Process” (Arlow, J. et al., 2005), chapter six to twelve.

4.1 OOA class diagram - Mobile Mashup Editor
The Object-oriented analysis (OOA) class diagram shows the business logic that was defined in the requirements and use cases. The noun/verb analysis is used on the software requirements artefacts to specify the class diagram shown in Figure 4.1:

Figure 4.1 - OOA class diagram (Mobile Mashup Editor)
In the prototype of the Mobile Mashup Editor the user can just work with one Mashup at a time. To ensure that the Mashup class has only one instance, the Design Pattern Singleton is used. MobileMashupEditor is the root class of the application. Its initGUI() operation creates the graphical user interface of the web application. A Mashup can contain one or more Pages with multiple Components. The composition association between the Mashup and the mashup Page means if the user deletes the Mashup all Pages will also be deleted. The composition association between the mashup Page and the Component means if the user deletes a Page also all the Components added to this Page will be deleted.

The abstract Component class realize the extensible part of the solution. New components will inherit from this class. The superclass contains all needed standard attributes and operations for the concrete component classes. In addition to that, this class defines the abstract, polymorphic operations initComponent(), configureComponent() and getComponentDescription(). initComponent() initializes the component when it is added to a mashup page. configureComponent() implements the logic that is needed to set the configuration for a concrete component. getComponentDescription() returns the description of the concrete component as a XML Element.

This description is used to create the XML description of the Mashup. The implementation of these abstract operations is done in the concrete component classes because only these classes know how to configure them and how to create their description. More information regarding the XML description can be found in section 4.6.3.

As samples, the diagram contains three concrete Component subclasses to show their association with the superclass. To keep the diagram clear not all concrete component classes that were defined in the requirements specification (FR-E2: Mashup components) of the prototype are shown here. The attributes and operations of a concrete component class depend on its requirements.

### 4.2 Scenario 1 – Mobile self-presentation

As mentioned before, the described Scenario 1 - Mobile self-presentation shall be realizable with the prototype implementation. For that reason this scenario is visualized as four activity diagrams, see Figure 4.2 to Figure 4.5, to show the realisation of the use cases. In this scenario the author creates a new mashup with four pages. The first page “Welcome” contains a label for introducing oneself to the user. The second page “My pictures” contains a personal Flickr feed where the author presents uploaded pictures from the Flickr service. The third page “News” contains a Twitter feed to provide news about oneself to the users.
Figure 4.2 - Activity diagram 1: Create a new mashup

Figure 4.3 - Activity diagram 2: Layout the "Welcome" page
Figure 4.4 - Activity diagram 3: Create and layout the "My pictures" - Flickr page
4.3 OOA class diagram - Mobile Mashup Viewer

The viewer generates the mobile application from the mashup as it was defined in the Mobile Mashup Editor. This means the mashup that will be deployed within the viewer application will have the same structure and layout as the mashup in the editor. That’s why the structure of the viewer OOA class diagram shown in Figure 4.6 is very similar to the one of the editor.
As seen in the editor, in the prototype of the Mobile Mashup Viewer the user can just work with one Mashup at a time. The root class of the component is the MobileMashupViewer class. This class is responsible for generating a list with all published mashups. It also loads the mashup description, for mashups that shall be deployed, from the server. Each concrete component contains a deployComponent() operation. The concrete components know how to deploy themselves within the mobile mashup application so that they are useable as full functional service, layout or device components.

As mentioned in the analysis of the editor, the abstract Components are the extensible part of the software system. The analysis of the concrete components is not part of the shown OOA class diagram.

4.4 Scenario - Deploy a mashup
The activity diagram shown in Figure 4.7 visualizes the scenario of deploying a mashup to generate a mobile application and the realisation of the use case UC-V2: Deploy mashup.
4.5 Conclusion requirements and analysis process

The requirement for a component-based, extensible software system was considered and realized within the requirements and analyse process. To realize the project idea and fulfil all requirements for the software system, the requirements and analyse process shows that two applications are necessary. (1) An editor web application to create, layout and publish the mashups and (2) a viewer application to generate mobile applications out of these mashups on the mobile device.

The editor contains only templates of the provided mashup components. These components can be configured with the settings that are needed in the running mobile application. The editor publishes a mashups as a XML description of its structure, content, layout and settings. This description is the interface between the editor and viewer. The viewer deploys the mobile application out of the XML description. The viewer contains the functional implementation of the mashup components.

4.6 Software design

This section documents the results of the software design process. This process is based on the results of the software analyse workflow and considers the programming languages and technologies that are used for the implementation of the software system.

4.6.1 Mobile Mashup Editor

The result of the object oriented design process is visualized in the Object-oriented design (OOD) class diagram shown in Figure 4.8.
The MobileMashupEditor class is the root class of the system and also the start class of the web application. For that reason this class has to implement the EntryPoint interface provided by the GWT framework. The method onModuleLoad() “serves as the real entry point to the application. The GWT compiler will start analyzing and eventually translating the application starting at this point” (Smeets et al., 2008).

The concrete mashup component classes are just visual components without any functionality. They are used to layout a mashup and to define its content. They appear as so-called widgets in the user-interface of the editor application. To visualize these classes they inherit from the GWT Widget class, the base class for user-interface objects.

Except for generating their description, based on their visual representation and the settings made in the ComponentSettingsDialog, these classes have no logic implementation. For that reason a separation in GUI and logic classes is not necessary.

Enhancements and changes compared to the analysis model
To allow the user the configuration of a mashup component a settings dialog is used. With the method createSettingsPanel(), each component creates its own settings panel. This panel is loaded into the ComponentSettingsDialog. The settings dialog calls the configureComponent() method to apply the settings in the concrete component.

The implementation of the methods selectComponent(), resizeComponent() and positionComponent() of the OOA model is not necessary. These functionalities are provided through GWT itself or external GWT libraries. The methods are removed from...
the design model. More information to the used external GWT libraries can be found in the chapter 5 “Implementation”.

The id and description attribute in the component is replaced with the Enumeration type ComponentType. This Enumeration implements the definition of all component types with their IDs, names and descriptions at one central point.

Server classes
The application needs the implementation of at least two server classes, a class to publish and to store the XML description of the mashup on the server and a class to upload resource files like audio, video, images that shall be used within a mashup to the server. More information about the server-site implementation can be found in chapter 5 “Implementation”.

Package structure
Figure 4.9 shows the package structure of the editor application. The client package is the root package for the client-site implementation of the application. The start class (MobileMashupEditor) is stored directly in the client package. The ui package contains all classes for the user-interface of the editor. The package mashup contains the implementation of the mashup components. Its under-package mashupcomponent is separated in the packages device, layout and service for storing the concrete mashup component widgets. The service package contains the client-site implementations for the communication with the server. The server package contains the server-site implementation of the application.
4.6.2 Mobile Mashup Viewer

The source code and classes of the mobile web application are stored in JavaScript files. To keep the class structure as defined in the OOA model, each class shall be stored in one JavaScript file. Two problems appear in this approach forced by the technologies:

1. When the application starts it has to load all the JavaScript files that it will use. Big files take longer to load and result in a slower application start.
2. The application makes a request for each JavaScript file. More files mean that more requests are necessary.

Therefore the source code shall be separated as follows. Classes that belonging logical together are stored in one JavaScript file. For example all layout components are stored in LayoutComponents.js or all components with Flickr functionalities are stored in FlickrComponents.js.

Figure 4.10 shows the resulting package structure of the viewer application, containing samples of each component type: layout, service and device components. The association between the classes are the same as in the OOA model and are not visualized in this figure.

