



Università degli Studi di Padova
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

**Tools and measurements for the products and
complex services quality in mobility
collaboration environments**

Relatore: Ch.mo Prof. Matteo Bertocco
Correlatore: Ing. Mauro Franchin

Laureando: Andrea Veronese

ANNO ACCADEMICO 2011 - 2012

Contents

1	Introduction	1
2	Context and Objectives	3
2.1	Company Context	3
2.1.1	Mantis Bug Tracker	4
2.1.2	MantisBT mobile interaction	7
2.2	Objectives	8
3	Environment Analysis	9
3.1	Cognitive Phase	9
3.1.1	Company Control System	10
3.1.2	Specific usages	12
3.1.3	MantisBT webservice	14
3.1.4	SOAP interface	17
3.2	Operative Phase	22
3.2.1	Native, web and hybrid approaches	23
3.2.2	Comparing the different approaches	29
3.2.3	Choosing the right approach	32
4	Development Environment	35
4.1	PhoneGap	35
4.1.1	Web Browser	36
4.1.2	Features	38
4.2	jQuery Mobile	39
4.3	Setting the environment	40
4.3.1	Local Development Environment	41
5	Software Design	47
5.1	General Architecture	47
5.1.1	Evaluation System	48
5.1.2	Project Structure	49

CONTENTS

5.2	Mobile Component (<i>App</i>)	53
5.2.1	Functional Analysis	53
5.2.2	Technical Analysis	56
5.3	Server Component	60
5.3.1	Functional Analysis	60
5.3.2	Technical Analysis	62
6	Final Considerations	65
A	Graphic Details	67
A.1	Login and Home pages	67
A.2	Resolved Tickets pages	70
A.2.1	Evaluate	72
A.2.2	Close	73
A.3	New Tickets pages	74
A.3.1	Assign	75
	Bibliography	77
	List of Tables	77
	List of Figures	78

Chapter 1

Introduction

This Master thesis work started when a personal interest met a real need in a local company, Mida Solutions (located in Padua, Italy). After the studies of Quality Engineering, a collaboration proposal - in line with the Quality theory for the companies and involving a software design and development component - permitted to focus our attention on the real need of the company.

From the beginning the scope of the proposal was very clear: to work on a project, strongly focused on products and services quality. The work was developed within an existing environment where the attention to Quality had become an actual daily *mission*.

Two are the main points of interest of this work: This work concentrates on two main point of interest

- the project nature, which is strongly oriented to the Quality theory;
- an innovative software design and development approach, which made it possible to obtain a cross-platform solution.

As regards the first point, all the work was driven by the volition of creating a tool closely connected with the company environment. The company had recently obtained the International Quality Certification, partly thanks to the adoption of a specific system, whose main task is to keep track of all the company activities. This system is still being used and allows the company to keep under control the set of tasks and activities which happen in and for the company every day. This system is perfectly in line with Quality standards: the developed tool fits this situation just because a strong interaction with such a system was required.

As has already been mentioned, the approach adopted for both the design and the development phases represents a point of interest. In par-

ticular, the hybrid application approach acts as a “happy medium” between two completely different approaches, the native and the web approaches.

As usually happens with hybrid solutions, it tries to capture the strength points from both the different approaches.

Chapter 2

Context and Objectives

This chapter provides a first analysis of the context where the thesis work was developed; this information is useful in order to understand the company environment and the motivations related to the project. In particular, there are interesting aspects of the quality certification - recently obtained by the company - that are closely connected with the requirement of such a project.

As regards the company context, a major role in this work was covered by the company management system which is being used to trace all the activities that take place in and for the company environment. This system was adapted for the purpose of activity tracing; starting from a pre-existent tool which had been created to track software defects during the software development process, a simple customization allowed the system to be a real and effective activity tracking tool.

Consequently, the thesis objectives were outlined within this context: the second part of the chapter will explain the two main objectives of the project. The first one is related to the company interest in designing and developing a mobile application which is able to interact with the aforesaid company management system installation. The second objective arises from the interest of providing an evaluation tool capable of improving the company tasks assignment strategy of human resources.

2.1 Company Context

The company environment where the thesis project was designed and developed is Mida Solutions S.r.l. . The company was founded in 2004, by a team of telecommunication experts, with the mission to provide added value innovative technologies for communication. Mida Solutions provides

expertise and a complete suite of Voice Applications and Added Value Services, with the goal of improving the Telephony Infrastructure functions.

In March 2012 Mida Solutions Quality System was officially certified by DNV¹ in accordance with ISO 9001: the certification was released according to international standards. This step aims to assure quality and reliability for both their products and their partners.

One of the most important roles that contributed to Quality certification was covered by a specific management tool that had been adopted in order to track all the company activities. This system was born on an existing software whose common use is to track software defects: the software is called Mantis Bug Tracker (hereinafter referred to as MantisBT).

The following description illustrates the structure of the system MantisBT considering all the settings that were applied for the adaptation to the company environment.

2.1.1 Mantis Bug Tracker

Mantis Bug Tracker is a web-based bug tracking system that was first made available to the public in November 2000. It is a free and open source system released under the terms of the GNU General Public License version 2 [?]. The most common use of MantisBT is to track software defects. It is often configured by users to serve as a more generic tracking system and project management tool. MantisBT is written in the PHP scripting language and works with MySQL, MS SQL, and PostgreSQL databases and a webserver. MantisBT has been installed on Windows, Linux, MacOS, OS/2, and others. It is released under the terms of the GNU General Public License (GPL)².

MantisBT provides a web-based interface which allows the user interaction; a special administration section allows the administrator user to make the appropriate customizations. It is important to underline that MantisBT uses access levels to define what a user can do. Each user account has a global or default access level associated with it. The default access levels supplied as standard with MantisBT are:

- viewer
- reporter

¹DNV (Det Norske Veritas) is an independent foundation with the purpose of safeguarding life, property, and the environment.

²<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

- updater
- developer
- manager
- administrator

Each feature has several configuration options associated with it and identifies the required access level to perform certain actions.

Customization

When the company Mida Solutions identified MantisBT as a strong and useful tool for the above-mentioned purpose, some customizations were applied. The most significant changes were made in the system nomenclature: it is important to underline that the structure of the tool was preserved, and the customization only affected the system strings in order to adapt the tool to the company reality and needs. All the changes applied are listed in Table 2.1.

Old Nomenclature	New Nomenclature
Bug (or Issue)	Ticket
Project	Group
Category	End Customer
Reporter	Poster
Due Date	Alert Date

Table 2.1: Nomenclature changes adopted

Thus, from now on the new nomenclature is adopted.

MantisBT statuses

The basic entity of MantisBT is a *ticket* which represents a generic activity of the company. Every ticket belongs to a specific *group* which could be nested in a father group. An important part of tickets tracking is to classify them as per their *status*. Each company team may decide to have a different set of categorizations for the status of the ticket, and hence, MantisBT enables the user to customize the list of statuses. MantisBT assumes that a ticket can be in one of three stages:

- opened

- resolved
- closed

Hence, the customized statuses list will be mapped in these three stages. For example, MantisBT comes out of the box with the following statuses: new, feedback, acknowledged, confirmed, assigned, resolved and closed. In this case “new” -> “assigned” map in opened, “resolved” means resolved and “closed” means closed.

Below is the explanation of the meaning of each standard status that are shipped with MantisBT (see Figure 2.1 for an explanatory representation).

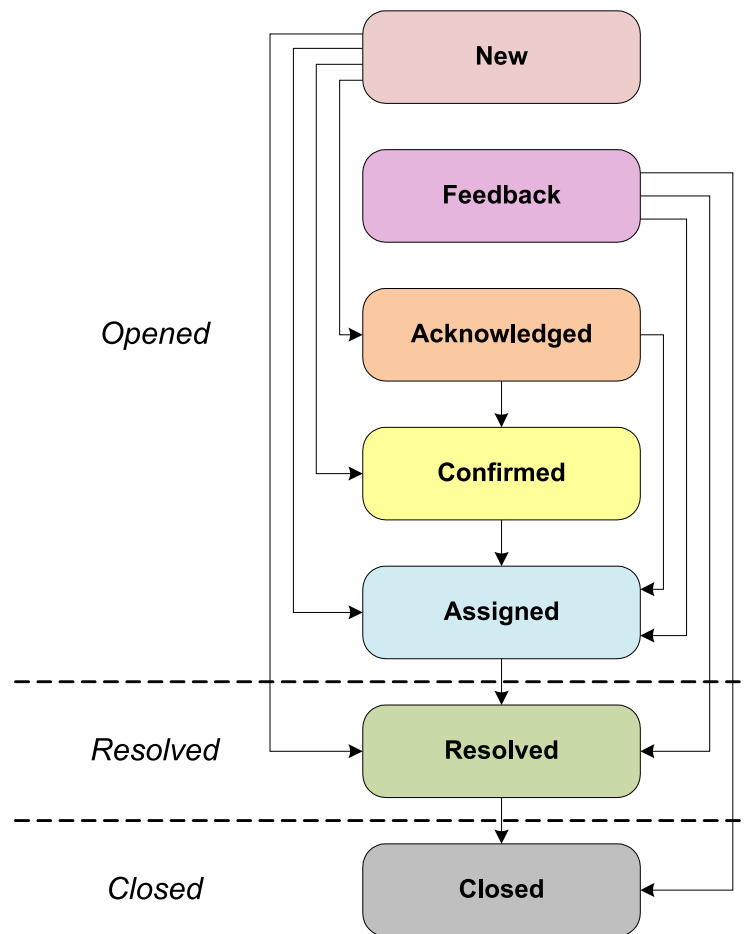


Figure 2.1: Diagram of the status transitions.

New This is the landing status for new tickets. Tickets stay in this status until they are assigned, acknowledged, confirmed or resolved. The

next status can be “acknowledged”, “confirmed”, “assigned” or “resolved”.

Feedback This status is used when feedback is required from the ticket poster. This status is used when a ticket doesn’t follow the standard path: the ticket that has been closed must be reopened. The ticket can then be moved to “assigned”, “resolved” or “closed”.

Acknowledged This status is used by the development team to reflect their agreement to the suggested feature request, or to agree with what the poster is suggesting in an ticket post, although they haven’t yet attempted to reproduce what the poster refers to. The next status is typically “assigned” or “confirmed”.

Confirmed This status is typically used by the development team to mention that they agree with what the poster is suggesting in the ticket and that they have confirmed and reproduced the ticket. The next status is typically “assigned”.

Assigned This status is used to reflect that the ticket has been assigned to one of the team members and that such team member is actively working on the ticket. The next status is typically “resolved”.

Resolved This status is used to reflect that the ticket has been resolved. A ticket can be resolved with one of many resolutions (customizable). The next statuses are typically “closed” or in case of the ticket being re-opened, then it would be “feedback”.

Closed This status reflects that the ticket is completely closed and no further actions are required on it.

A ticket is represented by several fields which define all its features: this possibility allows the user to provide a detailed description of every activity which evolves inside the company.

2.1.2 MantisBT mobile interaction

As I mentioned above, one of the main interests of Mida Solutions for this thesis project was to design and develop a tool that interacts with the company installation of MantisBT. All the discussion of this work starts from a key feature that is present in the MantisBT project: the SOAP interface. MantisBT is equipped of a SOAP webservice interface which

provides an easy way to connect to a Mantis installation. All the technical details are particularly analyzed in Subsection 3.1.4, however it's now important to specify that SOAP³ [?] is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

The MantisBT SOAP interface covers a key role in the communication of the developed application to the server component.

2.2 Objectives

The two main objectives that have inspired all the thesis work, evolved from a personal interest in the mobile application development which found a fitting opportunity in the company needs. Since MantisBT is a web-based system, it turns out to be inconvenient for a mobile experience, in terms of - one for all - ease of use. It is right here that the company's interest in a mobile tool finds its place: not only a simple system adaptation for the resized mobile screens, but a useful and practical tool for real interactions, in terms of read and write operations. During the first steps of collaboration, an interesting case came to the fore: the point is related to the modality of task assignment to different users. The main question is in fact: what is the right way - I would say, the *best* way - to assign a certain activity to a user. Is it possible to integrate a user evaluation system in the existing activity management company system? This question led to an interesting analysis in order to develop an evaluation system of the MantisBT tickets, aiming to improve the assignment criteria and, ultimately the users efficiency, which is strongly related to customer satisfaction.

³originally defined as Simple Object Access Protocol

Chapter 3

Environment Analysis

In this chapter the environment analysis is conducted through two distinct phases. This choice traces the Master thesis work experience: at the beginning, in effect, a deep analysis of the company environment was carried out. This analysis is defined in terms of study of the company's technological equipment: a major role was played by the system MantisBT which the thesis project interacts with. In particular, all the most usual interaction operations were traced: this study makes it possible to understand the specific usages of the company control system. This first step was named "cognitive phase": this includes a detailed description of the MantisBT structure - in conjunction with the usual operations - and a description of the MantisBT SOAP interface.

The second passage, which is referred to us with "operative phase", widely illustrates the study of the development environment choice. This phase occupied a prominent part of the experience since this choice characterized the following phases. All the analysis consists of an ample comparison between three possibilities for the application development. The process of choosing a development approach for the mobile application, namely native, web or hybrid, involves many parameters, such as budget, project timeframe, target audience and app functionality to name but a few. Each approach carries inherent benefits and limitations, and finding the one that best addresses the company's needs was a challenging task.

3.1 Cognitive Phase

As I have already mentioned, attention here focuses on the company control tool. At a later time the detailed analysis of the SOAP interface is carried out.

3.1.1 Company Control System

MantisBT is a web based bug tracking system which has become one of the most popular open source bug/issue tracking systems. MantisBT is developed in PHP, with support to multiple database backends including MySQL, MS SQL, PostgreSQL and DB2 (see the logo in Figure 3.1).



Figure 3.1: MantisBT logo (<http://www.mantisbt.org>).

The Activity Tracking System - based on MantisBT - provides an interactive web based platform for activities tracking. This generic process of activity tracking - which is represented by a *ticket* - contains the following information:

ID Unique ID of the ticket.

Description Detailed description of the ticket including what, where, why, how and when the ticket occurs.

Summary Summary of the ticket, it acts as a ticket title.

Group Specifies the company group which the ticket belongs to.

End Customer Specifies the final customer which the ticket is intended for.

Priority Priority is assigned for its urgency.

Severity Specifies its impact on the system.

Status Current status of the ticket.

Poster Information of the person already registered with the system who posts the ticket.

Handler The ticket poster may also assign ticket to specific user.

History Shows the historical ticket changes.

Notes Any other information that would be helpful in identifying the ticket.

Submitted Date Records the date when the submission has been performed.

Last Update Records the date of the last update action.

Due Date If present, records the date when the ticket expires, i.e. the deadline.

Relationships Indicates all the other tickets which the ticket is related to.

Tags All the words that would be helpful in identifying the ticket.

All this information is present both when a ticket has to be submitted in a group - the mandatory fields are the End Customer, the Summary and the Description - and when a ticket is displayed.

It is now important to remark that each user account has a global or default access level associated with it. This access level is used as the access level for such users for all actions associated with public groups as well as actions that are not related to a specific group. Users with global access level lower than a specific private group threshold will not have access to private groups by default.

The default access levels of the company control system are

- viewer
- poster
- updater
- developer
- manager
- administrator

Each feature has several configuration options associated with it and identifies the required access level to do certain actions. For example, viewing a ticket, posting a ticket, updating a ticket, adding a note, etc.

For example, in the case of posting tickets, the required access level is configurable using the specific posting ticket threshold configuration option (which is defaulted to “poster”). So for a user to be able to post a ticket against a public group, the user must have a group-specific or a global access level that is greater than or equal to “poster”. However, in the

case of posting a ticket against a private group, the user must have group specific access level (that is explicitly granted against the group) that is higher than “poster” or have a global access level that is higher than both the private group threshold and posting ticket threshold.

