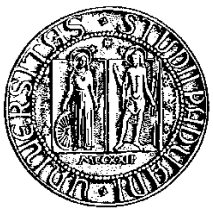

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingeneria Informatica

Analysis and implementation of social networking methodologies in a quality management system

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

Laureando: Alberto Rubin

Contents

Chapter 1

Introduction

The company where the thesis work was developed is Mida Solutions S.r.l. . The company was founded in 2004, by a team of telecommunication experts, with the mission to provide value added innovative technologies for communication. Mida focus has been since the beginning technology and people. Mida Solutions provides unique expertise and a complete suite of Voice Applications and Value Added Services, with the goal to definitely improve the Telephony Infrastructure functionalities.

In March 2012, DNV has officially certified Mida Solutions Quality System in accordance with ISO 9001. The certification of the management system demonstrates the continuous effort of the company in activating processes of continuous improvement, aiming at developing the business, in a logic of sustainability.

For a company insert a new product/service into the market is a significant burden from both the organizational and financial aspects. This phase, however, is only the end of a very complex process. In general it is possible to decompose this process in a succession of stages, the choice of which depends on many factors. Without going into detail, we can say that, for the success of the company, it is necessary that each of the activities identified is carried out in order to maximize the probability of success and, simultaneously, reduce time and costs.

Imagine, for example, to structure the process in five-step: marketing, design, manufacture, production and sale. Editing a project involves a cost that depends on the extent of correction needed and the time when this decision is taken, that is, depending on the stage reached in the process for the construction of the new product: the amount required for a change made in the early stages of the design is significantly less than that required for the modification of a product already on sale.

The process of inserting a new product in the market must take into account that is the customer who determines success or failure. A detailed design and implementation of a good product are therefore insufficient, but it is essential

that the company, at every stage, takes into consideration the needs of the customer and achieving what he expects. In conclusion, this can be translated in bring the customer's voice and let it guide you throughout the development process, from design to production of the good or service delivery.

With the coming of Web 2.0, in particular through to the spread of social networks, the level of interaction between web and user increased significantly. Observing this increase, companies have been able to use these new tools to establish direct communication with its customers in order to have instant feedback on every product or service, and thus be able to evaluate and possibly modify their strategies according to the level of satisfaction manifested. Such tools are increasingly taking more significant importance because they have become the new way through which the voice of the customer can be collected by the company.

But how to use them to extract the level of customer satisfaction? This thesis will attempt to provide an answer to this question.

Chapter 2

Part I - Customer Satisfaction & Social Networks

2.1 Social Network

If we were to provide a trivial definition of the phenomenon of social networks, we can say that a social network is a connection between people with different relationships: work, friendship, family. The Internet version of social networks amplifies the concept of sharing and participation and develops one of the most advanced forms of communication based on the design of relational maps. The construction of these relationships are structured in the manner of the so-called "web 2.0", result of the design of web sites and applications that put the content generated by the user or not, in the hands of the consumer.

Greatly simplifying the meaning of social networks, we can imagine it like a complex platform where you can create a network of people with whom you are in contact (at different levels) sharing information and ideas with them and following those they enter. In practice it is possible to have a profile (a card with information about who we are), you can insert messages (but also multimedia content like photos, videos and links to web pages) and read, comment and share what other users publish. You can usually tell the system who has or hasn't the ability to read our messages and information that we publish. In conclusion, a social network is a powerful tool to keep in touch with people, to learn and to inform.

One of the reasons that drives a company to use social networks is the need to listen what people say about the company and see which is the brand image among customers or users in general.

The choice is never speak whether or not a specific brand, product or service, but to participate or not in a conversation. One of the biggest mistakes made by marketing responsible and by IT manager is to limit themselves to the creation of a new account. In fact, the use of social networks as a tool to further expand

its business should be translated as the ability to listen to what others say about the company, what are the perceptions that others have of its market and what competitors do.

In this chapter we will analyze the characteristics that have made famous the main social networks in order to identify and analyze the features that can be used for the extraction of the level of customers satisfaction. The best known and most used social network is Facebook, followed by the no less important Twitter and Google+.