Figure 4.10 – Package structure (Mobile Mashup Viewer)
4.6.3 Mobile Mashup description

To be able to deploy the mobile mashup as it was defined by the author, the mashup description has to contain all used mashup components and their settings including the mashup settings, pages and page settings. The sequence diagram shown in Figure 4.11 visualizes the function calls that are necessary to get the descriptions of all these mashup elements:

![Sequence diagram: Publish mashup](image)

Figure 4.11 - Sequence diagram: Publish mashup

Table 4.1 shows the vocabulary to describe a mashup.

<table>
<thead>
<tr>
<th>Mashup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mashup_id</td>
<td>A unique mashup ID is used as identifier for a mashup and to localize it on the web server. The name of the directory on the server is the mashup ID. This ID is created from the current time stamp and a random number.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>page_id</td>
<td>A consecutive page ID used to differentiate between the single mashup pages.</td>
</tr>
<tr>
<td>name</td>
<td>The name the author has defined for the page.</td>
</tr>
<tr>
<td>colour</td>
<td>The name of the colour the author has defined as background colour for the page.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The unique type name of the component is used to identify the component in the viewer application.</td>
</tr>
<tr>
<td>width</td>
<td>The width of the component in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>The height of the component in pixels.</td>
</tr>
<tr>
<td>top</td>
<td>The absolute top position of the component on the mashup page in pixels.</td>
</tr>
<tr>
<td>left</td>
<td>The absolute left position of the component on the mashup page in pixels.</td>
</tr>
<tr>
<td>[setting]*</td>
<td>Component specific settings like the url for an image, video or audio file or the html text of a label.</td>
</tr>
</tbody>
</table>

Table 4.1 - Vocabulary to describe a mashup
4.7 Interface design

**Mobile Mashup Editor**
The user-interface of the editor will be realised through the combination of the different panels classes as shown in Figure 4.12:

![Figure 4.12 - User-interface panel arrangement](image)

The class diagram shown in Figure 4.13 visualizes the additional classes for the implementation of the graphical user-interface user-interface an their associations.

![Figure 4.13 – User-interface class diagram (Mobile Mashup Editor)](image)

The `MashupPanel` represents the GUI class of the `Mashup` logic class. The GUI class is also implemented with the `Singleton` design pattern, to ensure this class has only one instance. Because a mashup can contain multi-pages, the user-interface represents a
tabbed set of these pages. One instance of the *MashupPageDockPanel* class represents one tab in this set and corresponds to one mashup page. The class contains another panel representing the real mashup page (*MashupPagePanel*) an author can layout and a panel for setting the properties of this page (*MashupPageSettingsPanel*). The panel *MashupAdministrationPanel* shall contain the access to the functionalities for creating a new and publishing a mashup, adding and deleting a page and selecting the mashup components for adding them to a page.

To support the concept and the realization of a WYSIWYG editor, the user interface to layout a mashup page shall appear to the author like the screen of a mobile device as shown in Figure 4.14. The size of the mashup page in the editor shall have the same resolution as the one of the mobile device display.

![Mashup page user-interface design](image)

Figure 4.14 - Mashup page user-interface design

The solution is targeted to end-users without programming knowhow. For that reason the editor application shall be intuitively operable. Self-descriptiveness of the application shall be offered through tooltips and descriptions within the user-interface. Error tolerance shall be realized through hint and warn messages that appear to the user.

**Mobile Mashup Viewer**

The viewer and the generated mobile applications shall look the same and shall be usable as usual native applications on these mobile devices. That way the user does not have to get used to new layouts, designs, new gesture and user input methods. For the prototype implementation the user interface is orientated to the layout and interaction of native iPhone applications.

The user interface of the viewer application consists of different views, the Published mashups view, the Detail description view and the Mashup view. If mashups were published they shall be presented to the user in a list as shown in Figure 4.15. The user can select a mashup from this list to see its detailed description. In the Detail description view, see Figure 4.16, the user can start the deployment of the mashup by selecting the “Deploy” button. When the mashup was successful deployed the generated mashup pages will be shown in the Mashup view. In this view the user can navigate through the different pages of the mashup by using the “Next” and “Back” button, see Figure 4.17.
5 Implementation

This chapter contains the description of the prototypical implementation. It starts with the description of the software system architecture. Subsequently, it shows the explanation of the different parts of the prototype implementation and finishes with the description of important classes and source codes. The implementation is based on the Scenario 1 - Mobile self-presentation.

5.1 Architecture – Mobile Mashup Editor

The Mobile Mashup Editor web application consists of a client and a server component. The client represents the graphical user interface that allows the author to create mashups and customize its content with the provided components. The client generates the XML descriptions of the mashup and publishes them on the server.

The server component (Mobile Mashup Server) provides the functionalities for storing the XML description on the server and for uploading resource files like audio, images and video files from the client that shall be used in a mashup. The remote database offers a mashup repository that enables users to search for and to subscribe for the published mashups. The defined architecture of the editor application is shown in Figure 5.1.

As it was not required for the proof of concept of the architecture, the prototype implementation does not contain a remote database. There is no need to explore the connection between a web application and a database within the thesis, because these are no new technologies and several working solutions like MySQL and Oracle already exists. To provide the subscription to published mashups, the prototype implementation of the editor will write the data of published mashups in a XML file. This XML file is used in the Mobile Mashup Viewer. Instead of doing a database query the viewer will load this file to generate the list of published mashups. Moreover, having an XML file facilitates the debugging process during the development.

![Figure 5.1 - Mobile Mashup Editor architecture](image)
5.2 Architecture – Mobile Mashup Viewer

The Mobile Mashup Viewer is implemented based on the cross-platform mobile framework that was chosen in the chapter 2 “State of the art”. As mentioned before the viewer contains the functional implementation of the different mashup components that the editor offers as template components. The different data sources that can be used within a mashup can be accessed through these components.

The Mobile Mashup Server can provide Server site functionalities, for example own service or database access implementations. The Web APIs and Server site functionalities are accessed through the Service components. Service components are also used to allow the user to create new content by entering personal data. Mobile device features, content and sensor data are accessible through the implemented Device components. Device components can access the mobile device operating system functionalities through the cross-platform mobile framework. The framework has integrated Source Adapters to map the calls from the Mobile Mashup Viewer application to the device OS language. A local database connected to the viewer can provide offline work possibilities to the users. The local database is not included in the prototype implementation. Figure 5.2 shows the defined architecture of the viewer application.

![Figure 5.2 - Mobile Mashup Viewer architecture](image)

5.3 Prototype implementation

The parts of the prototype implementation for the analysed and designed software system can be summarized as follows:

1. The Mobile Mashup Editor will be implemented as a web application using the Google Web Toolkit. This editor offers the possibility to create customized mobile mashups through the combination and configuration of different components within a graphical user interface. To publish these mashups the editor generates the XML description of the created mashup as defined in section 4.6.3 and stores it on a web server.
2. The Mobile Mashup Viewer will be implemented as a cross-platform mobile application using the Titanium Mobile framework. The viewer offers the possibility to deploy a mobile application out of a published mashup description.

**Expected results**
The prototype implementation shall prove that the technologies explored in the chapter 2 “State of the art” are working and that it is possible to implement the analysed and designed software system.

**Delimitation**
The focus of the prototype implementation is not on the general mashup concept to combine as many different Web API services as possible, but rather to create a basis, extensible software system. The design decision to implement and integrate Additional components into the solution will enable developers to enhance the features of it.