Group specific access levels override the global access levels. For example, a user may have “poster” as the global access level, but have a “manager” access level to a specific group. Or a user may have “manager” as the global access level by “viewer” access to a specific group. Access levels can be overridden for both public and private groups. However, overriding access level is not allowed for users with global access “administrator”.

Each feature typically has multiple access control configuration options to define what access level can do certain operations. For example, adding a note may require “poster” access level, updating a note may require “developer” access level, unless the note was owned by the same user and in this case “poster” access level. Such threshold configuration options can be set to a single access level, which means users with such threshold and above are authorized to do such action. The other option is to specify an array of access levels which indicates that users with the explicitly specific thresholds are allowed to do such actions.

It is also worth mentioning that the access levels are defined by a specific enumeration configuration option, and it is possible to customize such list.

3.1.2 Specific usages

The main page of the system reports all the tickets that the logged user is allowed to view, which depends on the access level of the user. This page is structured in two main sections: the upper one contains the filter selection. A filter allows a user to select specific tickets depending on the selected values. This is a useful opportunity, since the company system contains thousands of different tickets. The second section shows a list of the selected tickets: if no filter is selected all the tickets are displayed by last update in descending order. A dedicated page allows a user to post a ticket: on this page all the details described in Subsection 3.1.1 can be inserted. Finally the system includes a wide administration section, which can be accessed only by users with administrator or manager access level. Here it is possible to manage the groups, the end customers, the system settings and the custom fields that could be added to a ticket specification. In particular, as regards this last detail, MantisBT offers the possibility for managers and administrators to define custom fields as a way to extend MantisBT to deal with information that is specific to a group. These are the main features of custom fields:

- custom fields are defined system wide,
- custom fields can be linked to multiple groups,
- custom fields must be defined by users with access level “administrator”,
- custom fields can be linked to groups by users with access level “manager” or above (by default, this can be configurable),
- number of custom fields is not restricted,
- users can define filters that include custom fields,
- enumeration custom fields can have a set of static values or values that are calculated dynamically based on a custom function.

The possible procedures that are usually performed by different users of the control system are now presented: this point could be useful in order to understand what are the usual actions related to a specific user. This also aims to highlight the reflection of the differences between users in terms of access levels and company role. Three main roles were identified in order to draw a typical usage flow. These three roles are:

1. administrative role
2. technical role
3. commercial role

Administrative role

As regards the first one, the principal actions that were found are administration and consultative actions. This means that apart from all the operations of system management (which are periodically performed), this role consists of general control actions. The administrator often performs insertion, modification and deletion actions on the system tickets, which includes the possibility of assigning a ticket to a specific user. Last but not least the administrative role is responsible for establishing when a ticket can be considered closed: this is an important action which implies that the ticket is completely closed and no further actions are required on it.

Technical role

The technical role is characterized in a set of actions that are closely related to the practical usage of the system: these actions include all the operations which cover the tickets evolution. In fact, this role is responsible for changing the ticket's status during all its lifecycle; even if the administrator is the person who closes a ticket, all the previous transitions are under the control of the technicians. This means it's usual to see that all the tickets updates are performed by users who cover technical roles.

Commercial role

The last role that was identified is related to all the system users that perform consultation actions for analyzing tickets performances and accordingly manage all the situations which are closely connected with the commercial sphere. Usually also a user who covers a commercial role in the system is allowed to add new tickets in order to follow all the company's activities which the company's commercial section is interested in.

Although these three roles have been enucleated, it's necessary to specify that apart from administrative actions, all the actions described are not rigidly assigned and outlined; however, access levels make it possible to maintain a well-ordered system structure and hierarchy.

3.1.3 MantisBT webservice

“A few years ago Web services were not fast enough to be interesting.”

The presence of a web service interface in MantisBT became a key point in the present Master thesis work; the interface comes as part of the standard MantisBT installation package and it allows an external system to connect to the MantisBT installation. This is the starting point for the project design phase.

In general, a Web service [?] [?] is a method of communication between two electronic devices over the World Wide Web. The W3C defines a “Web service” as “a software system designed to support interoperable machine-to-machine interaction over a network”. It has an interface described in a machine-processable format (specifically Web Services Description Language, known by the acronym *WSDL*). Systems interact with the Web service in a manner prescribed by its description using SOAP messages

(explained in Subsection 3.1.4), typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The *Web Services Description Language* is an XML-based language that is used for describing the functionality offered by a Web service. A WSDL description of a web service (also referred to as a WSDL file) provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a roughly similar purpose as a method signature in a programming language.

A WSDL document describes a web service using these major elements:

Element	Description
<code><types></code>	A container for data type definitions used by the web service
<code><message></code>	A typed definition of the data being communicated
<code><portType></code>	A set of operations supported by one or more endpoints
<code><binding></code>	A protocol and data format specification for a particular port type

MantisBT provides a Web Service whose description can be found at

```
<mantis-install-dir>/api/soap/mantisconnect.php?wsdl,
```

where `<mantis-install-dir>` is the directory where MantisBT system is installed.

The main structure of a WSDL is reported in Listing 3.1:

```

1 <definitions>
2
3 <types>
4   data type definitions .....
5 </types>
6
7 <message>
8   definition of the data being communicated....
9 </message>
10
11 <portType>
12   set of operations .....
13 </portType>
14
15 <binding>
16   protocol and data format specification ....
17 </binding>
```

```

18 |
19 | </definitions>

```

Listing 3.1: Main structure of a WSDL document.

The `<portType>` element is the most important WSDL element; it describes a web service, the operations that can be performed, and the messages that are involved. This element can be compared to a function library (or a module, or a class) in a traditional programming language. The `<message>` element defines the data elements of an operation; each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language. The `<types>` element defines the data types that are used by the web service; for maximum platform neutrality, WSDL uses XML Schema syntax to define data types. The `<binding>` element defines the data format and protocol for each port type.

A piece of the - seriously wide - WSDL of MantisBT is shown in Listing 3.2; for each aforesaid element some code has been reported.

```

1 | <definitions ...>
2 |
3 | <types>
4 |   ...
5 |   <xsd:complexType name="AccountData">
6 |     <xsd:all>
7 |       <xsd:element name="id" type="xsd:integer" ... />
8 |       <xsd:element name="name" type="xsd:string" ... />
9 |       <xsd:element name="real_name" type="xsd:string" ... />
10 |      <xsd:element name="email" type="xsd:string" ... />
11 |      <xsd:element name="access" type="xsd:integer" ... />
12 |    </xsd:all>
13 |  </xsd:complexType>
14 |  ...
15 | </types>
16 |
17 | ...
18 |
19 | <message name="mc_issue_getRequest">
20 |   <part name="username" type="xsd:string"/>
21 |   <part name="password" type="xsd:string"/>
22 |   <part name="issue_id" type="xsd:integer"/>
23 | </message>
24 |
25 | ...
26 |
27 | <portType name="MantisConnectPortType">
28 |   ...

```

```
29 <operation name="mc_version">
30   <input message="tns:mc_versionRequest"/>
31   <output message="tns:mc_versionResponse"/>
32 </operation>
33 ...
34 </portType>
35
36 ...
37
38 <binding name="MantisConnectBinding" ...>
39   <soap:binding style="rpc"
40     transport="http://schemas.xmlsoap.org/soap/http"/>
41
42   ...
43
44   <operation name="mc_version">
45     <soap:operation soapAction=".../mc_version" style="rpc"/>
46     <input>
47       <soap:body use="encoded"
48         namespace="http://futureware.biz/mantisconnect"
49         encodingStyle="..."/>
50     </input>
51
52     <output>
53       <soap:body use="encoded"
54         namespace="http://futureware.biz/mantisconnect"
55         encodingStyle="..."/>
56     </output>
57   </operation>
58
59   ...
60
61 </binding>
```

Listing 3.2: Extract of the MantisBT Web Service Description.

3.1.4 SOAP interface

SOAP [?] is a simple XML-based protocol to let applications exchange information over HTTP. It is important for application development to allow Internet communication between programs. Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

A better way to communicate between applications is over HTTP, be-

cause HTTP is supported by all Internet browsers and servers. SOAP¹ was created to accomplish this. SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

The SOAP protocol has been widely adopted for the communication of the application module with the MantisBT installation on the server. A technical description of the SOAP technology is now presented in order to understand the interaction module that has been developed and which is described in Section 5.2.

A SOAP message is an ordinary XML document containing the following elements:

- An **Envelope** element that identifies the XML document as a SOAP message
- A **Header** element that contains header information
- A **Body** element that contains call and response information
- A **Fault** element containing errors and status information

All the elements above are declared in the default namespace for the SOAP envelope:

`http://www.w3.org/2001/12/soap-envelope`

The default namespace for SOAP encoding and data types is:

`http://www.w3.org/2001/12/soap-encoding`

Here are some important syntax rules:

- A SOAP message must be encoded using XML
- A SOAP message must use the SOAP Envelope namespace
- A SOAP message must use the SOAP Encoding namespace
- A SOAP message must not contain a DTD² reference

¹SOAP became a W3C Recommendation 24. June 2003.

²A Document Type Definition (DTD) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, HTML).

- A SOAP message must not contain XML Processing Instructions

The skeleton of a SOAP message can be viewed in Listing 3.3:

```
1 <?xml version="1.0"?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5
6   <soap:Header>
7     ...
8   </soap:Header>
9
10  <soap:Body>
11    ...
12    <soap:Fault>
13      ...
14    </soap:Fault>
15  </soap:Body>
16
17 </soap:Envelope>
```

Listing 3.3: Skeleton of a SOAP message

Envelope

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message (Listing 3.4).

```
1 <?xml version="1.0"?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5   ...
6   Message information goes here
7   ...
8 </soap:Envelope>
```

Listing 3.4: SOAP message envelope.

The *namespace* defines the Envelope as a SOAP Envelope. If a different namespace is used, the application generates an error and discards the message. The *encodingStyle* attribute is used to define the data types adopted in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements. A SOAP message has no default encoding.

Header

The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element. All immediate child elements of the Header element must be namespace-qualified.

```
1 <?xml version="1.0"?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5
6   <soap:Header>
7     <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
8       soap:mustUnderstand="1">234
9     </m:Trans>
10  </soap:Header>
11  ...
12  ...
13 </soap:Envelope>
```

Listing 3.5: SOAP message header.

The example in Listing 3.5 contains a header with a `Trans` element, a `mustUnderstand` attribute with a value of 1, and a value of 234.

SOAP defines three attributes in the default namespace (`http://www.w3.org/2001/12/soap-envelope`). These attributes are:

- *mustUnderstand*
- *actor*
- *encodingStyle*

The attributes defined in the SOAP Header define how a recipient should process the SOAP message.

The SOAP *mustUnderstand* attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path. The SOAP *actor* attribute is used to address the Header element to a specific endpoint.

The *encodingStyle* attribute, as I said before, is used to define the data types adopted in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.

Body

The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message. Immediate child elements of the SOAP Body element may be namespace-qualified.

```

1 <?xml version="1.0"?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5
6   <soap:Body>
7     <m:mc_version xmlns:m="http://www.w3schools.com/prices">
8       </m:mc_version>
9   </soap:Body>
10
11 </soap:Envelope>

```

Listing 3.6: SOAP message body.

The example in Listing 3.6 requests the version of the MantisBT installation. Note that the `m:mc_version` is an application-specific element (this is just an example: all the details of the MantisBT interaction are widely described in Subsection 5.2.2) and it is not a part of the SOAP namespace.

Fault

The optional SOAP Fault element is used to indicate error messages. If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message. The SOAP Fault element has the following sub elements:

Sub Element	Description
<code><faultcode></code>	A code for identifying the fault
<code><faultstring></code>	A human readable explanation of the fault
<code><faultactor></code>	Information about who caused the fault to happen
<code><detail></code>	Holds application specific error information related to the Body element

HTTP protocol

HTTP communicates over TCP/IP. An HTTP client connects to an HTTP server using TCP; after establishing a connection, the client can send an HTTP request message to the server. The server then processes the request and sends an HTTP response back to the client: the response contains a status code that indicates the status of the request. A SOAP method is an HTTP request/response that complies with the SOAP encoding rules; a SOAP request could be an HTTP POST or an HTTP GET request.

3.2 Operative Phase

This ample section describes the challenging path that was followed in order to obtain a comprehensive and complete picture of the best approach to adopt for the development phase. The commitment to the approach choice is strictly related to two main points:

- the technological instrumentation that is used in the company,
- the specifics of the tool to design.

As regards the first point it is useful to say that every user who works in the company which this Master thesis work has collaborated with, has a personal mobile phone that can access to internet. Furthermore the company has different mobile devices like smartphones and tablets with different OS platforms. Thus the range of mobile OSes present in the company is quite wide and comprehends:

- iOS of Apple
- Android of Google
- Blackberry OS of Blackberry
- Windows Phone of Windows

Since these are strongly different platforms, it played a key role in the development approach choice.

The specifics of the tool to design has represented a significant aspect to keep in mind for all the choice process; in fact the best approach to use is also strictly connected to the several features that such an application may have. In this phase, all the project specifics were defined and grouped into different sets that had therefore conditioned the above choice.

These specifics sets have been expressed into the following terms:

- front-end operations cardinality,
- back-end degree of processing,
- graphic interface complexity degree.

All these details are widely illustrated in Section 3.2.3. Now it is sufficient to underline that in the following approach choice description they had contributed to determine the choice.

3.2.1 Native, web and hybrid approaches

As I mentioned above each of these three approaches carries benefits and limitations. In general, there is no such thing as the "best" development approach, as this is non-existent, but the environment analysis made it possible to list the pros and cons each carries and to describe the aforesaid requirements that best fit one or the other.

Native approach

Native apps have binary executable files that are downloaded directly to the device and stored locally. The installation process can be initiated by the user or, in some cases, by the IT department of the organization. The most popular way to download a native app is by visiting an app store, such as Apple's App Store, Android's Marketplace or BlackBerry's App World, but other methods exist and are sometimes provided by the mobile vendor. Once the app has been installed on the device, the user launches it like any other service the device offers. Upon initialization, the native app interfaces directly with the mobile operating system, without any intermediary or container. The native app is free to access all of the APIs that are made available by the OS vendor and, in many cases, has unique features and functions that are typical of that specific mobile OS.

To create a native app, developers must write the source code (in human-readable form) and create additional resources, such as images, audio segments and various OS-specific declaration files. Using tools provided by the OS vendor, the source code is compiled (and sometimes also linked) in order to create an executable in binary form that can be packaged along with the rest of the resources and made ready for distribution. These tools, in addition to other utilities and files, are normally called the software development kit (SDK) of the mobile OS. Although the development process is often similar for different operating systems, the SDK is

platform-specific and each mobile OS comes with its own unique tools. The Table 3.1 presents the different tools, languages, formats and distribution channels associated with the leading mobile operating systems.

	Apple iOS
Language	Objective-C, C, C++
Tool	Xcode
Packaging format	.app
App store	Apple App Store
	Android
Language	Java (some C, C++)
Tool	Android SDK
Packaging format	.apk
App store	Google Play
	Blackberry OS
Language	Java
Tool	BB Java Eclipse Plug-in
Packaging format	.cod
App store	Blackberry App World
	Windows Phone
Language	C#, VB.NET and more
Tool	Visual Studio, Windows Phone development tools
Packaging format	.xap
App store	Windows Phone Marketplace

Table 3.1: Mobile OSes comparison

These differences across platforms result in one of the most critical disadvantages of the native development approach – code written for one mobile platform cannot be used on another, making the development and maintenance of native apps for multiple OSes a very long and expensive undertaking. The reason for which many companies choose to develop natively is related to a significant factor: it's important to understand the role of the APIs.

Once the native application is installed on the mobile device and launched by the user, it interacts with the mobile operating system through proprietary API calls that the operating system exposes. These can be divided into two groups:

- low-level APIs
- high-level APIs

Low-level APIs It is through these low-level API calls that the app can interact directly with the touch screen or keyboard, render graphics, connect to networks, process audio received from the microphone, play sounds through the speaker or headphones, or receive images and videos from the camera. It can access the Global Positioning System (GPS), receive orientation information and, of course, read and write files on the solid-state disk or access any other hardware element available today or in the future.