2.1.1 Facebook

Facebook is a social network launched in 2004, owned and managed by the corporation Facebook, Inc. The site is free and takes profit primarily from personalized advertising for each user, offering products and services which are coherent with the activities performed by the user within the social network.

The interesting aspects of this social network are the following:

- social marketing on the platform. One of the main strengths of the social networks are the tools of social marketing that allow planning of real advertising campaigns targeted to certain categories of customers. The most important of these tools is Facebook Ads.
- analysis of campaign results. In addition to making possible the realization of advertising campaigns, Facebook provides the tools necessary for the analysis of the results of these campaigns.
- integration with external sites. You can use your Facebook account to interact with websites that have integrated inside itself the social network. This platform is called Facebook for Websites.
- openness to developers. Facebook has made available to the developers the APIs that allow the creation of applications.

Figure 2.1: Logo di Facebook.

Facebook Ads

"With Facebook Ads, users can learn about brands and businesses through trusted referrals from their friends on Facebook. Advertisers can connect with users by creating a presence on Facebook and targeting the exact people they want"

The presence on Facebook is a strategic element for companies that want to exploit the full potential of Web 2.0 for their business: the most popular social network now collects a pool of over one billion users worldwide with a growing trend. The creation of official pages for the company and discussion groups on specific topics that achieve indirectly a larger number of users, are among the main drivers of what is called social marketing.

For a company be present in Facebook has a number of advantages, the main ones are:

- increase knowledge of its brand
- update the clients on products, promotions and events
- promote their business indirectly through aggregation themes
- get feedback from the fans to know their opinions and their needs
- take advantage from viral marketing, encouraging fans to involve their friends

In the official press release of the service were cited some large companies that have seized the new opportunities opening up and promoting a fan page on Facebook in order to emphasize the importance of the tool for business marketing.

The ads on Facebook appear always in the advertising space located to the right of the page, and are divided into the following elements:

1. Body of the ad, consisting of title, image and brief description.
2. The names of the friends who are already fans of the page advertised.
3. Link to the page to become a fan or external link to the website being advertised.

Through the use of Facebook Ads you can define the exact characteristics of the recipients of the insertion, by setting up demographic or psychographic filters based on the data shown on the subscribers profiles. The parameters to choose from are numerous and allow a considerable depth in targeting.

- **Gender, age, level of education, sentimental situation and sexual orientation**.
- **Keyword**, calculated based on the profile information such as interests, activities, books, favorite movies, etc.
- **School or University**.
- **Connections**, allows you to contact those who are already fans of the page, or who use a particular application, or who has joined a certain event.
- **Friends of connections**, allows you to contact only those who are friends with those who are already fans, or who use a particular application, or who have joined to a certain event.
- **The birthday**, displays the ad on the user's birthday.
- **Geographical Location**, Facebook determines a user's location by IP address, but you can also use the address that the user has possibly included in his profile.

One of the drawbacks of this approach can be identified, in addition to the real reliability of the profiles, even in the actual completeness of these.

After having identified the characteristics of the recipients of the advertising message, Facebook is able to provide, in real time, a rough estimate of the number of users who will see the ad. In this way it is possible to realize the actual views that can be obtained with certain filters and if too restrictive, change them to broaden the target audience.

Facebook Insights

From the definition of the mechanism that regulates the ads, it can be said that the objectives of web advertising on Facebook can be mainly:

1. Divert users on its website.
2. Increase the number of fans of the page.

Start an advertising campaign through the social network is a necessary step, but not sufficient to ensure the spread of the knowledge of your product or service. Indeed, it is necessary to monitor the achievements of the campaign.

To make this possible, Facebook offers companies a structured analysis tool that allows you to measure the exposure, the actions and behavior of users in relation to their Facebook page. The data that Facebook Insight allows you to check are:

- **information about fans**: number, gender, geographical distribution.
- **interactions**: total number of comments, wall posts and preferences.
- **interactions for post**: average number of comments, wall posts and appreciation generated by each content.
- **quality of the post**: score that measures the interest of the contents of the page by Facebook users.
- **discussion post**: the number of discussion topics created by the users on the page.