### 5.4 Mobile Mashup Editor

This section contains the description of the Mobile Mashup Editor implementation. It starts with the description of the GWT development environment followed by the description of important classes and implementations.

#### 5.4.1 Development environment

The GWT framework provides a plug-in for the integrated development environment (IDE) Eclipse. The plug-in provides a comfortable access to the tools and features of the framework. It allows the creation of GWT Web Application projects and the deployment to the Google App Engine server. The plug-in contains also a local App Engine server to deploy and test the implemented web application directly within the IDE. The debugging of the Java code behind the web application is also possible.

#### 5.4.2 MobileMashupEditor class

As mentioned in the section 4.6 "Software design", the MobileMashupEditor class is the starting class of the application. Its method onModuleLoad(), that is shown in Code extract 5.1, is called from the GWT compiler. This method generates the user-interface, composed of the different panel classes that were described in the software design of the editor. When the user-interface classes are initialized the classes Mashup and MashupPanel are instantiated to create and visualize a new mashup.

```java
/**
 * Entry point of the application.
 */

public void onModuleLoad() {
    // initialize application GUI
    initGUI();
    // create a new empty mashup
    newMashup();
}
```

*Code extract 5.1 – onModuleLoad Component creation with GWT deferred binding*

#### 5.4.3 MashupComponentTreeItem class

To add a new mashup component to a page, the user can select the different components in a Tree widget (see Figure 22). When the user selects a tree item a new instance of the selected component type shall be created and visualized on an initial position of the current selected mashup page. To realize this functionality an own tree item class has been implemented (MashupComponentTreeItem). The constructor of the class has a
parameter of type \textit{Component}, see abstract class in Figure 5.3. When the constructor is called an instance of the selected mashup component is saved in this parameter. To ensure that a user can use a component more then once, a new instance of this component will be created, with each click on the tree item. The new created instance will be then added to the current page, see Code extract 5.2.

\begin{verbatim}
public void onClick(ClickEvent event) {
    Component newComponent = null;
    if (mComponent != null) {
        // create a new instance of the concrete component
        newComponent = mComponent.getNewInstance();
        // add component to the current page
        Mashup.getInstance().getCurrentPage().addComponent(newComponent, addType);
    }
}
\end{verbatim}

Figure 5.3 - Screenshot mashup components tree

\section*{5.4.4 Server-site implementations}
The server-site implementation provides the functionalities for publishing the mashup description and uploading the resource files. A directory for every mashup is created on the server. All files belonging to a mashup are stored in this directory. The directory name is the mashup ID. The \textit{GWT-RPC} technology provides the functionalities for communication between the client and server. This technology is used to transfer the files to the server.

\subsection*{Publish mashup description}
To transfer the mashup description XML-stream from the client to the server the session is used. The XML stream, generated by the client, is written into the user session with a request from the client to the server. The request is done via a \textit{Remote procedure call} (RPC). The functionality is implemented in a class that has to inherit from the class \texttt{RemoteServiceServlet}. The Code extract 5.3 shows the server site implementation for writing the XML-stream into the session. The \texttt{HTTPServlet} shown in Code extract 5.4, is called via an \texttt{HTTPRequest}. Within the servlet the server reads the XML-stream from the session to create and store the xml file from it.

\begin{verbatim}
public class SessionUpdateImpl extends RemoteServiceServlet implements SessionUpdateService {
    public String sessionUpdateServer(String input) {
        // get current session
        HttpSession session = getThreadLocalRequest().getSession();
        // set session attribute
        session.setAttribute("xml", input);
        return "session updated with : " + input;
    }
}
\end{verbatim}

Code extract 5.3 – writing XML stream into the session

\begin{verbatim}
public class FileSaveImpl extends HttpServlet {
    /**
     * Post for publishing the mashup description XML on the server.
     * The XML stream is read from the session.
     */
    protected void doPost(HttpServletRequest request,
                           HttpServletResponse response) throws
\end{verbatim}
ServletException, IOException {
    String mashup_id = (String) request.getParameter("id");
    // get XML from Session
    HttpSession session = request.getSession();
    String xml = session.getAttribute("xml").toString();
    // creating XML file from xml string
    ...
} ...

Code extract 5.4 – Reading XML stream from the session

Upload resource files
To make images, audio and video files available in a mobile mashup, these resources have to be uploaded to the mashup directory on the server. On the client-site the user can select a local resource within the upload dialog shown in Figure 5.4.

![Upload file to mashup server](DSC00215.JPG)

Figure 5.4 - Screenshot Upload dialog

When the user selects the “Submit” Button in the upload dialog shown in Figure 5.4, the doPost(...) method of the HTTPServlet shown in Code extract 5.5 is called. Within the servlet the server reads the submitted file item. If the file item is not null, it will be stored in the directory with the name of the given mashup ID.

```java
public class FileUploadImpl extends HttpServlet {
    ... 
    /**
     * Post for uploading a mashup resource file.
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // set type for response object
        response.setContentType("text/plain");
        // read given mashup_id parameter - used for directory name
        String mashup_id = (String) request.getParameter("id");
        // get file item from request
        FileItem uploadItem = getFileItem(request);
        // check file item not null
        if (uploadItem == null) {
            response.getWriter().write("The given file is null.");
            return;
        }
        // get file content
        byte[] fileContent = uploadItem.get();
        ... 
        // save the file in the resource folder of the mashup
        PrintWriter saveFile = new PrintWriter(new File(relativePath +
            mashupDirectory + mashup_id + "/" + uploadItem.getName()));
        saveFile.write(new String(fileContent));
        saveFile.flush();
        saveFile.close();
        message = "File upload successful.";
    }
```

Code extract 5.5 – Store submitted resource file on the server
5.4.5 Layout
To separate the layout from the content the layout of the UI components is done with CSS. Each class that inherits from UIObject, the GWT user-interface super class, has a operation setStylename(String stylename) to set the style, which is defined in a CSS file. The file MobileMashupEditor.css contains the style definitions for the application. As an example the style definition of the background image for the mashup page, which shall be visualized as a mobile device is shown in Code extract 5.6. The Code extract 5.7 shows how the style definition is set in the source code of the MashupPanel.

```java
private MashupPanel() {
    //set style
    this.setStyleName("mashuppanel");
}
```

*Code extract 5.7 - Setting CSS style in source code*

5.4.6 External GWT source libraries and examples
As defined in the requirements the user shall have the possibility to position and resize components on a page. Resize and position via drag-and-drop of widgets are not part of the standard GWT implementation. The drag-and-drop feature is integrated by using a free open source library [1]. The resize feature is based on a free source code example [2].

5.4.7 Implementation of the user-interface
The user interface is implemented as defined in the software design process. Figure 5.5 shows a screenshot of the graphical user interface of the editor application. One additional panel was added to the MashupPageSettingsPanel. The panel with the title “Component” shows the description of a selected component and provides a fast access to delete it from the page.
5.5 Mobile Mashup Viewer

This section contains the description of the Mobile Mashup Viewer implementation. It starts with the description of the development environment and continues with the description of important parts of the implementation.

5.5.1 Development environment

The viewer is implemented as a web application with the technologies JavaScript, HTML, and CSS. For the implementation of the application the development tool Dashcode, which is part of the iPhone SDK, is used. Dashcode allows the implementation of web applications optimized for the usage on a mobile device like the iPhone and provides a GUI editor to design these applications. It is possible to run and debug the implemented applications within the iPhone emulator. Even if it is optimized for the iPhone the created application can be used on other mobile devices that have a JavaScript enabled web browser.