High-level APIs In addition to providing the low-level hardware-access services I just mentioned, mobile operating systems also provide higher-level services that are important to the personal mobile experience. Such services include processes like browsing the web, managing the calendar, contacts, photo album and, of course, the ability to make phone calls or send and receive text messages. Although most mobile OSes include a set of built-in applications that can execute these services, a set of exposed high-level APIs is made accessible for native apps as well, allowing them to access many of the important services mentioned above (see Figure 3.2). Other APIs enable downloadable apps to access various cloud-based services that are provided by the OS vendor, such as push notifications or in-app purchases.

The graphical user interface (GUI) toolkit Another important set of APIs that the OS provides is the GUI toolkit. Each mobile OS comes with its own set of user interface components, such as buttons, input fields, sliders, menus, tab bars, dialog boxes and so on. Apps that make use of these components inherit the features and functions of that specific mobile OS, which normally results in a very easy and enjoyable user experience. It's important to note that different mobile platforms carry unique palettes of user interface (UI) components. As a result, apps that are designed to work for multiple operating systems require the designer to be familiar with the different UI components of each OS. Although APIs

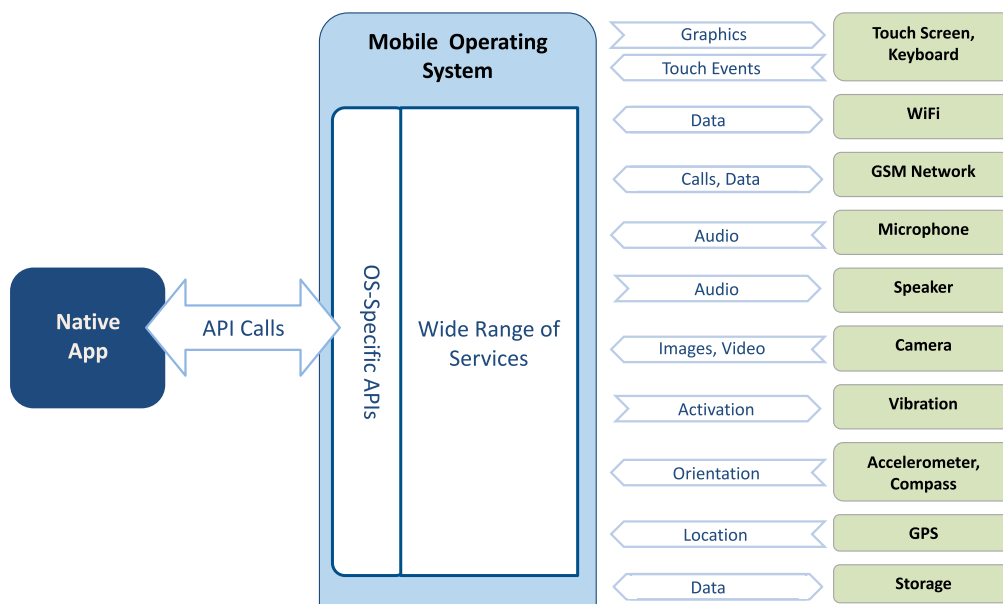


Figure 3.2: The native application interacts with the mobile operating system through proprietary API calls that the operating system exposes.

are OS-specific and add much complexity and cost to the development of multiple native apps, these elements are the only means of creating rich mobile applications that make full use of all the functionality that modern mobile devices have to offer.

Web approach

Modern mobile devices consist of powerful browsers that support many new HTML5 capabilities, Cascading Style Sheets 3 (CSS3) and advanced JavaScript. With recent advancements on this front, HTML5 signals the transition of this technology from a “page-definition language” into a powerful development standard for rich, browser-based applications. A few examples of the potential of HTML5 include advanced UI components, access to rich media types, geolocation services and offline availability. Using these features and many more that are under development, developers are able to create advanced applications, using nothing but web technologies. It is helpful to distinguish between two extreme web-app approaches. Everyone is familiar with mobile browsing and mobile-optimized websites. These sites recognize when they are accessed by a smartphone and serve up HTML pages that have been designed to provide a comfortable “touch ex-

perience” on a small screen size. But some companies go even further and enhance the user experience by creating a mobile website that looks like a native app and can be launched from a shortcut that is indistinguishable from that used to launch native apps.

There is a wide range of possibilities between these two extremes, with most websites implementing their own mix of features. Mobile web apps are a very promising trend. To capitalize on this trend and help developers build the client-side UI, a growing number of JavaScript toolkits have been created, such as `dojox.mobile`, `Sencha Touch` and `jQuery Mobile`, which generate user interfaces that are comparable in appearance to native apps. Both execute entirely within the browser of the mobile device and make use of the newest JavaScript, CSS and HTML5 features that are available in modern mobile browsers. One of the most prominent advantages of a web app is its multiplatform support and low cost of development. Most mobile vendors utilize the same rendering engine in their browsers, `WebKit` – an open-source project led mainly by Google and Apple that provides the most comprehensive HTML5 implementation available today. Because the application code is written in standard web languages that are compatible with `WebKit`, a single app delivers a uniform experience for different devices and operating systems, making it multiplatform by default. However, these advantages are not without a price.

Despite the potential and promise of web technologies in the mobile space, they still carry significant limitations. To understand these limitations I need to explain how web applications operate. Unlike native apps, which are independent executables that interface directly with the OS, web apps run within the browser. The browser is in itself a native app that has direct access to the OS APIs, but only a limited number of these APIs are exposed to the web apps that run inside it (see Figure 3.3). While native apps have full access to the device, many features are only partially available to web apps or not available at all. Although this is expected to change in the future with advancements in HTML, these capabilities are not available for today’s mobile users.

Hybrid approach

The hybrid approach combines native development with web technology. Using this approach, developers write significant portions of their application in cross-platform web technologies, while maintaining direct access to native APIs when required. The native portion of the application uses the

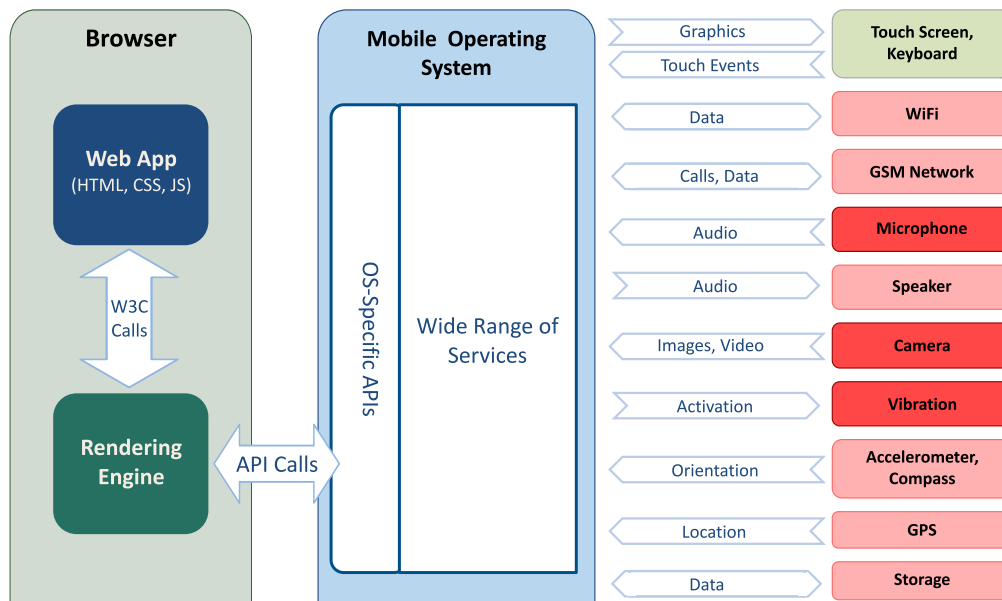


Figure 3.3: Web apps run within the browser which is in itself a native app that has direct access to the OS APIs: only a limited number of these APIs (colored in green) are exposed to the web apps that run inside it.

operating system APIs to create an embedded HTML rendering engine that serves as a bridge between the browser and the device APIs. This bridge enables the hybrid app to take full advantage of all the features that modern devices have to offer.

App developers can choose between coding their own bridge or taking advantage of ready-made solutions such as *PhoneGap* – open-source library that provides a uniform JavaScript interface to selected device capabilities that is consistent across operating systems.

The native portion of the app can be developed independently, but some solutions in the market provide this type of a native container as part of their product, thus empowering the developer with the means to create an advanced application that utilizes all the device features using nothing but web languages. In some cases, a solution will allow the developer to use any native knowledge he or she might have to customize the native container in accordance with the unique needs of the organization (see Figure 3.4)

The web portion of the app can be either a web page that resides on a server or a set of HTML, JavaScript, CSS and media files, packaged into the application code and stored locally on the device. Both approaches carry advantages and limitations. HTML code that is hosted on a server

enables developers to introduce minor updates to the app without going through the process of submission and approval that some app stores require. Unfortunately, this approach eliminates any off line availability, as the content is not accessible when the device is not connected to the network. On the other hand, packaging the web code into the application itself can enhance performance and accessibility, but does not accept remote updates. The best of both worlds can be achieved by combining the two approaches. Such a system is designed to host the HTML resources on a web server for flexibility, yet cache them locally on the mobile device for performance.

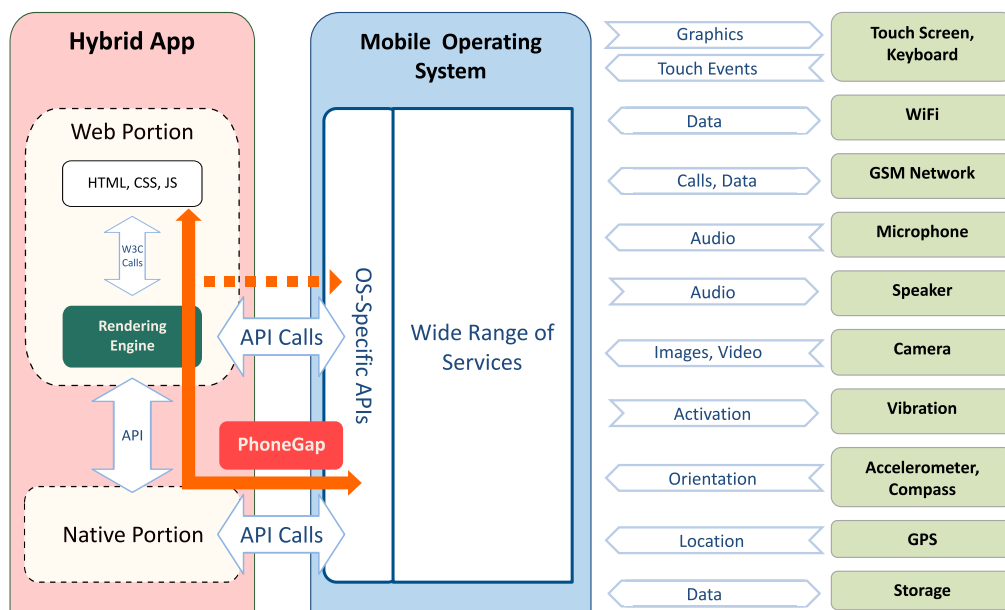


Figure 3.4: The native portion of the application uses the operating system APIs to create an embedded HTML rendering engine that serves as a bridge between the browser and the device APIs. This bridge enables the hybrid app to take full advantage of all the features (colored in green) that modern devices have to offer.

3.2.2 Comparing the different approaches

To summarize, a comparison of all three development approaches follows. The native approach excels in performance and device access, but suffers in cost and updates. The web approach is much simpler, less expensive and easier to update, but is currently limited in functionality and cannot

achieve the exceptional level of user experience that can be obtained using native API calls. The hybrid approach provides a middle ground which, in many situations, is the best of both worlds, especially if the developer is targeting multiple operating systems.

As can be inferred from the Table 3.2, no single approach delivers all the benefits all the time. Choosing the right approach depends on the specific needs of the organization and can be driven by many parameters, such as budget, timeframe, internal resources, target market, required application functionality, IT infrastructure and many others.

Feature	Native App	Hybrid App	Web App
Development language	Native only	Native and web or web only	Web only
Code portability and optimization	None	High	High
Access device-specific features	High	Medium	Low
Leverage existing knowledge	Low	High	High
Advanced graphics	High	Medium	Medium
Upgrade flexibility	Low	Medium	High
Installation experience	High	High	Medium

Table 3.2: Approaches comparison

Here is a list of the possible scenarios that was used in the process of choosing the best approach.

Scenarios for the native approach

Existing native skills One of the main arguments against the native approach is its lack of multiplatform support. Organizations asking to develop an application for multiple mobile platforms need to hire new employees or train in-house developers in a variety of native languages. Organizations that have such native skills in-house are able to take advantage of them, without significant new investments.

A single mobile OS In some cases, an organization will aim to release a mobile application to a limited target audience – one that is known to use a single mobile OS. For example, consider a scenario in which an internal application is distributed within an organization that issues a BlackBerry device to its employees. In this case, achieving multiplatform coverage

might not be a priority and, as developing a single native application requires a limited set of skills and tools, this approach can make much sense.

Native functionality Some applications are built around a single functionality. Take Skype, for example: Voice over Internet Protocol (VoIP) and access to the user's contacts are key elements of the app and, given available technologies today, can only be developed natively. For such applications, web languages are simply not yet sufficiently evolved and are far from capable of achieving the desired functionality.

Rich UI requirements For game-like applications that require a rich UI that provides real-time responsiveness, web technologies do not yet provide an adequate solution. For applications with such requirements, developers are still better off taking the native approach.

Scenarios for the web approach

Direct distribution Some organizations prefer distributing their apps in a manner that is controlled internally and is not subjected to what can sometimes turn into a long and uncertain approval process. In such cases, using purely web languages can completely circumvent the app-store process and allow the organization to fully control the distribution of the app and its periodical updates.

Pilot app When comparing the costs and time to market involved in the development of a native as opposed to a web app, using the web approach to create a pilot version of the app can be a compelling and cost-effective tactic. Once the concept has been proved, the organization can choose to create a new application from the beginning or use portions of the existing code in a hybrid application.

Visibility In addition to the distribution we already mentioned, another benefit of creating a web application is its visibility in search engine results which, in many cases, expose the application to a larger audience than that available through the app store alone.

Scenarios for the hybrid approach

Balancing the tradeoff Using the hybrid approach, companies can enjoy the best of both worlds. On the one hand, the native bridge enables

developers to take full advantage of all the different features and capabilities that modern mobile devices have to offer. On the other, all the portions of the code that are written using web languages can be shared among different mobile platforms, making the development and ongoing maintenance process centralized, shorter and cost-effective.

In-house skills Web development skills are very common and can easily be found in many organizations. By choosing the hybrid approach, supported by the right solution, web developers are able to build applications with nothing but web skills, such as HTML, CSS and JavaScript, while delivering a native-like user experience.

Future considerations HTML5 is rapidly growing in both availability and capabilities. Many analysts predict that it is likely to become the default technology for client-side application development. By writing most of the app in HTML, and using native code only where needed, companies can make sure that the investments they make today do not become obsolete tomorrow, as HTML functionality becomes richer and addresses a wider range of the mobile requirements of modern organizations.

3.2.3 Choosing the right approach

The choice of the approach fell on the *hybrid* strategy. This choice was the last action of a set of examinations in which many test modules for every approach had been created. Since the company was oriented to a multi-platform solution, the hybrid strategy had early encountered favorable opinions. Furthermore during the prototypes testing phase, several issues in the server interaction emerged with the native approach: in particular, the creation of the SOAP envelope for the communication met sticky problems in write-operations invocation.

Another point in favor of the hybrid choice is strictly related to the personal web development skills which early allowed me to build operative application test-modules with nothing but web skills, such as HTML, CSS and JavaScript.