Facebook Insights summarizes the general trend of the listed variables, allowing also to deepen the trend data and the types of interactions, and generate graphs of detail for each item that you want to analyze. An example of a screen obtained through the use of Facebook Insights is reported in Figure 2.2.

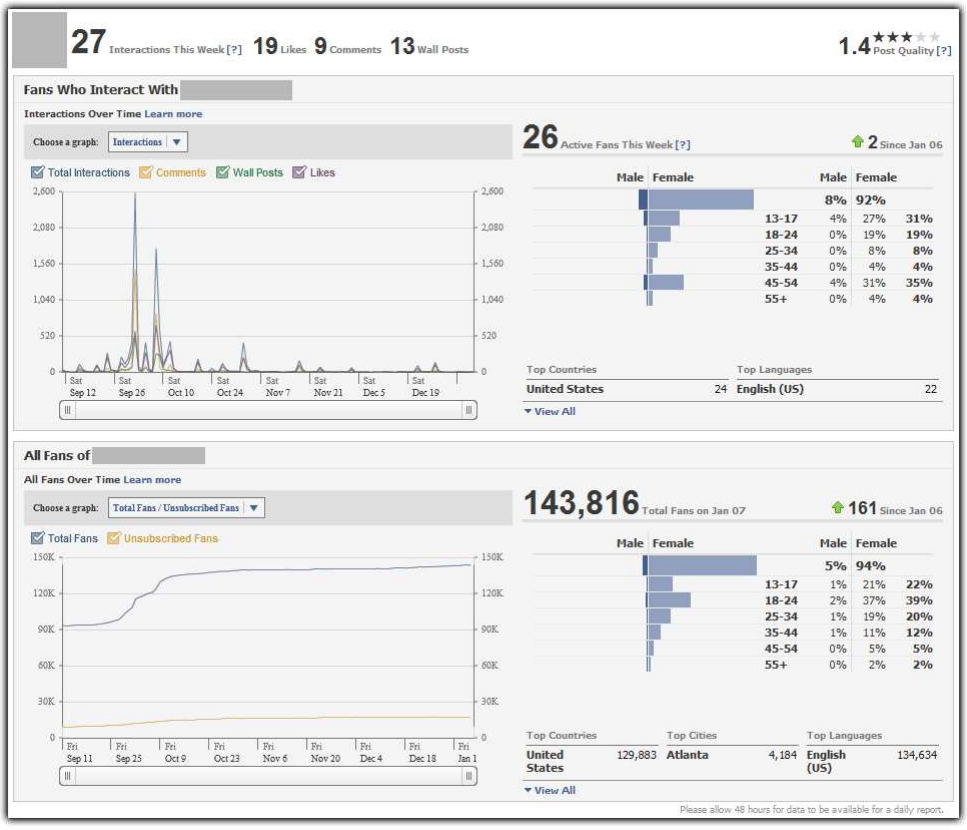

Figure 2.2: Screen example of Facebook Insights.

With this powerful tool is possible to analyze in more detail the quantity and quality of interactions occurred on the company profile, noting the level of involvement exercised by post on the fans, and then ensures the success of the entire social marketing strategy adopted.

2.1.2 Google plus

Google+ is the social network of the famous search engine launched June 28, 2011. It has a number of features that distinguish it and that allowed him to grow quickly despite the already established competition.

Figure 2.3: Google plus logo.

The main features offered are:

- Circles. They are one of the most important innovations introduced by the social network that allows to share relevant content with the right people and find content in which you are interested.
- +1 button. It is used to report publicly a content appreciated by the user.
- Hangouts. This feature allows you to make videoconference of up to 10 people.
- Chat. It allows you to exchange information between multiple people, in real-time.

Below we briefly discussed the possible uses of the circles and the +1 button in the corporate context analyzed in order to verify the usefulness of their use to achieve the prefixed objectives.