The Titanium Developer is the development tool that uses the Titanium Mobile cross-platform mobile framework to create native mobile applications. The implemented web application has to be imported into the Titanium Developer to compile it into native code for the iPhone and Android platform. The tool provides also the possibility to test these native applications on the device emulators and to build installation packages.
5.5.2 Start page of the mobile application

The page index.html is loaded when the application starts. All used JavaScript files are imported here. The `load()` method of the start class `MobileMashupViewer.js` is shown in Code extract 5.8. This method is called when the application has loaded all resources and is ready to start. It instantiates all parts of the application and calls the method `loadAllPublishedMashups()` to generate and visualize the list with all published mashups as described in the next section.

```javascript
/**
 * Load function.
 * Called by the index.html body-element's onload event when the
 * web application is ready to start.
 */
function load()
{
    // instantiate application parts
dashcode.setupParts();
    // visualize the list with all published mashups to the user
    loadAllPublishedMashups();
}
```

Code extract 5.8 - MobileMashupViewer.js - load() method

5.5.3 Get published mashups

As described in section 5.1 “Architecture – Mobile Mashup Editor”, the information about all published mashups are stored in an XML file (PublishedMashups.xml). An `XMLHttpRequest` is used to send a request for this file to the server. If the request is completed and the file was found, the server response returns the XML data as a `XMLDocument` object, see Code extract 5.9. On the client site, this document object will be parsed and the information about the published mashups will be provided to the user within a graphical List element, see Figure 6.10 and Figure 6.16.

```javascript
/**
 * Load published mashups XML and fill data in an array
 * that will be used for the GUI list element that shows all
 * published mashups
 */
function loadAllPublishedMashups()
{
    xmlHttp = new XMLHttpRequest();
    ...
    // load xml with information about all published mashups
    xmlHttp.open("GET", server_url + "PublishedMashups.xml", true);
    // define function for request was completed event (state=4)
    xmlHttp.onreadystatechange= function() {
        if(xmlHttp.readyState == 4) {
            var xmlDoc = xmlHttp.responseXML;
            // get all mashup elements
            var mashups = xmlDoc.documentElement.getElementsByTagName('mashup');
            // loop over all mashup elements
            for( var i=0; i < mashups.length; i++ ) {
                // fill array with mashup data
                mashupsArray[i] = new Array();
                mashupsArray[i]["id"] = mashups[i].getElementsByTagName(‘id’)[0].childNodes[0].nodeValue;
                ...
            }
            //reload list element for new published mashup information
            var list= document.getElementById(‘list’);
        }
    }
}
```

43
list.object.reloadData();
// show list with all published mashups
goToMashupList();
}
};
// send request
xmlHttp.send(null);
...

Code extract 5.9 – MobileMashupViewer.js – loadAllPublishedMashups()

5.5.4 Deploy mashups
To deploy a mashup the user selects a mashup in the published mashup list. The viewer application sends again an XMLHttpRequest to receive the XML Document object with the mashup description from the server. On the client site the document object will be parsed and the containing information will be saved in a multi-dimensional array with all mashup pages, components and their settings. In the deployment process shown in Code extract 5.10, the application loops over this array (pagesArray) to create the mashup Page elements. Within this loop another loop creates the mashup Components by using the method createMashupComponents(currComponentSettings) and adds them to the current page. In the end of the first loop the created Page element will be added to a page container that is used to visualize the pages within the mobile application.

/**
 * Mashup deployment.
 * Deploy pages and components from filled pagesArray.
 */
function deployMashup() {
  ...
  // get page container where the single pages will be added
  var pages = document.getElementById("mashupPages");

  // loop over all pages
  for( var i=0; i < pagesArray.length; i++ ) {
    // create a new Page component
    var currPage = new Page(pagesArray[i]["id"],
                            pagesArray[i]["colour"]);
    // add all components to page
    for( var j=0; j < pagesArray[i]["components"].length; j++ ) {
      //get all settings of the current component
      var currComponentSettings = pagesArray[i]["components"][j];
      // create component
      var createdComponent = createMashupComponent(currComponentSettings);
      //add component to the page
      if(createdComponent != null) {
        currPage.appendChild(createdComponent);
      }
    }
  }
  ...
  //add current page to the page container
  pages.appendChild(currPage);
}

Code extract 5.10 – MobileMashup.js – deployMashup()
5.6 Additional component development

Additional components are an important part of the solution in both the editor and the viewer. To offer new mashup components with requested functionalities two kinds of implementations are necessary. To provide new mashup components within the editor a GWT developer has to implement them in the Mobile Mashup Editor application. To provide the functionalities of this component in the mobile mashup application a Mobile framework developer has then to implement the functional component in the Mobile Mashup Viewer application.

To update of the Mobile Mashup Editor, the web application has to be deployed on the web server. Users have access to the new features when load the application next time into their web browser.

The Mobile Mashup Viewer can be updated over the procedure the mobile devices offer for update an installed application. In the iPhone’s case, the update can take place over the App Store. For Android phones, this can be done over the Android Market. Figure 5.6 visualises the additional component development workflow.

![Figure 5.6 - Additional component development workflow](image)

5.6.1 Implementation of a service component

As an example for the implementation of a mashup component, the implementation of a service component that shows a gallery of Flickr pictures for a given search term will be described in step by step within this section. The implementation starts with functional component in the Mobile Mashup Viewer. When this component is working, a template component that allows to add it to a mashup is implemented in the Mobile Mashup Editor.

**Mobile Mashup Viewer**

1. A new JavaScript file (`FlickrComponents.js`) in the service package to encapsulate all Flickr functionalities is created.

2. The methods for the creation of the `FlickrGalleryComponent` are added in the method `createMashupComponent()` that is called in the deployment process as described in section 5.5.4 “Deploy mashups”. The component will be created when the component type “FlickrGalleryComponent” was found in the loaded description of the mashup, see Code extract 5.11.
function creatMashupComponent(componentSettings) {
    ...
    if(componentSettings["type"] == "FlickrGalleryComponent") {
        // create mashup component with standard settings
        FlickrGalleryComponent.prototype = new
            MashupComponent(componentSettings);
        // create Flickr gallery component with specific settings
        var flickrGallery = new
            FlickrGalleryComponent(componentSettings);
        return flickrGallery;
    }
    ...
}

Code extract 5.11 – Creation of the FlickrGalleryComponent (MashupComponents.js)

3. The *FlickrGalleryComponent* class contains all the functionalities to create the picture gallery from the *Flickr* service for the configured search term. Code extract 5.12 shows the constructor of the class that returns the created and configured *Flickr* gallery as a component.

```javascript
// FlickrGallery component
function FlickrGalleryComponent(componentSettings) {
    // Call Flickr REST method for photosearch
    var feedURL =
        photos.search&api_key=APIKEY&text=" +
        componentSettings["searchterm"];
    // search for flickrgallery_pictures element to add pictures
    // within this element
    for(var i=0; i< this.component.children.length; i++) {
        if(this.component.children[i].id.search( 'flickrgallery_pictures') != -1) {
            // make a request to the flickr service and load
            // result in the element with the given id
            loadFlickrGalleryXMLHTTP(feedURL,
                this.component.children[i].id);
        }
    }
    return this.component;
}
```

Code extract 5.12 – Implementation of the FlickrGalleryComponent class

**Mobile Mashup Editor**

1. A new *Java* class with the name *FlickrGalleryComponent* in the service package is created. The class inherits from the abstract class *Component* and implements all abstract methods with the concrete functionalities for the *Flickr* component.