Since the web approach prevents the use of all the services provided by the mobile device, the focus shrank on the comparison between the native and hybrid approaches.

As I outlined in Section 3.2 three factors played a main role in the choice:

Front-end operations cardinality The number of operations that the application has to perform in the interaction with the server-side is limited to a few tens; this aspect relieves an operative burden on the application so as to encourage the hybrid approach.

Back-end degree of processing Also for this aspect, the hybrid approach seemed to be a right solution. The degree of elaborations at the application level is not so high to require the native approach. As it is possible to see in the discussion of Section 5.2, the application's main role is to display content data retrieved from the server. Although there are some elaborations, these can be definitely faced with an hybrid application.

Graphic interface complexity degree As regards this point, the jQuery Mobile library in combination with PhoneGap, allowed a rapid creation of the user interface (details in Section 4.2): the hybrid approach favored a unique and working graphic solution that is automatically optimized for every mobile device screen size - both smartphones and tablets.

Chapter 4

Development Environment

After choosing the approach to tackle the work, the analysis moves on to an accurate description of the development environment adopted. This is an indispensable task in order to grasp all the benefits that this choice involves. The cross-platform is the key feature of this strategy: this chapter deals with the environment description.

4.1 PhoneGap

PhoneGap (previously called Apache Callback, but now Apache Cordova) is an open-source mobile development framework produced by Nitobi, purchased by Adobe Systems. It enables software programmers to build applications for mobile devices using JavaScript, HTML5 and CSS3, instead of device specific languages (see intuitive representation in Figure 4.1).

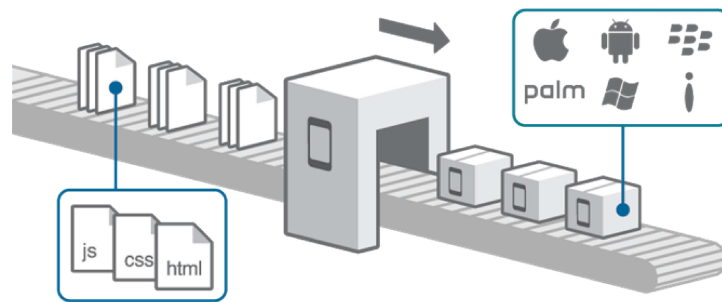


Figure 4.1: PhoneGap executive flow.

PhoneGap is a HTML5 application framework that is used to develop hybrid applications. This means that developers can develop Smartphone

and Tablet applications with their existing knowledge of HTML, CSS, and JavaScript. With PhoneGap, developers don't have to learn OS specific languages. These kind of applications are not purely HTML/JavaScript based, nor are they native. Parts of the application, mainly the UI, the application logic, and communication with a server, is based on HTML/JavaScript. The other part of the application that communicates and controls the device (phone or tablet) is based on the native language for that platform. PhoneGap provides a bridge from the JavaScript world to the native world of the platform, which allows the JavaScript API to access and control the device (phone or tablet).

The PhoneGap framework is primarily a JavaScript Library that allows HTML/JavaScript applications to access device features. The PhoneGap framework also has a native component, which works behind the scene and does the actual work on the device (phone or tablet). In Figure 4.2 is presented the overall PhoneGap architecture. An application build using PhoneGap will primarily have two parts:

1. The JavaScript Business Logic Part, which drives the UI and its functionality.
2. The JavaScript Part, which accesses and controls the device (phone or tablet).

PhoneGap was made possible due to a *commonality* between all of the mobile platforms: the *browser*. If it were not for this common component, PhoneGap would not have been possible.

4.1.1 Web Browser

The browser world was largely fragmented until just a few years ago. At the time, different browsers adhered to W3C standards to different degrees. Firefox and Safari browsers were at the forefront in terms of adhering to standards, while others lagged behind.

A lot has changed since then. Now, browsers are looking better in terms of adhering to standards (more so on the mobile platforms). This is also true because most modern mobile platforms have the same webkit-based browser.

Also, newer browsers, both on desktops and smartphones, have started to adhere to newer standards like HTML5/CSS3. This adds more features to the browser world and lessens the fragmentation across mobile platforms.

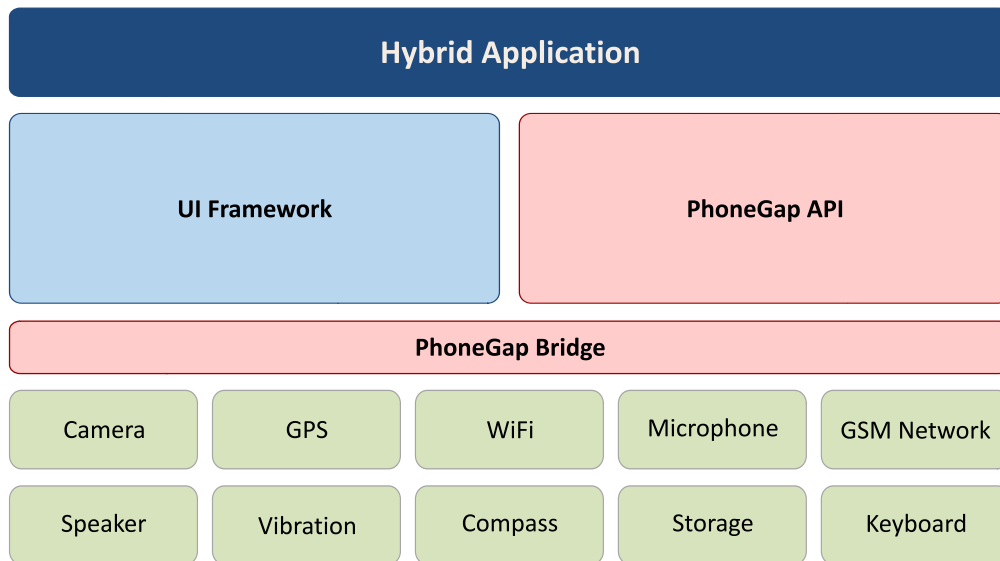


Figure 4.2: PhoneGap application architecture

Table 4.1 lists mobile platforms and their corresponding browser platforms. All mobile platforms except the Windows 7 Phone use a webkit-based browser. While the Windows 7 Phone has its own browser, the good news is that all of the browsers listed here are already adhering to HTML5/CSS3 standards, and with the passage of time, their adherence will continue to improve.

Mobile OS	Browser
Android	Webkit-based
iOS	Webkit-based
BlackBerry OS	Webkit-based
Windows 7 Phone	IE 7-based
WebOS	Webkit-based
Nokia	Webkit-based
BADA	Webkit-based

Table 4.1: Mobile Browsers

PhoneGap uses these modern browsers as the platform for building HTML5/CSS3-based applications.

4.1.2 Features

The main features of PhoneGap are:

1. The camera API of PhoneGap allows applications to retrieve a picture from either the camera or fetch the images from already existing photo galleries.
2. The contacts API of PhoneGap is a way for applications to read and write contacts. Many social applications can benefit from syncing phone contacts with contacts on social channels.
3. The geolocation API helps to retrieve the device's geolocation. This is good for many applications, including map-based applications, and applications like foursquare, where the user can check-in to a place by using their GPS location. There is an option to fetch one reading of change in device geo location or to continuously receive the changes in device geo location.
4. The file API of PhoneGap allows applications to read, write, and list directories and file systems. This is handy if the application is planning to change the contents of a file in the file system of the phone. This API can also help write file explorer applications.
5. The compass API of PhoneGap helps the applications know the bearing of the phone. This proves to be useful for map and navigation applications, since the map rotates as the user changes the bearing of the phone. There is an option to fetch one reading of change in device heading or to continuously receive the changes in device heading.
6. The accelerometer API of PhoneGap enables the application to sense change in the device's orientation, therefore, it is able to act accordingly. This can be useful in creating applications that have a bubble level (making sure the phone is aligned horizontally to the ground). There is an option to fetch one reading of change in device orientation or to continuously receive the changes in device orientation.
7. The media API allows applications to control the media sensors and applications on the device. This API allows applications to record and playback audio and video recordings.
8. The notification API allows applications to notify the user that something has occurred, by making a beep, vibration, or providing a visual alert.

9. The storage API of PhoneGap provides a built-in SQL database for the applications. An application can insert, retrieve, update, and delete data through SQL statements. Applications can query data in the database, and search for a specific e-mail in a locally stored list of e-mails.
10. The network API of PhoneGap provides the applications with the ability to see the state of the network. Instead of this state being just online and offline, this tells the application whether the device is on a 2G/3G/4G network or a Wi-Fi network. Such information often helps the application decide when to retrieve certain kinds of information.

4.2 jQuery Mobile

While PhoneGap provides a platform to allow JavaScript apps to access native phone features, there are many other things that contribute to a mobile HTML app. One of the most important parts of the mobile HTML application is the UI. The entire UI could be written by hand using HTML, JavaScript, and CSS. However, any web developer knows that there are many issues with this approach, including the following:

- Not all browsers are same; a cross-browser framework is necessary to be successful. Even if most mobile browser are webkit based, it's best to use a framework that abstracts the browser differences from a developer.
- A framework that lets a programmer to write less and do more would help actually to focus on the business logic.
- Creating an aesthetically good-looking HTML UI requires designer skills. At the same time, most mobile clients have predefined themes or schemas. It would help a developer if a framework provides good-looking UI out of the box.

One of the easiest frameworks to use with PhoneGap to write the UI is jQueryMobile. First of all, jQueryMobile is built on top of the very popular jQuery. jQuery is known to be a JavaScript library that increases developer productivity and helps developers with cross-browser compatibility. At the same time, there are many free plug-ins available with jQuery to do a lot of things.

jQueryMobile is a UI framework built for a mobile UI. It has a declarative UI, which means the developer doesn't have to code the UI in JavaScript, but just declares it in HTML. jQueryMobile also provides an excellent looking UI out of the box.

All this makes jQueryMobile the most easy to use JavaScript UI framework and the most appropriate framework for a mobile UI of moderate complexity.

The framework includes an Ajax navigation system that brings animated page transitions and a core set of UI widgets; the principals are:

- pages
- dialogs
- toolbars
- listviews
- buttons with icons
- form elements
- accordions
- collapsibles

jQuery Mobile is easy to learn with a simple, markup-based system to applying behavior and theming. For more advanced developers, there is a rich API of global configuration options, events, and methods to apply scripting and generate dynamic pages.

To make this broad support possible, all pages in jQuery Mobile are built on a foundation of clean, semantic HTML to ensure compatibility with pretty much any web-enabled device. In devices that interpret CSS and JavaScript, jQuery Mobile applies progressive enhancement techniques to unobtrusively transform the semantic page into a rich, interactive experience that leverages the power of jQuery and CSS. Accessibility features such as WAI-ARIA are tightly integrated throughout the framework to provide support for screen readers and other assistive technologies.

4.3 Setting the environment

The PhoneGap environment can be setup in the following two manners:

- Local development environment

- Cloud build environment on PhoneGap Build

The local development environment setup includes the developer setting up environments for each mobile platform that the developer wants to launch a PhoneGap application on.

The cloud build environment called “PhoneGap Build” allows the developer to build PhoneGap applications without the need for a local development environment. This means that a developer will only code the PhoneGap portion of the application, which requires HTML, JavaScript, and CSS. This code will then be provided to the PhoneGap Build service. The PhoneGap Build service will build the required binaries for each platform and the developer can download these.

4.3.1 Local Development Environment

In this subsection it’s explained how to set up a local development environment for the following platforms:

1. Android
2. iOS

The description is focused on these two platforms because in the company environments, where the Master thesis work has been developed, they cover most of the adopted devices.

Getting started with Android

1. Requirements

- Eclipse 3.4+

2. Install SDK and Cordova¹

- Download and install Eclipse Classic (<http://www.eclipse.org/downloads/>);
- Download and install Android SDK (<http://developer.android.com/sdk/index.html>);
- Download and install ADT Plugin (<http://developer.android.com/sdk/eclipse-adt.html#installing>);

¹Apache *Cordova* is the actual name of PhoneGap

- Download the latest copy of Cordova and extract its contents (<http://phonegap.com/download>).

3. Setup New Project

- Launch Eclipse, and select menu item New Project;
- Specify new Android Application Project (Figure 4.3);

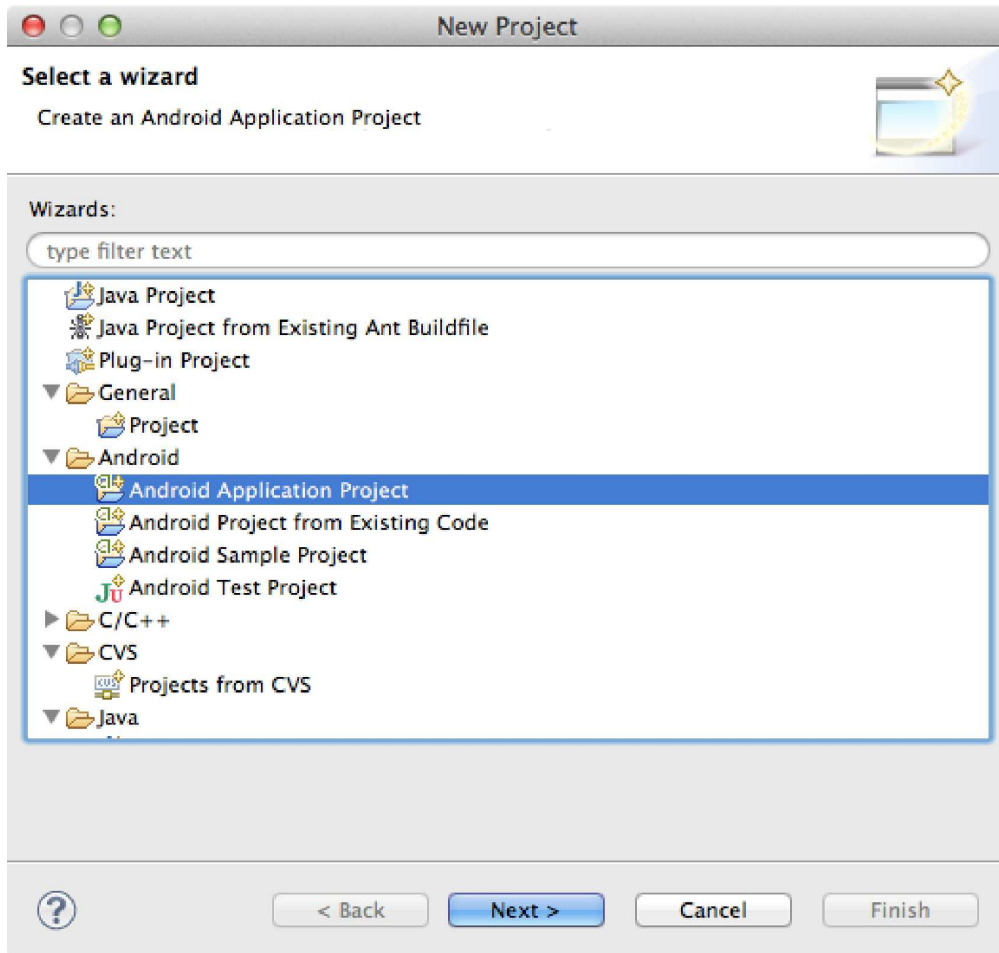


Figure 4.3: New Android Application project.

- Specify new Application Project information (Application name, Project name, Package name with Namespace and Build SDK);
- Select a graphic of the launcher icon;
- Create a Blank Activity;

- Make sure the activity doesn't inherit from anything. Once this is done, click Finish (Figure 4.4).