The circles

"Circles make it easy to share with the right people"

The main purpose of the Google+ circles is to manage groups of people according to the real social connections of life. It is useful to separate the family members, co-workers and friends in order to allow targeted communications between people who share interests or experiences. The grouping of contacts allows

then to decide to whom make known certain content and those whom keep them hidden. In a business context this instrument can be used in various ways, for example:

- Share specific communications between the contacts that have in common a distinct feature.
- Contact directly their contacts with questions and surveys to obtain information on the basis of which create the circles. Based on the answers to the question ¨What product do you use?¨You can, for example, add the user to the circle containing all the users that use that particular product.
- Create circles based on the role played by contacts in the company (for example executives, customers, employees). This allows you to share information and carry out discussions on specific topics of interest to users and allow you to gain new information about them.
- Involve contacts associated to customers to get feedback, interact with them and know them better.

Figure 2.4: Circles in Google plus.

Analysis of functioning Summarizing the functioning of the circles we can state that this instrument takes care of showing the contents only to users who belong to a certain set and consequently hides them to all the others.

From the implementation side this can be translated in the association between user and circle through the use of a special pointer to each user in the structure representing the circle. In fact, the circles are nothing more than a

list of users identifiers, used for targeted shares: When you decide to share content with a particular circle, you go to set the visibility of that content only to members of the circle. Exploiting such structure is also possible to dynamically locate the circles of which a given user is a member, verifying his presence in each of them. The scheme of such mechanism is shown in Figure 2.5.

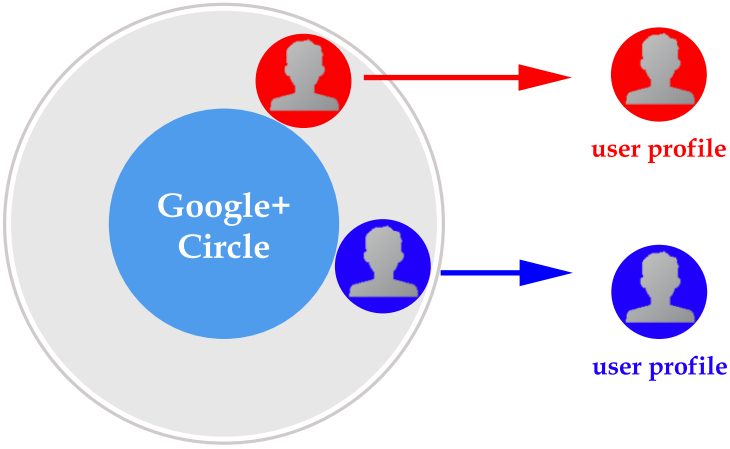

Figure 2.5: Scheme of circles.

+1 button

Many websites have already integrated into their pages the Google +1 button and it is used to signal to the world the user's appreciation for that page. Besides increasing the counter associated to the button to increase the index of overall enjoyment of the page, you can share a link to your Google+ profile, possibly accompanied by a brief comment. The last great advantage of this button is constituted by the influence that the clicks made on it have on the position of the page within the search engine.

Figure 2.6: +1 button of Google.

Moving the functionality within a corporate social network, where users are customers who have purchased a particular product or service, this turns out to

be a very powerful tool because it allows you to have a first feedback on the level of satisfaction given by customers to that particular product or service.

2.1.3 Twitter

Twitter is a free social networking and microblogging service, created in 2006, which provides users a personal page updated via text messages, called tweets, with a maximum length of 140 characters. The main feature of Twitter is the speed, that is the real-time nature of the information.

Twitter than other social networks, it is much easier at functionality level, but this does not mean that a company can not use Twitter for your business. In fact, it is one of the tools that companies can use to bridge the gap between the company and its customers or fans and if used intelligently can offer great benefits to the brand and its products. The best known example of successful use is that of Dell, which offers its products on offer (out of production and inventory) through the social network. The company announced that it has earned, using Twitter without any particular strategy, a figure equal to about $ 7 million. This is not a particularly significant figure for Dell but it proves that the Twitter audience is active on this platform.

Figure 2.7: Twitter's logo.

In the following we will analyze the characteristics functionality of Twitter considered to be particularly interesting, namely the hashtag and the promoted Tweets.

Hashtag

In Twitter the use of the hashtag facilitates categorization, search and aggregation of messages and discussions. The character "#"before specific keywords allows the transformation of these words in active links that create digital communities gathered around a topic. In this way it is possible to collect all the recent posts mentioning the same hashtag in a single page called hashtag page. In Figure 2.8 is shown an hashtag page generated from the hashtag "#GoodThingsInTheWorld".