2. In the constructor of the class the defined type of the component is setted as shown in Code extract 5.13. The type is written in the component description and is used to identify the component in the viewer application.

```javascript
//set component type
setComponentType(ComponentType.FLICKR_GALLERY_COMPONENT);
```

Code extract 5.13 – Define component type

3. The component shall be visualized in the GUI as an example of a *Flickr* gallery as it would be presented on the mobile device, see Figure 6.7. For that reason an
image of a *Flickr* gallery is added to the *Widget* that represents the component in the GUI, see Code extract 5.14.

```java
public void initComponent() {
    ...
    // add Flickr gallery example image to the component widget
    mWidget.add(new Image("img/components/FlickrGalleryComponent.png"));
}
```

*Code extract 5.14 – Set image for component visualization in GUI*

4. Within the method `createSettingsPanel()` a panel with a `TextBox` for the definition of the search term is created, see Code extract 5.15.

```java
protected VerticalPanel createSettingsPanel() {
    VerticalPanel dialogContents = new VerticalPanel();
    dialogContents.setSpacing(4);
    // Create search term textbox
    Label title = new Label("Search term");
    mSearchTermTextBox = new TextBox();
    dialogContents.add(title);
    dialogContents.add(mSearchTermTextBox);
    return dialogContents;
}
```

*Code extract 5.15 – Create settings panel for the component*

5. The method `getComponentDescription()` of the class shown in Code extract 5.16 writes the standard component attributes for type, position and size in an XML Element. The method reads also the value from the search term input field and appends it on the XML Element.

```java
public Element getComponentDescription(Document descDocument) {
    //get standard attributes type, top, left, height, width
    Element flickGalleyComponent = super.getComponentDescription(descDocument);
    //Specific elements – defined search term
    Element searchTermElement = descDocument.createElement(AttributeType.SEARCH_TERM.elementName());
    Text value = descDocument.createTextNode(mSearchTermTextBox.getText());
    searchTermElement.appendChild(value);
    //Create XML structure
    flickGalleyComponent.appendChild(searchTermElement);
    return flickGalleyComponent;
}
```

*Code extract 5.16 – write component description into an XML Element type*

6. To allow the user to select the component in the GUI, a new tree item is added in the service component tree in the class *MashupAdministrationPanel*, see Code extract 5.17.

47
private Widget createServiceComponentWidget() {
    ...
    TreeItem flickrTree = new TreeItem("Flickr");
flickrTree.addItem(new MashupComponentTreeItem(img1, 
        "Flickr gallery", new FlickrGalleryComponent(), 
        AddToPageType.DRAGABLE_WITH_HEADER));
    ...
}

Code extract 5.17 - Create and add new tree item for component

The test of this and other implemented components is described in the chapter 6 “Exploring and testing the prototype”.

5.7 Deployment diagram

The deployment diagram in Figure 5.7 shows the hardware components of the software system and the software that is installed on it. On the left, the server side is located. The Mobile Mashup Server hosts the Mobile Mashup Editor web application. On the right side, on the top, a web browser client is presented. The client gets into connection with the Mobile Mashup Server to create new mashups through the Mobile Mashup Editor, which can be accessed with a web browser. Therefore, HTTP connection between these two components is required.

On the bottom right side, the mobile device is presented. The mobile device, making use of the Mobile Mashup Viewer to consume and deploy the mashups located on the Mobile Mashup Server. An HTTP connection is required between them to allow the consumption of the published mashups and the uploaded resource files.

![Deployment diagram of the software system](image)

Figure 5.7 - Deployment diagram of the software system
6 Exploring and testing the prototype

This chapter describes the test of the prototypical implementation. The test is based on the scenario description in section 3.1.1 and the described use cases. The proof-of-concept scenario for this test can be described as follows; an author creates a mashup to introduce the project he/she is currently working on. The mashup contains three pages. On the “Welcome” page, the first page of the mashup, the author adds text and an image to describe the project. The second page “Pictures” contains a Flickr image gallery to a search term that is related with the project. The third page “Contact” contains a Google map showing the authors address and the possibility to call the author directly. First the creation and publishing of the mashup with the Mobile Mashup Editor is tested. Following the Mobile Mashup Viewer is tested within the device emulators.

6.1 Mobile Mashup Editor

The author uses the “Page” menu shown in Figure 6.1 to create a mashup with three pages.

![Figure 6.1 - Screenshot Pages menu](image)

In the “Page settings” menu shown in Figure 6.2 the author defines the page name and the background colour.

![Figure 6.2 - Screenshot Pages settings](image)

On the first page of the mashup the author adds a text and an image by clicking on the respective Layout component. The selected component will be added to the current selected mashup page. Figure 6.3 shows the Layout components of the prototype.

![Figure 6.3 - Screenshot Layout components](image)

Components can be configured in a settings dialog that will be appear by double-click on the components header panel. This header panel is visualized in a blue colour to highlight it for the user, see Figure 6.4. The author can also drag the component by
clicking and this panel. The label component is configured with a rich text editor as shown in Figure 6.5.

The author can resize and position the components for his needs. Figure 6.6 shows the layout of the first page.

Figure 6.4 - Screenshot header pane    Figure 6.5 - Screenshot Label component configuration

Figure 6.6 - Screenshot Layout of the Welcome page
The *Flickr* gallery component is added to the second page. The author can define the search term for the gallery again by double-click on the component header panel. The configuration within the settings dialog is shown in Figure 6.7.

![Figure 6.7 - Screenshot Layout of the Pictures page and configuration of the “Flickr gallery” component](image)

The contact page contains a *Google* map component that shows the authors address on a map. To contact the author with a direct call, the *Device component* “Call number” is added and configured with the authors telephone number as shown in Figure 6.8.
Publish mashup

The author has the possibility to publish his mashup by clicking the “Publish” item in the “Mashup” menu. In a publish dialog (Figure 6.9) the author can add detail data for the created mashup to provide additional information to the user. A user can read this information before he/she deploys the mashup on the mobile device. When the author presses “Publish” the mashup description will be generated and stored on the web server and the XML file with the published mashups will be updated.
6.2 Mobile Mashup Viewer

The test of the viewer is done within the Titanium developer tool and the device emulators. First the application is tested within the iPhone emulator. After this the application is tested in the Android

**iPhone Test**

The following images are showing screenshots of the iPhone emulator making use of the Mobile Mashup Viewer to consume the created mashup. After starting the application the list with the names of all published mashups is loaded as shown in Figure 6.36. The user can select a mashup by clicking on it. When a mashup was selected, a new view is loaded that shows the detailed description of the mashup and provides a button for the deploying as shown in Figure 6.11. When the user selects the “Deploy” button the mashup will be deployed and shown in a new view. Figure 6.12 to Figure 6.14 show the deployed mashup with the three different pages. Figure 6.15 shows the function of the “Call number” component.
**Android test**
The following section shows in the Figures 6.16 to Figure 6.21 the screenshots of the same test within the *Android* emulator.

![Android test screenshots](image)

**Test results**
The test of prototype has proven that the chosen technologies are working and that the software system allows to create mashups in an end-user programming way. The prototype implementation fulfils the requirements that were defined and specified in the chapter 3 “Software requirements”. The screenshots of the tests within the device emulators prove that the mashup applications work cross-platform on devices with the *iPhone* or the *Android* platform. The screenshots of the *Android* test (see Figure 6.16 to Figure 6.21) show errors in the rendering of the buttons, these layout errors can be fixed by using images as buttons.
7 Conclusion

This chapter reflects upon the work that was done in the thesis and presents a summary of the advantages of the developed solution as well as the planned future works.