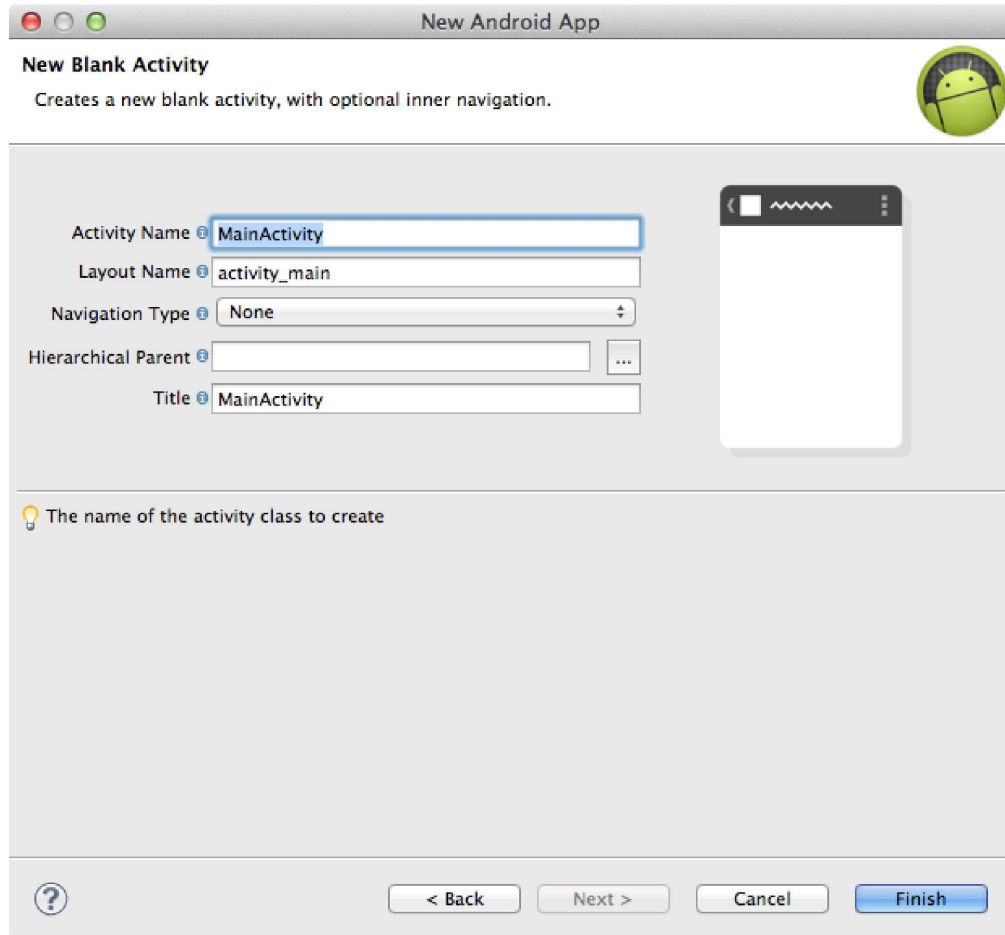


Figure 4.4: Click Finish to create the project.

- In the root directory of the project, create two new directories:
 - /libs
 - assets/www
- Copy **cordova-2.0.0.js** from the Cordova download earlier to assets/www
- Copy **cordova-2.1.0.jar** from the Cordova download earlier to /libs
- Copy the **xml** folder from the Cordova download earlier to /res

- Verify that **cordova-2.1.0.jar** is listed in the Build Path for the project. Right click on the /libs folder and go to

Build Paths/ > Configure Build Path...

Then, in the Libraries tab, add **cordova-2.1.0.jar** to the project.

- Edit the project's main Java file found in the src folder in Eclipse (Figure 4.5):
 - Add `import org.apache.cordova.*;`
 - Change the class's extend from `Activity` to `DroidGap`;
 - Replace the `setContentView()` line with `super.loadUrl("file:///android_asset/www/index.html");`.

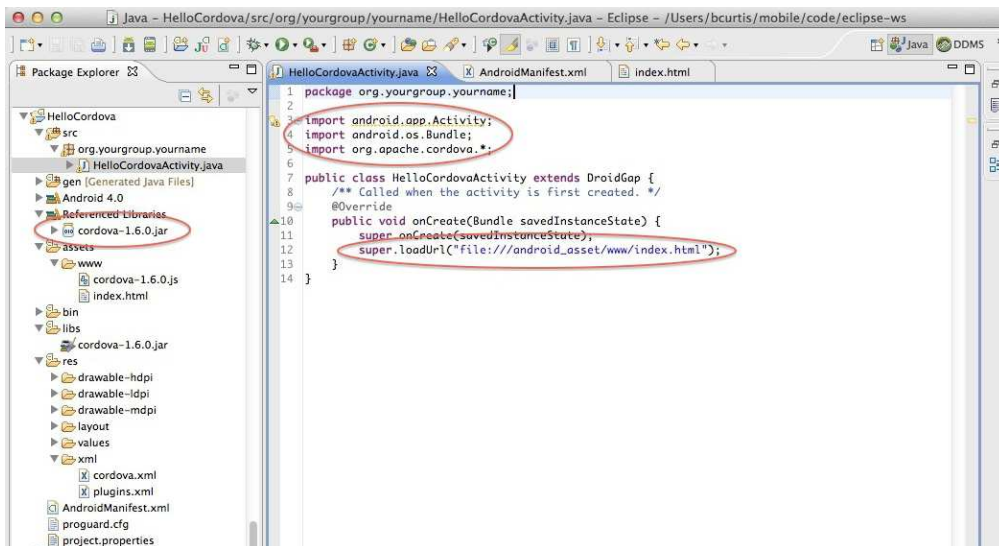
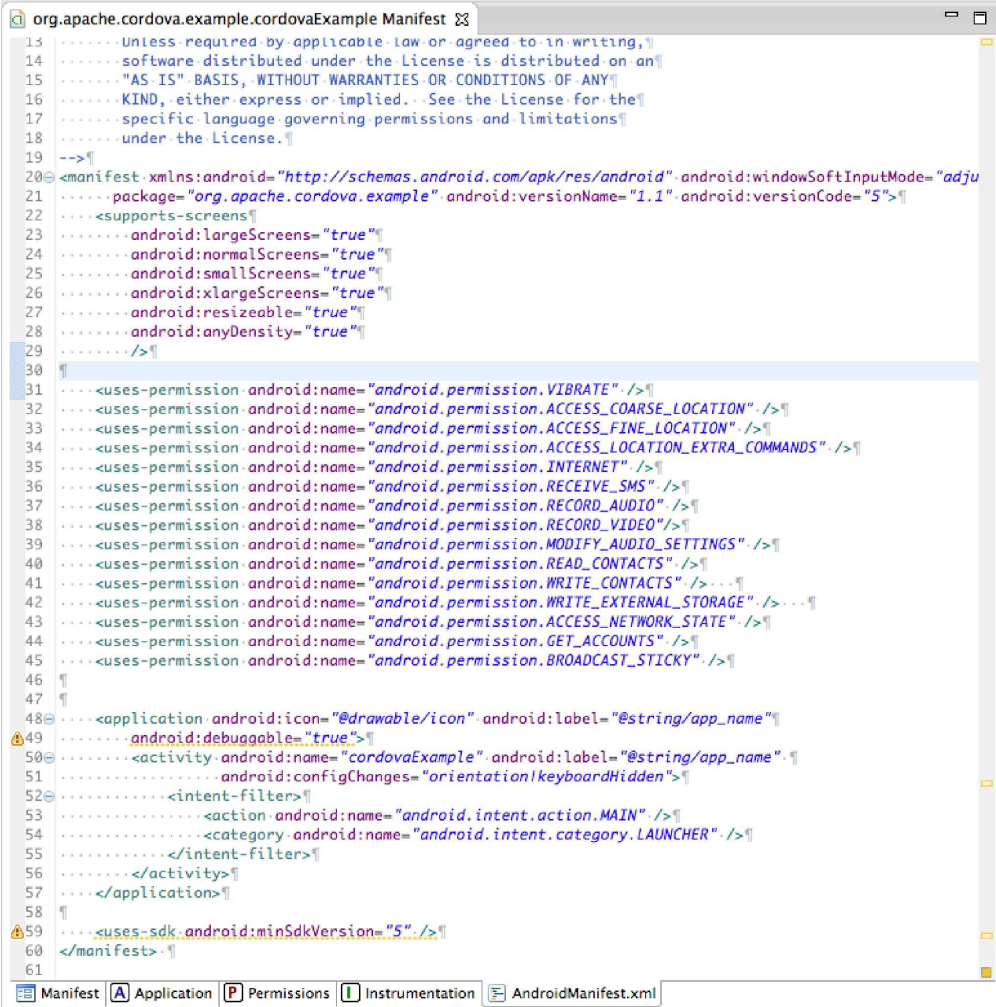


Figure 4.5: Edit the project's Java file.

- Right click on `AndroidManifest.xml` and select `Open With > Text Editor`;
- Edit the `AndroidManifest.xml` file; it should look like Figure 4.6.

4. Deploy to simulator

- Right click the project and go to `Run As > Android Application`
- Eclipse will ask to select an appropriate AVD. If there isn't one, it's necessary to create it.



```

13 .....Unless required by applicable law or agreed to in writing,
14 .....software distributed under the License is distributed on an
15 ..... "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
16 ..... KIND, either express or implied. See the License for the
17 ..... specific language governing permissions and limitations
18 ..... under the License.
19 -->
20 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:windowSoftInputMode="adju
21 ..... package="org.apache.cordova.example" android:versionName="1.1" android:versionCode="5">
22 ..... <supports-screens
23 .....     android:largeScreens="true"
24 .....     android:normalScreens="true"
25 .....     android:smallScreens="true"
26 .....     android:xlargeScreens="true"
27 .....     android:resizeable="true"
28 .....     android:anyDensity="true"
29 ..... />
30
31 ..... <uses-permission android:name="android.permission.VIBRATE" />
32 ..... <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
33 ..... <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
34 ..... <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
35 ..... <uses-permission android:name="android.permission.INTERNET" />
36 ..... <uses-permission android:name="android.permission.RECEIVE_SMS" />
37 ..... <uses-permission android:name="android.permission.RECORD_AUDIO" />
38 ..... <uses-permission android:name="android.permission.RECORD_VIDEO" />
39 ..... <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
40 ..... <uses-permission android:name="android.permission.READ_CONTACTS" />
41 ..... <uses-permission android:name="android.permission.WRITE_CONTACTS" />
42 ..... <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
43 ..... <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
44 ..... <uses-permission android:name="android.permission.GET_ACCOUNTS" />
45 ..... <uses-permission android:name="android.permission.BROADCAST_STICKY" />
46
47
48 ..... <application android:icon="@drawable/icon" android:label="@string/app_name"
49 .....     android:debuggable="true">
50 .....     <activity android:name="cordovaExample" android:label="@string/app_name"
51 .....         android:configChanges="orientation|keyboardHidden">
52 .....         <intent-filter>
53 .....             <action android:name="android.intent.action.MAIN" />
54 .....             <category android:name="android.intent.category.LAUNCHER" />
55 .....         </intent-filter>
56 .....     </activity>
57 ..... </application>
58
59 ..... <uses-sdk android:minSdkVersion="5" />
60 </manifest>
61
Manifest Application Permissions Instrumentation AndroidManifest.xml

```

Figure 4.6: Edit the Android Manifest.

Getting started with iOS

1. Requirements

- Xcode 4.3+
- Xcode Command Line Tools
- Intel-based computer with Mac OS X Lion or greater (10.7+)
- Necessary for installing on device:
 - Apple iOS device (iPhone, iPad, iPod Touch)
 - iOS developer certificate

2. Install the iOS SDK and Cordova

- Install Xcode from the *Mac App Store* or *Apple Developer Downloads*;
- Install the Xcode Command Line Tools (from Xcode Preferences);
- Download and extract the latest release of Apache Cordova.

3. Install *CordovaLib*

- Download the Cordova source (<https://github.com/apache/incubator-cordova-ios>);
- Extract to a permanent location on the hard drive.

4. Create a New Project

- Launch *Terminal.app*;
- Drag the `bin` folder (located in the permanent folder location of Cordova) to the *Terminal.app* icon in the Dock: it should launch a new Terminal window;
- Type in `./create <project_folder_path> <package_name> <project_name>` then press `Enter` (see Table 4.2).

Command	Description
<code><project_folder_path></code>	is the path to the new Cordova iOS project (it must be empty if it exists)
<code><package_name></code>	is the package name, following reverse-domain style convention
<code><project_name></code>	is the project name

Table 4.2: Terminal commands for creating a new iOS Project

- Locate the new project folder just created;
- Launch the `.xcodeproj` file in the folder.

5. Deploy to Simulator

- Change the *Active SDK* in the Scheme drop-down menu on the toolbar to `iOS [version] Simulator`;
- Select the *Run* button in your project window's toolbar.

Chapter 5

Software Design

This chapter illustrates the whole software design phase; this phase played a key role in this Master thesis work. All the software requirements are expressed here, and they are structured so as to obtain a clear representation of the tools to be developed.

The design phase was focused on two tools:

- applicative component,
- server-side component.

In fact these are the two basic components which defined the entire work structure. The chapter starts by describing the general architecture of the project and later focuses on the aforesaid components.

For each one, a general description introduces two more specific analyses: firstly, the functional analysis covers all the functional requirements that were expressed within the company's environment. Secondly, a specific technical analysis describes all the implementation details.

5.1 General Architecture

The installation of MantisBT exposes a SOAP interface which provides all the operations that can likewise be performed via web interface. Through the SOAP interface the applicative component performs all the operations; besides common operations, which are described in the following sections, this component presents an evaluation section for a specific purpose: the application user (with administrative permissions) can retrieve performance information related to realization indexes of users. These values act as performance indicators on all the company's activities that are tracked by the control system.

As I said in the Introduction chapter, this work focused the attention on a project that could be fully integrated in the *existing* activities control system. This aspect played a key role in the applicative design. Generally speaking, the application retrieves - and exposes - performance information that is used by the administrative user to take decisions that are therefore in accordance with past company performances, which is the role of the server-side component.

The design phase led to an important decision, which is that of limiting strongly the excessive number of elaborations on the applicative side, in order to keep the applicative computational exertion as lean as possible. Like the application, also the server component uses the SOAP interface to perform the evaluation actions. It is worth underlining that the server component is a module that is fully integrated in the control system. This is a feature that aims to get an early adoptable solution within the company environment.

5.1.1 Evaluation System

As I mentioned in Section 2.1, Mida Solutions was officially certified in accordance with ISO 9001, an aspect characterized the whole view of this Master thesis work. The effort was to integrate an activities evaluation module into the existing certified Quality Management System (referred to as QMS).

In fact, an effective ISO management system must incorporate monitoring and measurement of key quality performance indicators, which would provide management objective data upon which to base decisions.

Internal auditing is an “early warning system” to help the company spot problems that could impact customer satisfaction or operating efficiency - giving the chance to address and solve them before they are detected by others, rather than afterwards. And then management review provides management with solid data, enabling management to make decisions based on actual facts and evidence.

Generally speaking, ISO 9001 includes a continuous process improvement system that requires:

- well-defined objectives,
- ...which are put in place,
- ...with methods for data collection and analyze.

This means that the organization will establish, document, implement and maintain a QMS, and continually improve its effectiveness in accordance with this International Standard. In particular, the organization will:

- Identify processes needed for QMS and their application throughout the organization
- Determine sequence and interaction of these processes
- Determine criteria and methods needed to ensure that both operation and control of processes are effective
- Ensure availability of resources and information necessary to support operation and monitoring of these processes
- Monitor, measure and analyze these processes
- Implement actions necessary to achieve planned results and continual improvement of these processes.

Thus, the attention focused on the analysis of a function - fully integrated into the system - which could carefully follow the above points. The possibility of evaluating classified tickets makes the system a precious source of performance indicators for every user: this aims straight at the continuous improvement expressed in the ISO 9001.

5.1.2 Project Structure

The Activity Tracking System acts as an intermediary between the *Application* and the *Server Component*. More precisely, both components retrieve information from the tracking system to perform different actions. As shown in the components diagram of Figure 5.1, the *Server Component* interacts through the SOAP interface. Also the application speaks directly to the tracking system, which is done to obtain all the informative content that the user requests from the mobile device.