For a company is important to create specific hashtag for the brand name, for the product and/or for an event, to which must be added the short description that allows to obtain uniqueness and originality. Once you create an hashtag, Twitter analyzes it: if already present inserts it inside the specific hashtag page,

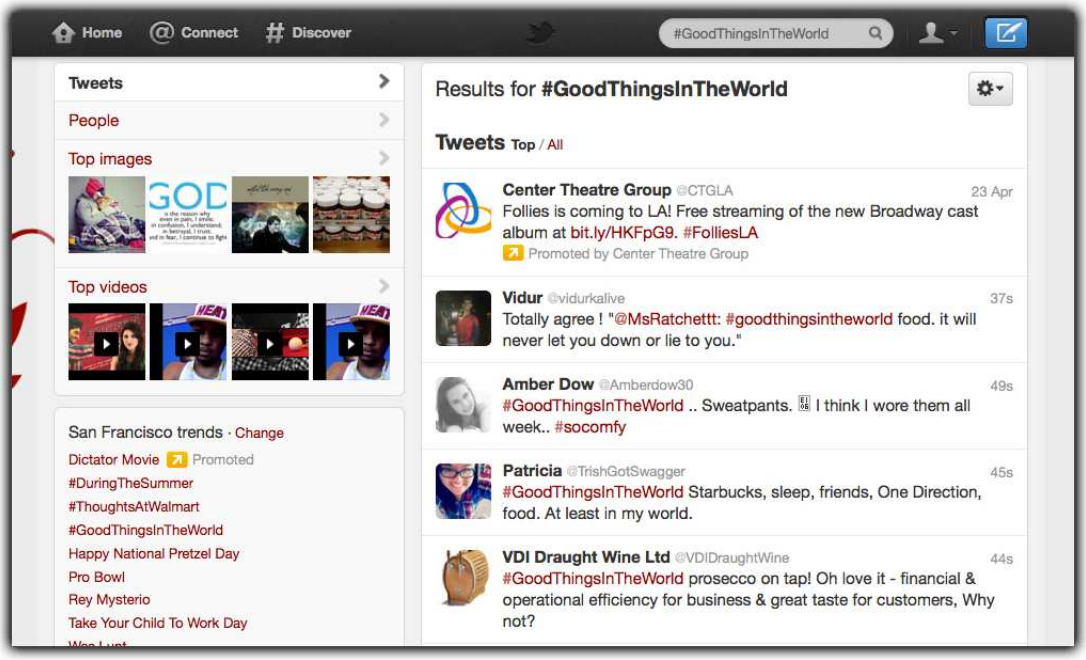

Figure 2.8: Example of hashtag page.

but if it is a new hashtag, it is recognized as original and Twitter goes to create a new hashtag page where they will be placed and maintained all tweets that contain the keyword.

The use of this tool can be designed to achieve two objectives: one is to be understood as a mechanism for dialogue with the user, the other as a tool to encourage social collaboration to deliver value. From the fusion of the two objectives is possible to implement an effective strategy that could optimize and integrate the internal process of ticket management, responding to the first client requesting information and, at the same time, communicate to all others who have the same problem. In this way, also users may provide information or propose their own solutions, thus to integrate the work of customer service.

Promoted Tweets

One of the main problems for those who promote a product or service through social networks is the speed with which the post on the boards of Facebook, and even more in the Tweet, run away. This forces us to frequently update the content so as not to fall into invisibility within a few seconds and may therefore be considered spammers.

Twitter has found the solution to this problem by creating a new advertising formula called ¨Promoted Tweets˝. The Promoted Tweets can appear as a result

of a search, in the timeline of the promoter or in that of users and above all remain visible for days without being hidden by the messages of other users.

Another interesting feature is the ability to identify the recipients of its Promoted Tweets by geographical criteria (Italy, France, Japan, etc..) or by sector (auto, finance, politics, travel, etc..). An example of a Promoted Tweet is shown in Figure 2.9.

Figure 2.9: Example of Promoted Tweet.

Not being banner but tweet in all respects, to the Promoted Tweets can be answered, they can be added as a favorite, and above all you can retweet to other, allowing the user to become a instrument for the spread of the advertisement.