7.1 Reflections upon the work done

The thesis focuses on today’s mobile application development and mobile mashup domain. Initially this thesis identified limitations in mobile application development due to restrictions in the mobile phone hardware capabilities and a missing end-user programming solution. Furthermore the developers of mobile applications have to know different programming languages to provide their applications on the different mobile platforms, because the SDKs are tied to the particular mobile platforms. The drawback in the mobile mashup domain is the lack in possibilities to create mashups optimized for mobile usage. The goal of the thesis was to solve these drawbacks, by implementing an end-user programming software system to create and deploy mobile mashup applications. The software system will allow developers to create customized applications without knowhow in programming languages.

To accomplish this goal, I conducted first a literature study to find recent and suggested features for mobile applications, mobile mashups and end-user programming software systems. The project idea was developed from these identified features and represents the basis for the subsequent software development process. The requirements catalogue that was created within the requirements analysis process in chapter 3 responds to the question: Which are the requirements for the design and development of an end-user programming software system that supports the creation and deployment of cross-platform mobile mashups? The software analysis and design process in chapter 4 delivers the answer to the question: How should the architecture for such a system look like? A modular approach is a good solution for an architecture with those characteristics. Therefore, I have decided to create a Mobile Mashup Editor and a Mobile Mashup Viewer. The Mobile Mashup Editor is a web application that is hosted on the Mobile Mashup Server and usable with a standard JavaScript enabled web browsers. The Mobile Mashup Viewer is a cross-platform native application running on mobile devices. The viewer consumes the mashups offered by the Mobile Mashup Server and provides the capabilities for them to work cross-platform through their deployment within the viewer application.

Different technologies and frameworks were explored to be able to answer the question: What are the most suitable, currently available technologies to implement such a software system? In relation to the mashup editor I compared different frameworks for the development of Rich Internet Application. The Google Web Toolkit was chosen because it is platform independent and the created results can be presented through a web browser without the need of plug-ins, which will make the application easier to use than other solutions.

For the selection of a technology for a mobile mashup viewer, I performed a comparison study of existing mobile frameworks. The study had the selection criteria that the created mobile mashups applications shall work cross-platform, have the possibility to access the device features and content and can use Web 2.0 services. Cross-platform mobile frameworks allow the development of mobile applications that can fulfill all these requirements. The research in existing cross-platform mobile frameworks has shown that this market is growing and still evolving. The decision for the Titanium Mobile framework was based on the supported technologies, features, existing documentations and the roadmap for the further development. With the framework it was possible to implement a web application for deploying mobile
mashups having a high coverage of mobile functionalities and a promising future to cover other mobile device platforms.

The performed tests have proven the validity of the approach and the developed prototype. The system offers new ways and possibilities for the creation of mobile applications that will be accessible from several mobile platforms. Moreover, the end-user application, the Mobile Mashup Editor, has been shown as an easy to use development platform, as the developer can make use of visual computing concepts and the technique “What You See Is What You Get”.

7.2 Advantages of the solution

Most of the features that were suggested in the literature study, are integrated in the implemented solution. The following listing emphasizes the potential benefits of the software system and answers the question: What are the potential benefits of using this software system for mobile development?

1. The end-user programming software system makes it easy and comfortable for everybody to create and provide customized mobile applications.
2. The use of cross-platform technologies on the editor and viewer site does not tie the solution to particular platforms and devices.
3. The installed viewer application on the mobile device allows an easy access to the published mashups. So developers can provide them without the restrictions of an application store or licensing.
4. The solution is component-based and therefore extensible through the integration of additional components.
5. The generated mobile applications will have the layout and functionalities that were designed and can use the features of the mobile device like GPS, camera and the device content.
6. The customization feature realized through the combination of provided components does not limit the users to just one existing service, he/she may be able to combine different services like Flickr, Last.fm, Google maps on his/her personal needs in one mashup.

7.3 Future work

In the future development of the software system the architecture components remote database and local database shall be integrated. The remote database on the mashup server shall provide the mashup repository and a user management system. The local database in the viewer application shall provide offline work possibilities and synchronisation with the remote database.

The Mobile Mashup Editor shall provide the possibility to save and load created mashups. Another direction is the specification and development of Additional components that can be used in different ongoing research projects. One important feature for mobile mashups that shall be explored in the future work will be location awareness. Moreover the usability and user experience aspects of the proposed solution will be tested with end-users. Requirements regarding trust and privacy for the mobile mashup domain will be explored and shall be fulfilled in a product-oriented version of the software system.
8 References


Software references


APPENDIX A – Requirements catalogue

This appendix contains the requirements catalogue for the software components. It includes all requirements that were described in a narrative form in sections 3.4 and 3.6.

**Functional requirements – Mobile Mashup Editor**

<table>
<thead>
<tr>
<th>FR-E1</th>
<th>Mashup creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have</td>
</tr>
<tr>
<td>Requirements</td>
<td>The system shall allow the user to instantiate a new mashup (FR-E1.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-E1: Create new mashup</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FR-E2</th>
<th>Mashup components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have</td>
</tr>
</tbody>
</table>
| Requirements | The system shall provide specified layout, service, and device components that an author can add to his mashup.  
Provided *Layout components* are:  
- Audio (FR-E2.01)  
- Image (FR-E2.02)  
- Label (FR-E2.03)  
- Video (FR-E2.04)  
Provided *Service components* are:  
- Map (FR-E2.05)  
- Flickr gallery (FR-E2.06)  
- Twitter feed (FR-E2.07)  
Provided *Device components* are:  
- GPS (FR-E2.08)  
- Camera (FR-E2.09)  
- Call (FR-E2.10)  
- Short message (FR-E2.11)  
- E-Mail (FR-E2.12) |
| References | FR-E3: Mashup content and layout, UC-E6: Add component |

<table>
<thead>
<tr>
<th>FR-E3</th>
<th>Mashup layout and content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have</td>
</tr>
</tbody>
</table>
| Requirements | The system shall provide the following functionalities to define the content and layout of a mashup:  
- Add new pages to a mashup (FR-E3.01)  
- Delete pages from a mashup (FR-E3.02)  
- Add provided components to a mashup page (FR-E3.03)  
- Delete added components from a mashup page (FR-E3.04)  
- Free positioning of components within a mashup page (FR-E3.05)  
- Resizing of components within a mashup page (FR-E3.06) |
| References | UC-E2: Add page, UC-E3: Delete page, UC-E4: Add components, UC-E6: Delete components, UC-E7: Position component, UC-E8: Resize component |

<p>| FR-E4 | Component configuration |</p>
<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall allow the user to make additional setting for mashup pages and components that require them. (FR-E4.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-E4: Set page properties, UC-E9: Configure component</td>
</tr>
</tbody>
</table>

**FR-E5** Publish mashup

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall allow the user to publish mashup descriptions (FR-E5.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-E12: Publish mashup</td>
</tr>
</tbody>
</table>

**FR-E6** GUI – Graphical User Interface

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The editor shall work with the “What You See Is What You Get” (WYSIWAY) concept. (FR-E6.01).</td>
</tr>
<tr>
<td>References</td>
<td>Section 4.7 Interface design</td>
</tr>
</tbody>
</table>

**Non-functional requirements – Mobile Mashup Editor**

**NF-E1** Technology

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall be implemented as a web application (NF-E1.01).</td>
</tr>
<tr>
<td>References</td>
<td>Section 2.3 Web development frameworks</td>
</tr>
</tbody>
</table>