Both the Tracking System and the *Server Component*, which is a background daemon, perform an initial configuration which draws from a common repository. In order to maintain information consistency, this configuration repository is nothing but a specific set of special tickets which are properly created and managed to this purpose. This feature allows the two components - the *Application* and the daemon - to “speak the same

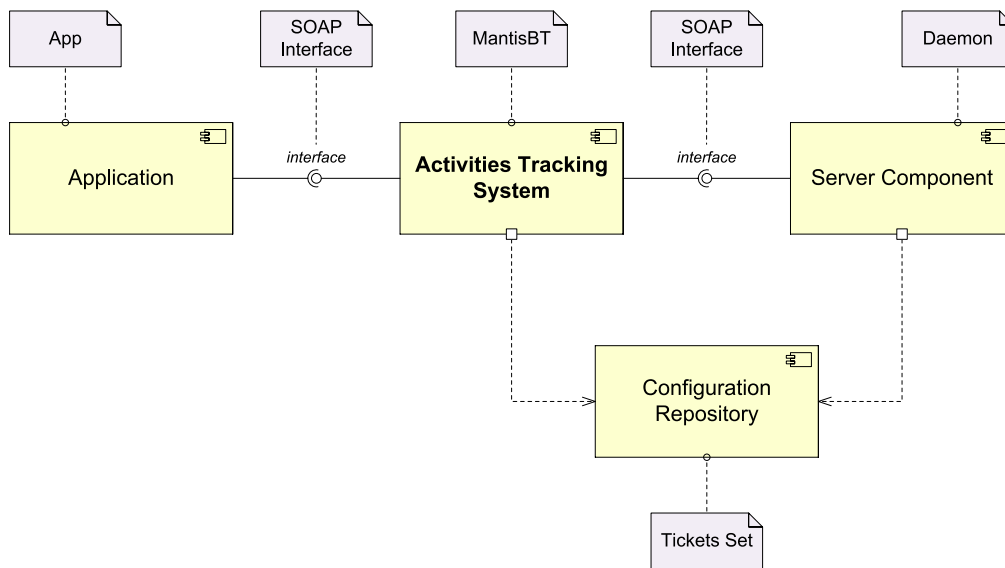


Figure 5.1: Components diagram of the entire project.

language”; I would say, the same *existing* language. This is in line with a precise strategy of software integration.

Although the *Application* and the *Server Component* are modules which operate with the same System, they are completely independent: more precisely, each one exists even if the other does not work, although each one handles data that are used by the other.

An operative example clarifies this concept: through the *Application* a user - with administrative permission - can close a ticket that represents an activity. As mentioned in Section 2.1.1 when a ticket is marked as closed this means that the ticket is completely closed and that no further actions are required on it: this is a completely independent action performed by the *Application*. However, the action affects the server-side workflow. The *Server Component* in fact, performs a periodic evaluation on closed tickets.

Figure 5.2 illustrates the structural independence of the two main operative modules.

The deployment diagram depicts where the developed components reside. It is interesting to notice that the *Server Component* - named *Daemon* in the diagram - resides in the same machine of the Activities Tracking System. As already explained, the configuration repository, consisting of a set of special tickets resides inside MantisBT. Finally, the *Application* works on the user mobile device.

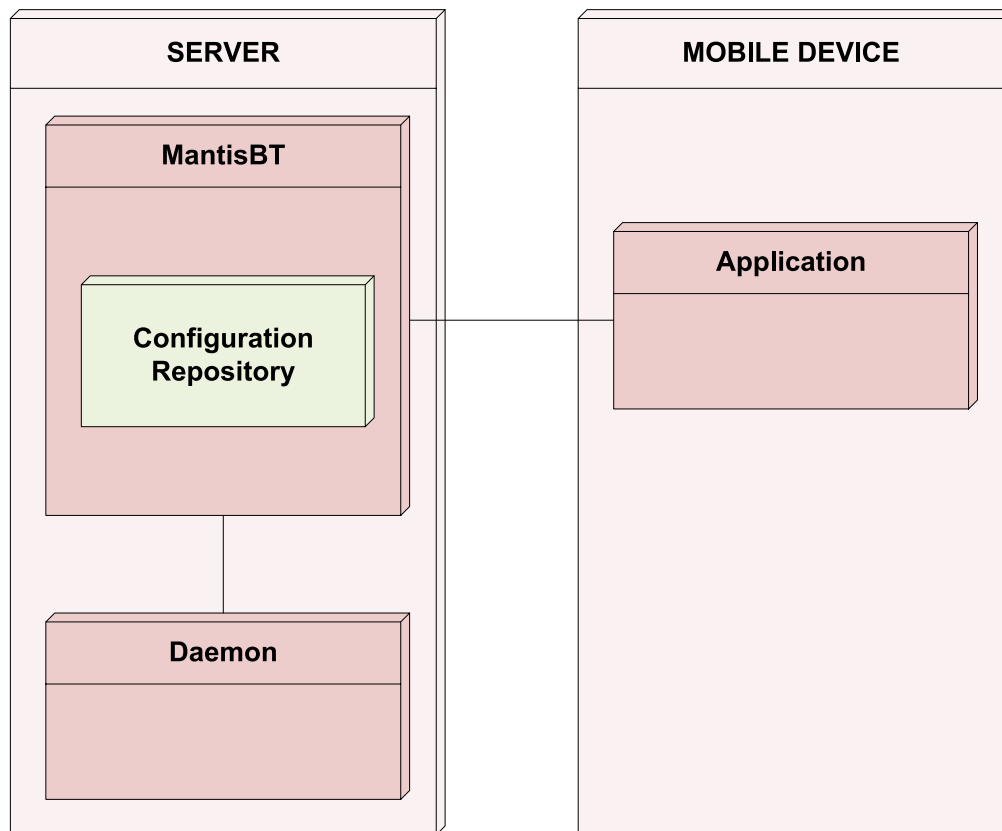


Figure 5.2: Deployment diagram of the entire project.

Before focusing on the two main components - the *Application* and the *Server Component* - it is important to delineate the interaction structure, as a first step towards a more detailed description of the communicative and operative modules. For this intent, Figure 5.3 and Figure 5.4 illustrate two sequence diagrams related to the *Application* and to the *Server Component* respectively.

As regards the *Application*, the main operations performed are:

Initialization where the *Application* retrieves the service exposed by the MantisBT system and invokes a proper method to get all the tickets with status *resolved* and *new*;

Ticket Closure where the *Application* updates a *resolved* ticket to *closed* on the user's will;

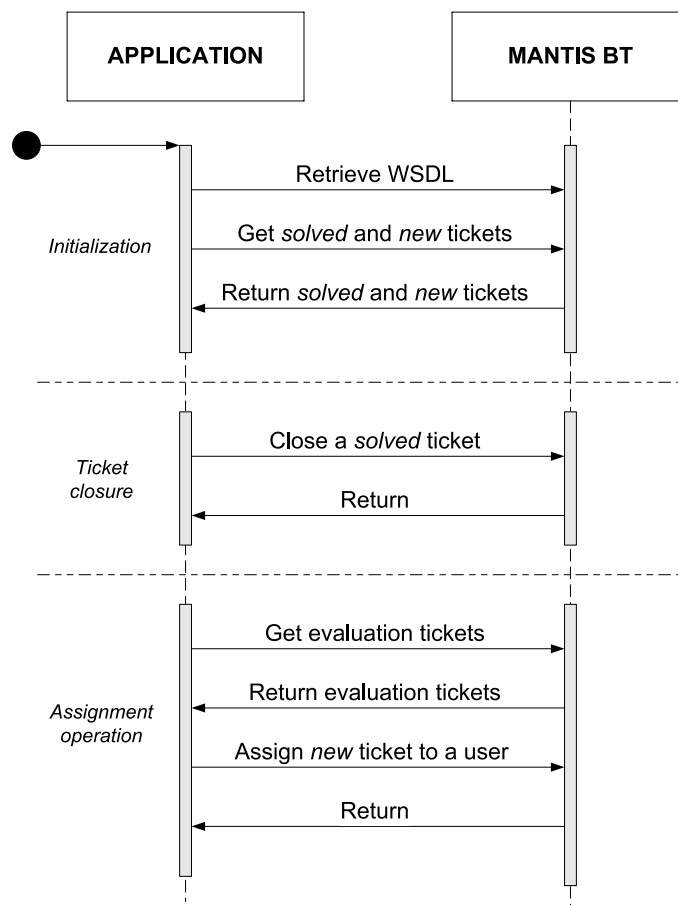


Figure 5.3: Sequence diagram of the *Application*.

Assignment where, after retrieving all the special tickets which store the evaluation values, the *Application* assigns a selected *new* ticket to a selected user, on the user's will.

The last two operations can be performed in either order and as many times as desired, till the end of all available tickets.

On the other hand the *Server Component's* main operations are:

Initialization where the *Application* retrieves the service exposed by the MantisBT system and invokes a proper method to get all the tickets with status *closed*;

Evaluation where the *Server Component* computes all the evaluation values and performs update or creation actions on the special evaluation

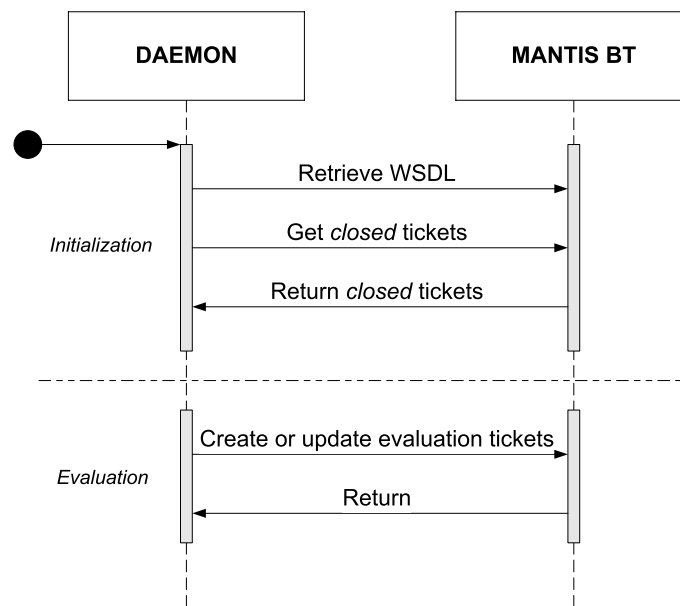


Figure 5.4: Sequence diagram of the *Server Component*.

tickets (details in Section 5.3).

5.2 Mobile Component (*App*)

The *Application* was developed using the PhoneGap technology as exposed in Chapter 4. In this section, the functional analysis introduces all the requirements of the applicative component; and later the technical analysis underlines all the details adopted in reference to the functional specifications.

5.2.1 Functional Analysis

As regards the operations required, the applicative module is a component which allows a user to perform two operations (in accordance with the sequence diagram of Figure 5.3):

1. evaluating and closing a resolved ticket;
2. classifying and assigning a new ticket to a specific user.

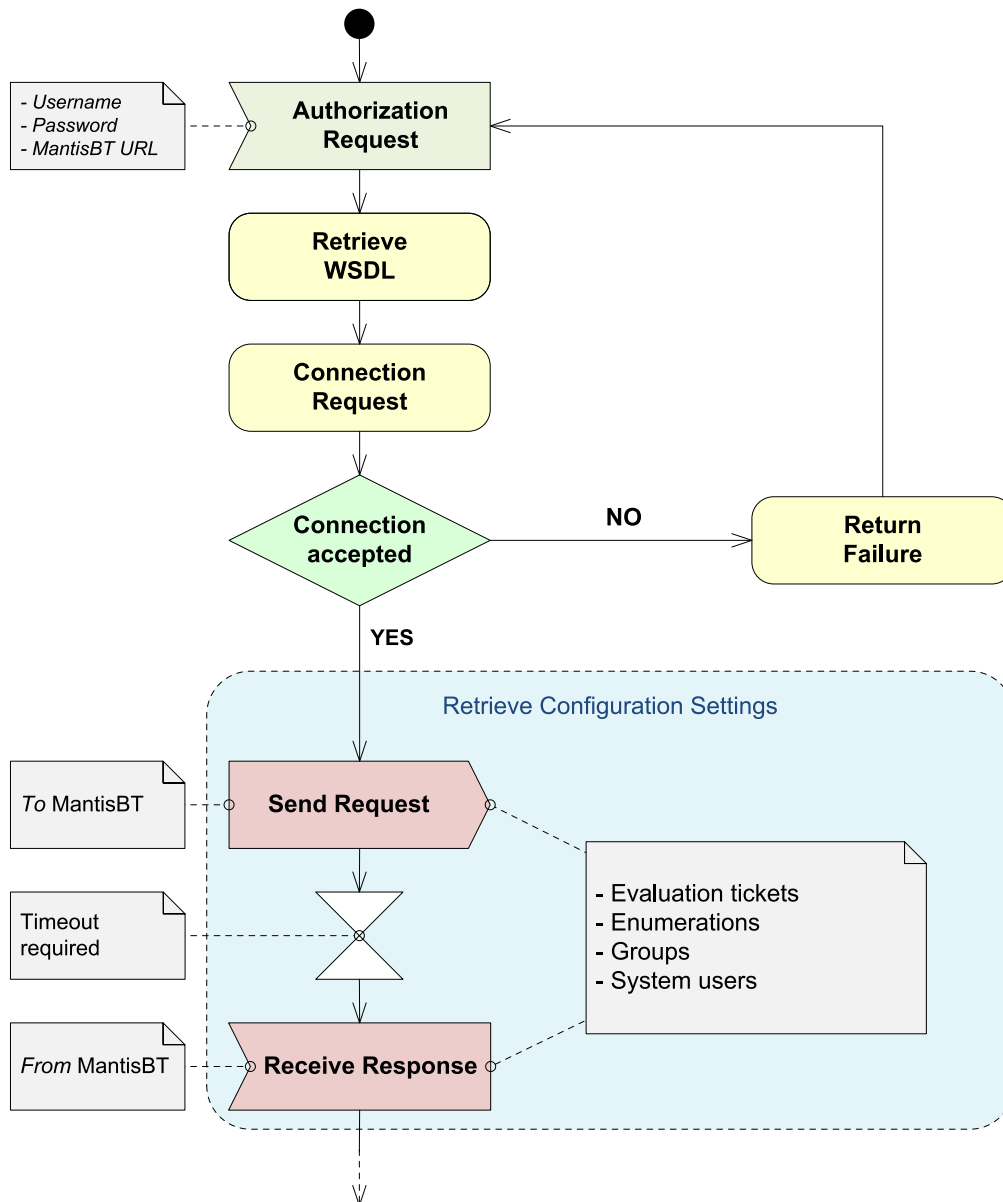


Figure 5.5: Activity diagram for the initialization operations.

There is a common set of actions for both these operations as is explained in the activity diagram of Figure 5.5:

Authentication The first action the user performs is authenticating in the system. Once authentication has taken place, the user will perform a specific set of operations.

Web Service connection The *Application* connects to the endpoint of the Web Service: all the methods exposed by the SOAP interface are ready to use.

Configuration The *Application* retrieves configurations information that is used to set up the application environment.

Evaluating and Closing resolved tickets

As regards the operations of evaluating and closing tickets, these are intimately related to the ticket evaluation system which allows the administrator to classify and evaluate the tickets.

List resolved tickets The *Application* lists all the tickets that are in the *resolved* status;

Display ticket The *Application* shows all the details of a selected ticket;

Classify If a selected ticket has no classification - a common situation for all the tickets created before the evaluation system adoption - the function allows the user to classify it.

Vote and Close The *Application* allows a user to express an evaluation for a selected ticket. Once the ticket has been evaluated - and classified - its status can be set to *closed*.

Classifying and Assigning new tickets

The operations of classifying and assigning a new ticket to a specific user follow these steps:

List new tickets The *Application* lists all the tickets that are in the *new* status;

Display ticket The *Application* shows all the details of a selected ticket;

Classify This function allows the user to classify the ticket.