**NF-E2** Programming language and framework

<table>
<thead>
<tr>
<th>Priority</th>
<th>Should have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall be written in Java with the Google Web Toolkit framework (NF-E2.01).</td>
</tr>
<tr>
<td>References</td>
<td>Section 2.3.1 Framework decision</td>
</tr>
</tbody>
</table>

**Functional requirements – Mobile Mashup Viewer**

**FR-V1** Provide published mashups

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall provide all published mashups to the user. (FR-V1.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-V1: Browse mashups</td>
</tr>
</tbody>
</table>

**FR-V2** Select mashups

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall allow the user to select a mashup from the published mashups (FR-V2.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-V1: Browse mashups</td>
</tr>
</tbody>
</table>

**FR-V3** Load mashup

<table>
<thead>
<tr>
<th>Priority</th>
<th>Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>The system shall load the description of the selected mashup from the web server (FR-V3.01).</td>
</tr>
<tr>
<td>References</td>
<td>UC-V2: Deploy mashup</td>
</tr>
<tr>
<td>FR-V4</td>
<td>Generate mobile application</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Priority</td>
<td>Must have</td>
</tr>
<tr>
<td>Requirements</td>
<td>The system shall deploy a mobile application with the layout and components as defined in the loaded mashup description (FR-V4.01).</td>
</tr>
<tr>
<td>References</td>
<td>FR-V3: Load mashup, UC-V2: Deploy mashup</td>
</tr>
</tbody>
</table>

**Non-functional requirements – Mobile Mashup Viewer**

<table>
<thead>
<tr>
<th>NF-V1</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have</td>
</tr>
<tr>
<td>Requirements</td>
<td>The system shall be implemented as a cross-platform mobile application (NF-V1.01).</td>
</tr>
<tr>
<td>References</td>
<td>Section 2.4 Cross-platform mobile development, NF-V2: Development framework</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NF-V2</th>
<th>Development framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Should have</td>
</tr>
<tr>
<td>Requirements</td>
<td>The system shall be implemented with the cross-platform mobile framework <em>Titanium Mobile</em> (NF-V2.01).</td>
</tr>
<tr>
<td>References</td>
<td>Section 2.4.3 Framework decision, NF-V1: Technology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NF-V3</th>
<th>Supported devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Should have</td>
</tr>
<tr>
<td>Requirements</td>
<td>The system shall work on devices with the Android OS (NF-V3.01) or the iPhone OS (NF-V3.02).</td>
</tr>
<tr>
<td>References</td>
<td>Section 2.4.3 Framework decision</td>
</tr>
</tbody>
</table>
APPENDIX B – Use cases

This appendix contains the detailed use case specifications for the software components. It includes all use cases that were described in a narrative form in sections 3.5 and 3.7.

Use cases – Mobile Mashup Editor

<table>
<thead>
<tr>
<th>UC-E1</th>
<th>Use case: Create mashup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author creates a new mobile mashup.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>None.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. The use case starts when the Author loads the application in the web browser or when the Author selects “New” in the “Mashup” menu.  
2. If the Author selects “New”  
   2.1 The system removes the current mashup from the editor.  
3. The system creates a new mashup with one empty page (UC-E2: Add page).  
4. The system displays the new mashup. |
| **Postconditions** | 1. The system has removed the current mashup.  
2. The system has created a new mashup with one empty page.  
3. The system has displayed the new mashup. |

<table>
<thead>
<tr>
<th>UC-E2</th>
<th>Use case: Add page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author adds a new page to the mashup.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The system created a mashup.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. The use case starts when the Author selects “New page” in the “Page” menu.  
2. The system creates a new empty page.  
3. The system adds the new page at the end of the current mashup pages.  
4. The system displays the empty page. |
| **Postconditions** | 1. The system has created a new empty page.  
2. The system has displayed the new page. |

<table>
<thead>
<tr>
<th>UC-E3</th>
<th>Use case: Delete page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author deletes a page from the mashup.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The system created a mashup.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. The use case starts when the Author selects “Delete page” in the “Page” menu.  
2. If the mashup contains more than one page  
   2.1 The system deletes the current selected mashup page.  
   2.2 The system displays the last page of the mashup with its components and settings as current page.  
3. Else  
   3.1 The system tells the Author that a mashup must contain at least one mashup page. |
| **Postconditions** | The system has deleted the current selected mashup page. |
Alternative flow
1. The alternative flow starts with step 3.1 of the main flow.
2. The system has told an error message.

**UC-E4 Use case: Add component**

<table>
<thead>
<tr>
<th>Description</th>
<th>The Author adds a new component to a mashup page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Author</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The system created a mashup.</td>
</tr>
</tbody>
</table>
| Main flow   | 1. The use case starts when the Author selects a mashup component in the “Layout”, “Services”, or “Device” menu.  
2. The system displays the description for the added component.  
3. The system adds the selected component at an initial position on the current mashup page.  
4. The system displays the added mashup components on the current mashup page. |
| Postconditions | 1. The component has been added to the current mashup page.  
2. The system has displayed the selected component at the current mashup page. |

**UC-E5 Use case: Select component**

<table>
<thead>
<tr>
<th>Description</th>
<th>The Author selects a component that is on a mashup page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Author</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The mashup page contains at least one component.</td>
</tr>
</tbody>
</table>
| Main flow   | 1. The use case starts when the Author selects one of the components that are on the current mashup page.  
2. The system displays the component as selected.  
3. The system displays the description for the added component. |
| Postconditions | 1. The system has displayed the component description.  
2. The system has displayed the component as selected. |

**UC-E6 Use case: Delete component**

<table>
<thead>
<tr>
<th>Description</th>
<th>The Author deletes a selected component from a mashup page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Author</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The mashup page contains at least one component.</td>
</tr>
</tbody>
</table>
| Main flow   | 1. include (UC-E7 Select component)  
2. The Author deletes the component.  
3. The system deletes the component and removes it from the mashup page. |
| Postconditions | The component has been deleted from the current mashup page. |

**UC-E7 Use case: Position component**

<table>
<thead>
<tr>
<th>Description</th>
<th>The Author positions a selected component via-drag-and-drop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Author</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The mashup page contains at least one component.</td>
</tr>
</tbody>
</table>
| Main flow   | 1. include(UC-E7 Select component)  
2. The use case starts when the Author drags the component to any position on the mashup page.  
3. The Author drops the component on any position on the page.  
4. The system displays the component at the defined position. |
<p>| Postconditions | The system has displayed the component at the defined position. |</p>
<table>
<thead>
<tr>
<th>UC-E8</th>
<th>Use case: Resize component</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author resizes a component on the mashup page.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The component is defined as resizable.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. include(UC-E7 Select component)  
2. The use case starts when the Author selects the component-resize feature.  
3. The Author resizes the component.  
4. The system resizes the component to the defined size. |
| **Postconditions** | The system has resized the component to the defined size. |

<table>
<thead>
<tr>
<th>UC-E9</th>
<th>Use case: Configure component</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author configures a selected component.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The selected component is defined has configurable.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. include(UC-E7 Select component)  
2. The use case starts when the Author selects the component-configuration feature.  
3. The system displays the configuration-popup for the component.  
4. The Author enters the component settings.  
5. If the Author selects “Apply” in the configuration-popup  
5.1. The system updates the properties for the component.  
6. The Author closes the configuration-popup. |
| **Postconditions** | 1. The system has displayed the configuration-popup.  
2. The system has updated the properties for the component.  
3. The system has closed the configuration-popup. |

<table>
<thead>
<tr>
<th>UC-E10</th>
<th>Use case: Set page properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author sets the properties for the current selected page.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The system created a mashup.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. The use case starts when the Author changes one of the properties in the “Pages settings” menu.  
2. If the Author selects “Save” in the “Pages settings” panel  
2.1. The system updates the properties for the current selected page. |
| **Postconditions** | The system has updated the properties for the current selected page. |

<table>
<thead>
<tr>
<th>UC-E11</th>
<th>Use case: Publish mashup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The Author publishes a mashup description.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Author</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>The system created a mashup.</td>
</tr>
</tbody>
</table>
| **Main flow** | 1. The use case starts when the Author selects “Publish” in the “Mashup” menu.  
2. The system creates the mashup description.  
3. The system saves the mashup description on the web server. |
| **Postconditions** | 1. The system has created the mashup description.  
2. The system has saved the description on the web server. |
## Use cases – Mobile Mashup Viewer

### UC-V1 Use case: Browse mashups

<table>
<thead>
<tr>
<th>Description</th>
<th>The MashupUser browses through the published mashups.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>MashupUser</td>
</tr>
<tr>
<td>Preconditions</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Main flow**

1. The use case starts when the MashupUser selects “All Mashups”.
2. The system searches for all published mashups.
3. If the system finds published mashups
   3.1. The system displays a list of all published mashups.
4. Else
   4.1. The system tells the MashupUser that no published mashups could be found.

**Postconditions**

The system has displayed a list with all published mashups.

**Alternative flow**

1. The alternative flow starts with step 4.1 of the main flow.
2. The system has told an error message.

### UC-V2 Deploy mashup

<table>
<thead>
<tr>
<th>Description</th>
<th>The MashupUser deploys a published mashup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>MashupUser</td>
</tr>
<tr>
<td>Preconditions</td>
<td>At least one mashup were published with the Mobile Mashup Editor.</td>
</tr>
</tbody>
</table>

**Main flow**

1. The use case starts when the Author selects a mashup in the published-mashups list.
2. The system loads the description of the selected mashup from the web server.
3. The system generates a mobile application with the layout and components as defined in the description of the mashup.
4. If the generation of the mobile application was successful
   4.1. The system displays the generated mobile application.
5. Else
   5.1. The system tells the MashupUser that the generation of the mobile application was not successful.

**Postconditions**

The system displays a mobile application with the layout and functionalities as defined in the mashup description.

**Alternative flow**

1. The alternative flow starts with step 5.1 of the main flow.
2. The system has told an error message.

### UC-V3 Use case: Use mobile application

<table>
<thead>
<tr>
<th>Description</th>
<th>The MashupUser uses a generated mobile application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>MashupUser</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The generation of the mobile application was successful.</td>
</tr>
</tbody>
</table>

**Main flow**

1. The use case starts when the system displays the mobile application.
2. The author can use the mobile application as defined in the description of the mashup.

**Postconditions**

None.
APPENDIX C – Cross-platform mobile frameworks

This appendix contains detailed information about the cross-platform mobile frameworks PhoneGap, Rhomobile and Titanium Mobile. Used technologies, the license model, supported devices and features are documented for each of these frameworks.

1 PhoneGap

Web site
http://phonegap.com

Technologies and supported languages
Standard web technologies / web applications using:
- JavaScript
- HTML
- CSS

License
Open Source

Roadmap
http://phonegap.pbworks.com/Roadmap

Supported devices

<table>
<thead>
<tr>
<th>Device*</th>
<th>Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>Y</td>
</tr>
<tr>
<td>Android</td>
<td>Y</td>
</tr>
<tr>
<td>Blackberry</td>
<td>N</td>
</tr>
<tr>
<td>Symbian</td>
<td>N</td>
</tr>
<tr>
<td>Palm</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 8.1 - PhoneGap supported devices

Legend: Y - framework tested for device N - framework not tested for device
* not supported yet, but on the road map: Windows Mobile

Supported features

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone</th>
<th>Android</th>
<th>Blackberry</th>
<th>Symbian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Camera</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Contact support</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Geo Location</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Local DB</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Sound</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Vibration</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 8.2 - PhoneGap accessible device features

Legend: Y - feature supported N - feature not supported
Pro
- Supports Dashcode Projects (Apple’s mobile web application development Tool).

Contra
- Rare documentation, tutorials and examples
- Development needs work through the source code.
- Supported device features still under development and work not stable.
- Application development in two different projects: (1) setup XCode project for iPhone development, (2) setup Eclipse project for Android development.

3 Titanium Mobile

Web site
www.appcelerator.com/products/titanium-mobile

Technologies and supported languages
Standard web technologies (web applications) using:
- HTML
- CSS
- JavaScript
- based on WebKit standard

License
Open source

Roadmap
http://www.codestrong.com/timobile/roadmap

Supported devices

<table>
<thead>
<tr>
<th>Device*</th>
<th>Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>Y</td>
</tr>
<tr>
<td>Android</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 8.3 - Titanium Mobile supported devices

Legend: Y - framework tested for device  N - framework not tested for device
* not supported yet, but on the road map: RIM Blackberry, MS Windows Mobile, Symbian and Palm Pre
Supports features

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone</th>
<th>Tested</th>
<th>Android</th>
<th>Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>Y</td>
<td>*</td>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>Camera</td>
<td>Y</td>
<td>*</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Contact support</td>
<td>Y</td>
<td>*</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Geo Location</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>Local DB</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Push Notification</td>
<td>Y</td>
<td>X</td>
<td>N</td>
<td>X</td>
</tr>
<tr>
<td>Photo Gallery</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Sound</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Video</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Vibration</td>
<td>Y</td>
<td>*</td>
<td>Y</td>
<td>*</td>
</tr>
</tbody>
</table>

Table 8.4 - Titanium Mobile accessible device features

Legend: Y - feature supported   N - feature not supported   X – feature was successfully tested
* just testable on the real device, not with the emulator

Pro
- The framework provides a development tool to create new mobile application projects and to deploy and test them within the different devices emulators.
- Just one application project for all devices.
- Good documentation, tutorials, developer forum and sample applications.

Contra
- Supports for now just iPhone and Android.

Additional information
Forum: http://support.appcelerator.net

2 Rhomobile

Web site
http://www.rhomobile.com/home

Technologies and supported languages
Standard web technologies / web applications using:
- HTML and Ruby
- JavaScript (AJAX does not work on RIM and Symbian)
- CSS

License
Dual license:
- Offers GPL to open source applications
- Commercial license for everyone else
Roadmap
http://wiki.rhomobile.com/index.php/Rhodes#Device_Capabilities_/2F_Native_UI_Ele-
ments

Supported devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>N</td>
</tr>
<tr>
<td>Android</td>
<td>Y</td>
</tr>
<tr>
<td>Symbian</td>
<td>N</td>
</tr>
<tr>
<td>BlackBerry</td>
<td>N</td>
</tr>
<tr>
<td>Windows Mobile</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 8.5 - Rhomobile supported devices

Legend: Y - framework tested for device  N - framework not tested for device

Supports Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone</th>
<th>Android</th>
<th>Symbian</th>
<th>Blackberry</th>
<th>Windows Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Geo Location</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PIM contacts</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Local DB</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Push Notification</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 8.6 - Rhomobile accessible device features

Legend: Y - feature supported  N - feature not supported

Pro
- Sync functionality with a server.
- Push Notification for iPhone and Blackberry.
- Supports almost all devices.

Contra
- Needs Ruby knowhow.
- No multimedia features supported on the device.

Additional information
Wiki: http://wiki.rhomobile.com
Forum: http://groups.google.com/group/rhomobile