List users The *Application* lists all the users that can handle a ticket with the selected classification: it provides a set of filters which allows the user to focus the assignment procedure on specific indicators.

Assign to a user The *Application* makes it possible to assign a selected user to the ticket.

5.2.2 Technical Analysis

This section deals with the technical structural details of the *Application*. The main modules which are described are:

- the module for communication and interaction with the web service;
- the structure and flow of the application body.

Web Service communication

AJAX is the acronym of “Asynchronous JavaScript and XML”, a technology based on `XMLHttpRequest`, which is now supported by all main browsers. The basic idea is quite simple - and not actually a breakthrough - but it allows updating a page following a server request, without reloading the entire set of data. The environment is composed of:

1. a Web Service with the required methods on the server-side;
2. the *Application* which uses the WSDL (Web Service Description Language) to automatically generate a JavaScript proxy class so as to allow using the Web Service return types.

The diagram of Figure 5.6 shows the SOAP *Application* workflow for asynchronous calls. The *Application* invokes the `SOAPClient.invoke` method using a JavaScript function and specifying the following:

- Web Service URL,
- Web method name,
- Web method parameter values,
- Call mode (asynchronous or synchronous),
- Callback method invoked upon response reception (optional for sync calls).

The `SOAPClient.invoke` method executes the following operations (numbers refer to the diagram of Figure 5.6)

1. It gets the WSDL and caches the description for future requests,
2. It prepares and sends a SOAP request to the server (invoking method and parameter values),

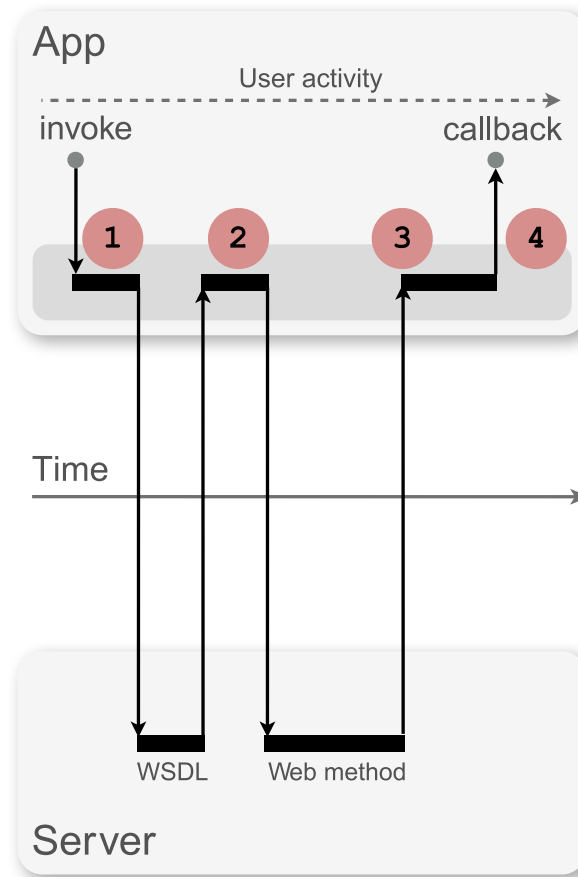


Figure 5.6: *Application* workflow for asynchronous calls.

3. It processes the server reply using the WSDL so as to build the corresponding JavaScript objects to be returned,
4. If the call mode is async, the Callback method is invoked, otherwise it returns the corresponding object.

Application structure

The structure of the PhoneGap project is composed of the following parts:

index.html contains the main skeleton of the application. All the pages of the *Application* are declared in the same file, in line with the jQuery Mobile declarative strategy (Section 4.2);

SOAP client library contains the implementation in Javascript of the module which has previously been presented: this library allows the application to invoke the Web Service methods.

Javascript classes These comprehend the methods for the retrieved data management, and subsidiary methods for improving the user's graphic experience (pleas refer to details in Appendix A).

Before describing the operations performed, it is useful to analyze the structure of the main objects of the interactions:

- the Ticket,
- the Group,
- the User.

Table 5.1 presents the attributes of the objects' structures that were used in the application development.

Object	Attribute	Description
Ticket	id	Unique identification number
	view_state	The viewing status of the ticket (public or private)
	last_updated	The date of the last update
	project	The group which the ticket belongs to
	category	The End Customer for the ticket
	priority	The priority of the ticket
	severity	The severity of the ticket
	status	The status of the ticket
	reporter	The user who has posted the ticket
	summary	The summary of the ticket
	reproducibility	The reproducibility of the ticket
	date_submitted	The date of the submission
	handler	The user who handles the ticket
	description	The description of the ticket
	custom_fields	The fields that have been added for the customization
	due_date	The alert date
	tags	The tags associated with the ticket
notes	The notes added for the ticket	

Object	Attribute	Description
Group	id	Unique identification number
	parent_id	The id of the father group
	last_updated	The date of the last update
	name	The name of the group
	status	The status of the group
	view_state	The viewing status of the group
	access_min	The minimum access level to access the group
	description	The description of the group
	reporter	The user who has posted the ticket
	subprojects	The subgroups children of the group
User	id	Unique identification number
	name	The username of the user
	real_name	The real name of the user
	email	The email of the user
	access	The global level access of the user

Table 5.1: Main objects' structures.

All the methods (exposed by the SOAP interface) adopted to perform the *Application's* operations are:

`mc_project_get_users` Returns all the appropriate users assigned to a specified group;

`mc_project_get_issues` Returns all the tickets that match the specified group;

`mc_enum_access_levels` Returns the enumeration for access levels;

`mc_enum_status` Returns the enumeration for statuses;

`mc_projects_get_user_accessible` Returns the the list of groups that are accessible to the logged in user;

`mc_issue_update` Updates a selected ticket.

5.3 Server Component

The *Server Component* is a module which has the task of processing all the closed - therefore already classified and evaluated - tickets of the Activities Tracking System, and constructing performance indicators of the system. This new evaluation module entails the presence of *five* new custom fields that must be added in the System configuration section.

Three of them act as classification fields; in fact, the following indicators compose the classification tern which is used to classify the tickets:

- Product
- Region
- Technology

Every activity which is tracked by a ticket falls on a specific value for each field. An *Evaluation* value is set as custom field and shows the evaluation that has been assigned to the ticket. Lastly, a *Closure Date* custom field stores the date of the ticket closure.

5.3.1 Functional Analysis

As regards the functionalities that the *Server Component* provides, these were designed in order to obtain a specific classification of the activities performances. The purpose is to provide a useful evaluation scheme when a new activity has to be assigned to a user. Thus, the past of the company is no longer ignored in the logic of the system.

The main operations that the *Server Component* performs are:

- retrieving the Web Service exposed by MantisBT installation;
- retrieving all the evaluation values;
- retrieving all the closed activities;
- for every activity - and each one has a handler - storing three evaluation values for each couple

```
handler - product_value  
handler - region_value  
handler - technology_value
```

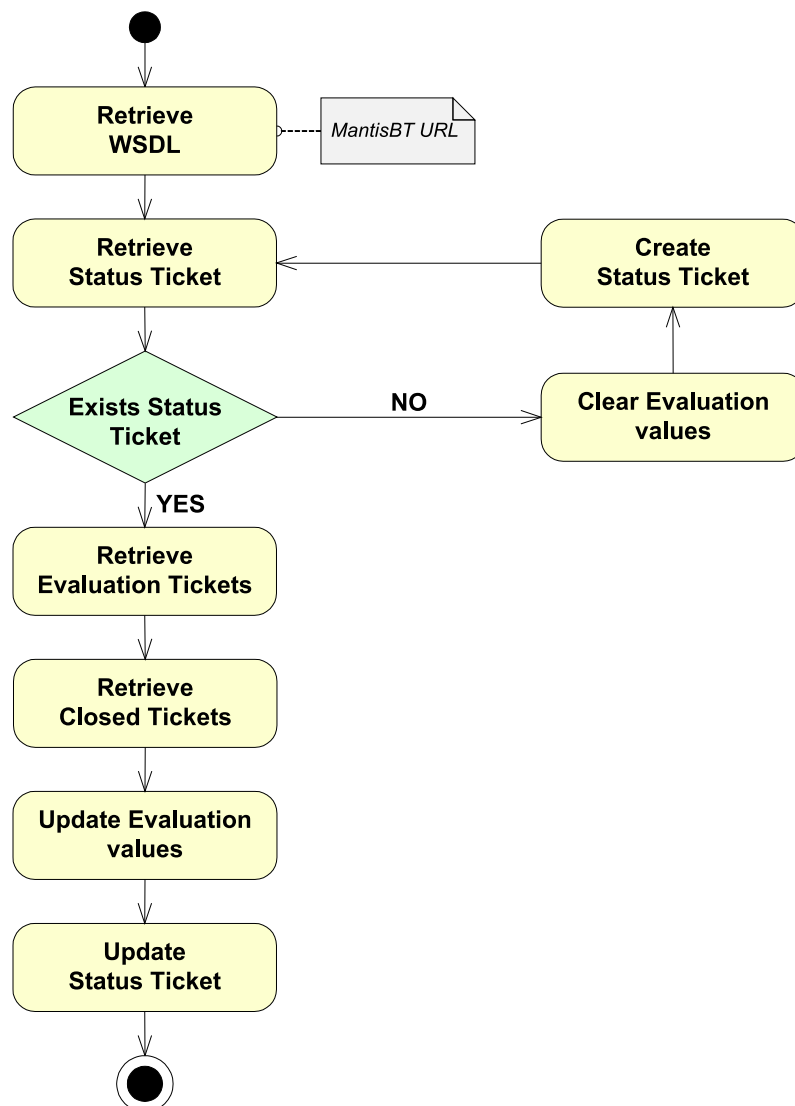



Figure 5.7: General structure of the *Server Component*.

- constructing a global table which stores all the evaluation averages;
- providing the possibility to perform both an incremental table update and an entire re-analysis from the ground;
- the evaluation table must be available for the *Application*.

A *Status Ticket* storing a last-update date determines the update level to perform. If it is present, the updating is computed from the last update.

Otherwise the whole evaluation table is reprocessed.

Figure 5.7 illustrates the general structure of the operations performed by the *Server Component*.

5.3.2 Technical Analysis

The component is written in Java language and is designed to periodically perform a complete evaluation. This detail loosens the concurrency and synchronization constraints: it was programmed to work in background - as a daemon service acts - and to collect once a day all the system tickets in order to create or update the evaluation table.

Technically speaking, a useful library was adopted to facilitate communication with the MantisBT installation. This library is

```
mantis-axis-soap-client-1.2.0.jar
```

It is composed of SOAP stubs generated using Axis for the MantisBT SOAP API with a special script which is *wsdl2java*. This tool takes a WSDL document and generates fully annotated Java code from which to implement a service. The WSDL document must have a valid portType element, but it does not need to contain a binding element or a service element. Using the optional arguments it is possible to customize the generated code. In addition, *wsdl2java* can generate an Ant based makefile to build an application.

Axis is an implementation of the SOAP protocol. It shields the developer from the details of dealing with SOAP and WSDL. Axis is used on the server side to write the web service, and it is used on the client side to facilitate the writing of the client. Axis is essentially Apache SOAP 3.0. It is a from-scratch rewrite, designed around a streaming model (using SAX internally rather than DOM).

The aforementioned library provides all the structures described for the *Application*, which are ready to use with an object-oriented approach. This useful feature facilitated data manipulation both in the requests and in the responses to and from the MantisBT installation, leading to a satisfactory communication level.

```
1 ...  
2 try {  
3     URL url = new URL(<MantisBTinstallPath >);  
4  
5     MantisConnectLocator mcl = new MantisConnectLocator ();
```

```
6   MantisConnectPortType pt = mcl.getMantisConnectPort(url);
7
8   pt.<MantisBTMethod>(parameters);
9 }
10 catch (Exception e) {
11     e.printStackTrace();
12 }
13 ...
```

Listing 5.1: Skeleton of a request to the web service.

A generic request to the web service is done with the following steps (see Listing 5.1):

- instance creation of `MantisConnectLocator` class;
- acquisition of the endpoint to communicate with the web service;
- method invocation;
- returned data management.

It's important to correctly handle the exception during these operations.

The way the *Server Component* operates starts from the creation of three specific and private subgroups within a father private group with name *Resources Evaluation*. The three subgroups are:

- Region,
- Product
- Technology

These subgroups are populated with *Evaluation Tickets* (a ticket of this group is referred with t_e) which are updated according to the following policy (the explanation has considered the instance with the *Region* subgroup).

The *daemon*:

- retrieves all the closed tickets of MantisBT and for each ticket (referred with t):
 - search in the subgroup Region for a ticket t_e which matches the `Handler-RegionValue` pair of t ;

- if exists, updates it;
- otherwise creates a new ticket t_e with such a values pair.

The updating operation deserves a deeper consideration; all the t_e tickets have two custom fields:

Cumulative Evaluation Value - CEV represents the summatory of the already precessed evaluations;

Number of processed tickets - N corresponds to the number of the already processed tickets.

For the new average value computation - which corresponds to the updating operation -, given a new evaluation value v it's sufficient to perform

$$\frac{CEV + v}{N + 1}$$

Finally the Evaluation system provides the possibility to “clear” the past of a user for a specific value of one of the three subgroups: this is the situation where - for example - a user has followed a training course and all the evaluations from that time forth are no longer useful. Since the system is implemented on the ticket policy, this case is covered by simply closing the aforesaid ticket - the one that is no longer reliable - and creating a new ticket which now is the one to act as real performance indicator.

Chapter 6

Final Considerations

This Master thesis work represented an interesting opportunity of collaboration with a real company environment.

A rewarding aspect is that the developed tools can be effectively adopted by the company. Another important aspect is that the efforts that were made can be useful for a future improvement of the work, which can be handled by some other colleagues or company employees.

In this regard, here is an interesting improvement that can be the starting point for an extension process:

- create a module for the *App* which manages the notification that are thrown from MantisBT; at the moment, the notification structure is implemented with e-mails. The use of an XMPP server may improve this notification service.

Appendix A

Graphic Details

The development of the *App* produced an acceptable result in terms of both user experience and graphic rendering. This second aspect was made possible by the use of the jQuery Mobile library which is described in Section 4.2.

In this appendix chapter all the main views of the application are reported: the graphic realization follows the executive flow expressed and analyzed in Section 5.2.

The description illustrates the main actions that can be performed with the *App*: the scenario which the component interacts with is a test MantisBT installation with the same characteristics and settings of the Activities Tracking System present in Mida Solutions. Fictitious tickets were added to the system in order to simulate the company environment; however, the groups and all the displayed fields are modeled on the company reality.

A.1 Login and Home pages

The Login Page is the first page that is displayed to the user (Figure A.1): this is the starting point for the interaction with MantisBT. The fields that are required are:

Server URL is the server address where MantisBT is installed;

Username is the same username that the user adopts to log in MantisBT via web interface;

Password is the password associated to the username.

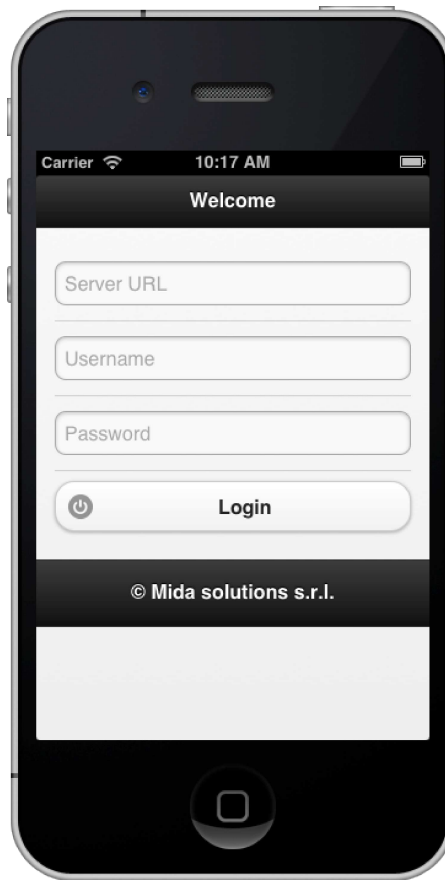


Figure A.1: Login page.

After the log in page, the *App* automatically retrieves the web service interface and performs the data requests to the server. The access level management of MantisBT is directly reflected in the *App* access level policy. The set of actions that a user can perform via web interface, corresponds to the actions that can be performed with the *App*.

Hence, the page that is displayed (Figure A.2) presents two buttons which correspond to the two main actions that the applicative component provides. These actions are widely described in Section 5.2 and they are related to operations on:

Resolved Tickets This section provides the functionality of classifying, evaluating and closing a resolved ticket;

New Tickets This section provides the functionality of classifying and

assign a new ticket.

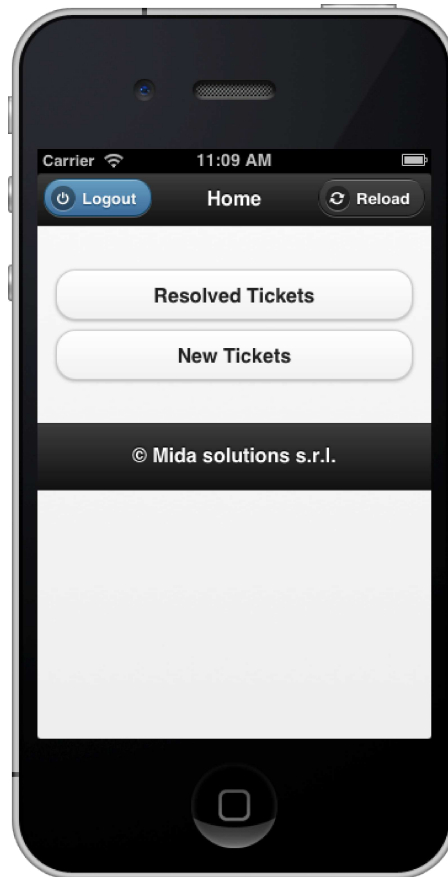


Figure A.2: Home page.

The header bar presents two buttons:

Logout This button allows the logged in user to be logged out from the system: the page that will be displayed is the Login Page;

Reload When the user wants to reacquire the tickets - this because perhaps some changes happened on the server side - uses this button. As described in Section 5.2.2 the WSDL is cached, so this operation takes a time that is reasonably short.

A.2 Resolved Tickets pages

By selecting the first button in the Home page all the resolved ticket - grouped by company subgroups - are listed (the company group where this operation acts is the *Customer Care* group; however the *App* is structured that this is a settable parameter - i.e. other groups can be analyzed, and/or more that one at the same time).

Figure A.3 shows the described page; this is a clickable and scrollable list. For each subgroup the tickets are sorted by last modification date. The content that is displayed on a single ticket line is settable; in the example there are the id, the summary, and the submission date.

This introduces a feature, related to the orientation graphic management - allowed by jQuery Mobile library. In fact this library independently manages the graphic adaptation to the screen size and orientation. Thus, in this page, the horizontal orientation allows to view more ticket details; the tablet screen would show still more.

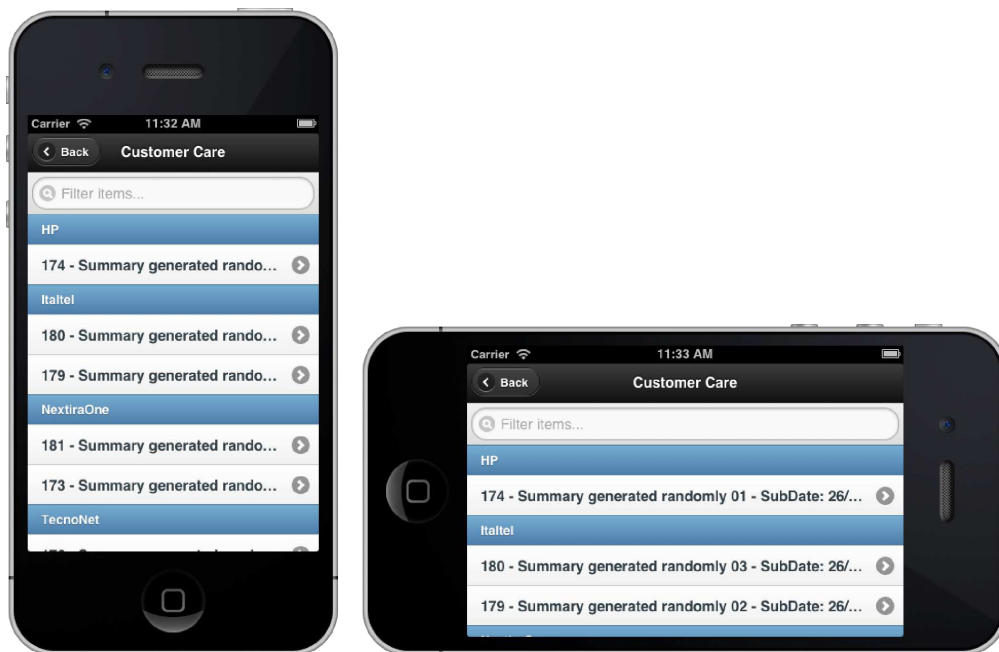


Figure A.3: Resolved tickets page.

It's interesting to notice that the "Filter" bar is a useful feature that is completely managed by the jQueryMobile library.

After selecting a ticket (in this example the ticket with id 174 has been chosen), a page with all the ticket details is shown; also in this page, the

content is organized with a scrolling list. Unlike before, now the footer bar is fixed. This bar presents two buttons:

Evaluate allows the user to assign an evaluation to a classified ticket; this feature also gives the possibility to change the actual classification of the ticket;

Close allows the user to update the status of the ticket to close; this action can be performed only if the ticket has a classification (a ticket with evaluation 0 is considered as not influent in the evaluation values computation).



Figure A.4: Resolved ticket details page.

Figure A.4 shows the described page; the illustration on the right shows the page contents after a scrolling action: the last information is related to the classification and evaluation values.

A.2.1 Evaluate

By clicking on the *Evaluate* button, a specific page shows the Classification and Evaluation values. Each value can be modified: only after clicking the *Apply* button the changes are applied. If the user does not apply the changes and goes back to the previous page a pop-up message asks for the confirmation.



Figure A.5: Evaluate page.

All the possible values that can be assigned to each field are dynamically

acquired from the server: thus, no values are wired in the *App*. It is interesting to notice that the list of the possible values is displayed with the specific platform layout.

Figure A.5 shows on the left the evaluation page for ticket 174, and on the right the list of the possible values for the Evaluation field.

A.2.2 Close

By clicking on the *Close* button, a pop-up box asks for the confirmation. When user clicks on *Yes*, the status of the ticket is updated to *Closed*; thus the ticket will not be displayed in the Resolved List page anymore. Figure A.6 shows the pop-up box.

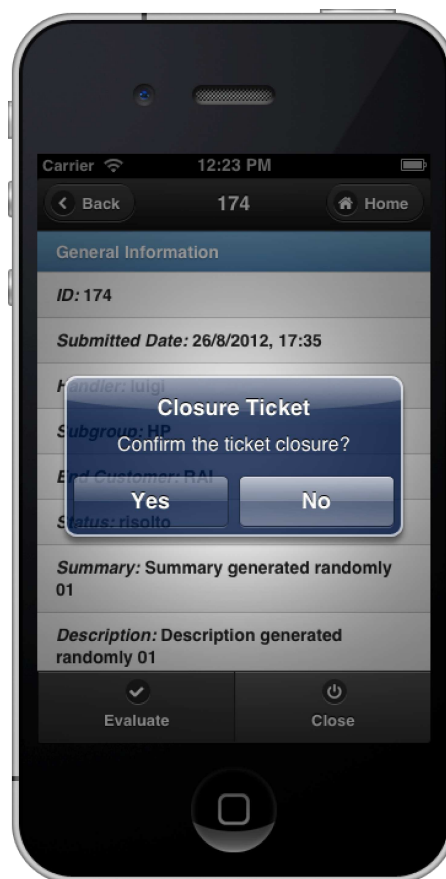


Figure A.6: Close ticket confirmation pop-up.

A.3 New Tickets pages

By selecting the second button in the Home page all the new tickets - grouped by company subgroups - are listed (as in the previous section).

By selecting a ticket (in this example the ticket with id 161 is chosen), a set of information is displayed; no classification or evaluation information appear because these are *New* tickets that need an handler to be assigned.

Figure A.7 shows on the left the list of new tickets, on the right the details of ticket 161.



Figure A.7: New tickets list (on the left) and details of a selected ticket (on the right).

In the second page the footer bar contains a button that allows the user to access to the ticket assignment section.

A.3.1 Assign

By clicking on the *Assign* button, a specific page is displayed. It is structured in three main parts:

- Classification
- Evaluation Table
- Assignment

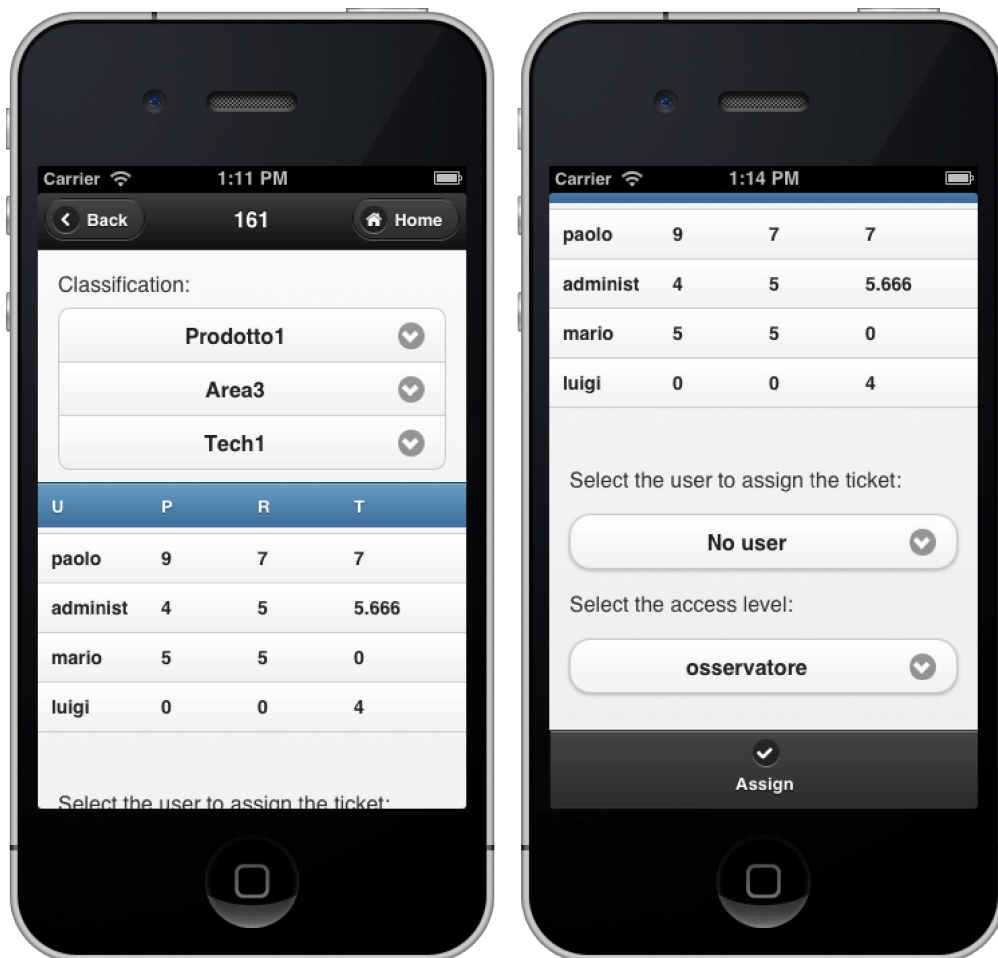


Figure A.8: Assign page.

The Classification section allows the user to choose a classification term of values. After selecting a value, the table in the second section - which

contains the users with the evaluation values - is sorted by decreasing value for the selected field. This means that when the user select a specific value for the region, the table is sorted by the region value; the same for the other fields. The table provides also the functionality of filtering the classification values; the user in fact may want to have a view of only a subset of the entire set of fields.

Figure A.8 shows the described page: by selecting a classification the table sorts all the users. In the view on the right, the third section is shown. This section allows the user to select the handler of the ticket and the appropriate access level to assign.

The *Assign* button permits to apply the selected values to the ticket.

Bibliography

- [1] “Mantis Bug Tracker, a free popular web-based bugtracking system.” <http://www.mantisbt.org>, 2000.
- [2] “SOAP, a simple object access protocol.” <http://en.wikipedia.org/wiki/SOAP>, 1998.
- [3] “Web Services Glossary.” <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>, 2004.
- [4] “Web Services Tutorial.” <http://www.w3schools.com/webservices>, 2012.
- [5] “SOAP Tutorials.” <http://www.w3schools.com/soap>, 2012.
- [6] “Native, web and hybrid apps.” <http://www.xcubelabs.com/blog/native-web-and-hybrid-apps-understanding-the-difference/>, 2012.
- [7] “Hybrid HTML5 Apps Are Less Costly to Develop Than Native.” <http://www.readriteweb.com/mobile/2012/01/hybrid-html5-apps-are-more-les.php>, 2012.
- [8] “Native, web or hybrid mobile-app development.” http://www.mobileconnectevent.com/downloads/white-papers/Mobile_Thought_Leadership_White_Paper.pdf, 2012.
- [9] J. M. Wargo, *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley Professional, 2012.
- [10] R. Ghatol, *Beginning PhoneGap - Mobile Web Framework for JavaScript and HTML5*. Apress, 2011.
- [11] J. Munro, *20 Recipes for Programming PhoneGap: Cross-Platform Mobile Development for Android and iPhone*. O’Reilly Media, 2012.

- [12] J. Marinacci, *Building Mobile Applications with Java: Using the Google Web Toolkit and PhoneGap*. O'Reilly Media, 2012.
- [13] T. Myer, *Beginning PhoneGap*. Wrox, 2011.
- [14] G. Bai, *jQuery Mobile First Look*. Packt Publishing, 2011.
- [15] M. Doyle, *Master Mobile Web Apps with jQuery Mobile*. Elated Communications Ltd., 2011.

List of Tables

2.1	Nomenclature changes adopted	5
3.1	Mobile OSes comparison	24
3.2	Approaches comparison	30
4.1	Mobile Browsers	37
4.2	Terminal commands for creating a new iOS Project	46
5.1	Main objects' structures.	59

List of Figures

2.1	Diagram of the status transitions.	6
3.1	MantisBT logo	10
3.2	Native Approach structure	26
3.3	Web Approach structure	28
3.4	Hybrid Approach structure	29
4.1	PhoneGap executive flow.	35
4.2	PhoneGap application architecture	37
4.3	New Android Application project.	42
4.4	Click Finish to create the project.	43
4.5	Edit the project's Java file.	44
4.6	Edit the Android Manifest.	45
5.1	Components diagram of the entire project.	50
5.2	Deployment diagram of the entire project.	51
5.3	Sequence diagram of the <i>Application</i>	52
5.4	Sequence diagram of the <i>Server Component</i>	53
5.5	Activity diagram for the initialization operations.	54
5.6	<i>Application</i> workflow for asynchronous calls.	57
5.7	General structure of the <i>Server Component</i>	61
A.1	Login page.	68
A.2	Home page.	69
A.3	Resolved tickets page.	70
A.4	Resolved ticket details page.	71
A.5	Evaluate page.	72
A.6	Close ticket confirmation pop-up.	73
A.7	New tickets pages	74
A.8	Assign page.	75