2.2 IT tools for customer satisfaction mining

"A satisfied customer is the best business strategy of all."

For a company it's important to constantly capture the perceptions that people have of the brand and its products since this translates into increased profits and reduced costs. But how to collect this information? Nowadays there are various technologies that combine together help businesses to collect the moods of their customers. In this thesis work we will try to identify a set of tools to accomplish this task, and among all these will be implemented one.

The final goal is to provide an estimation of the level of customer satisfaction about the products sold by the company. To do this it is necessary proceed in two phases: the first is to collect data on how the user interacts with the instruments at its disposal, and the second must process the data previously collected in order to extract the estimated value of satisfaction.

The traditional techniques that allow the realization of these steps, such as filling in questionnaires, are not enough to collect meaningful data because they are considered too invasive and the user usually tends to ignore them or give unreliable answers.

The need therefore arises to design tools that are able to extract the expressed sensations by the analysis of the actions performed by the user. These instruments must play their role while providing an additional service to the customer, thus facilitating the interaction between customer and company. The main source of information is exactly this interaction between customer and company, because the manner in which it takes place and the tools used are points in which can happen the gathering of interest data.

The identification of the most suitable tools, requires an initial in-depth analysis of the interaction that correlates the customer to the company. The client interacts through representations of products (purchased or potentially purchasable) provided by the company through various channels. It is therefore important to identify the modes of interaction that are traceable. Tools that enable the collection of perceptions about customers need to create new communication channels parallel to the existing ones.

The channel more easily traceable is the company website, which plays the role of interface between the two entities. Through it, the customer can see the online manuals, the page of Frequently Asked Questions (FAQ) and browse the list of products. With these simple actions, using a special component for tracking the paths followed, it is possible to extract a first set of information. This information can give us an estimate on the products that most affecting the users and on the emotional state of the customer. In fact, if a user navigates repeatedly between the same pages of the FAQ you can guess that the customer has a problem with that product. This type of interaction takes place in a static

Figure 2.10: Interactions between customer and company.

manner and provide indirect perceptions, ie of which has an estimated value and therefore uncertain.

A more dynamic way to obtain information consists in using techniques of social networking, which are a collection of tools that allow a more direct interaction between customer and company. Some of these tools appear among the characteristic features of social networks analyzed above, such as the "I Like" button, the system for managing comments, chat, post, etc. The data collected with the use of these technologies are more explicit and allow to give more truthful satisfaction estimates, since they express feelings directly. It is clear the need to create alternative communication channels that can host these tools.

The immediate scenario that emerges is the possibility to realize a system capable of providing a score obtained by extracted perceptions, weighed according to the used tool: the scores obtained from instruments that provide direct perceptions will have a greater weight than those that provide indirect perceptions.

The next step is to identify a set of tools that are able to collect the information of interest in an alternative way than traditional methods using as a reference model the characteristics identified in the analysis phase of the most popular social networks. Each tool is described through the use of at least two summary tables:

- the first table is the table of access, which is used to specify the categories of users to whom the tool is addressed;
- the other tables specify the actions that the various types of users can perform.

2.2.1 Chat

One of the first tools identified to achieve the objectives set is the chat. Chat is an additional service, available only to registered customers, halfway between call and email. Customers and employees have different needs so there are two different versions of the chat: one customer side and another employee side.

Customers who need to communicate with the company can use this tool. In order to avoid a high workload for the company, it was decided to offer the service only to the category of customers registered at the company (for example, only those who have already purchased a product) and for a limited number of times. Through this instrument customers can open chat after choosing the reason for which they decided to use it and the product of which they wish to discuss. Subsequently, the chat is opened and the conversation begins. With the aim to increase effectiveness and speed of communication, have been introduced a number of features, for example the possibility to upload images like screenshots of error messages.

In Figure 2.11 is shown the storyboard of the customer chat. The basic idea is to provide an intuitive and easy to use tool to promote active participation. This implies a maximum reduction of the number of screens and buttons in order to avoid unnecessary steps. As shown in the figure, the user need only select the motivation and the product he wants to discuss, and in the next screen he can immediately communicate with the employee. With a few clicks the chat is started.

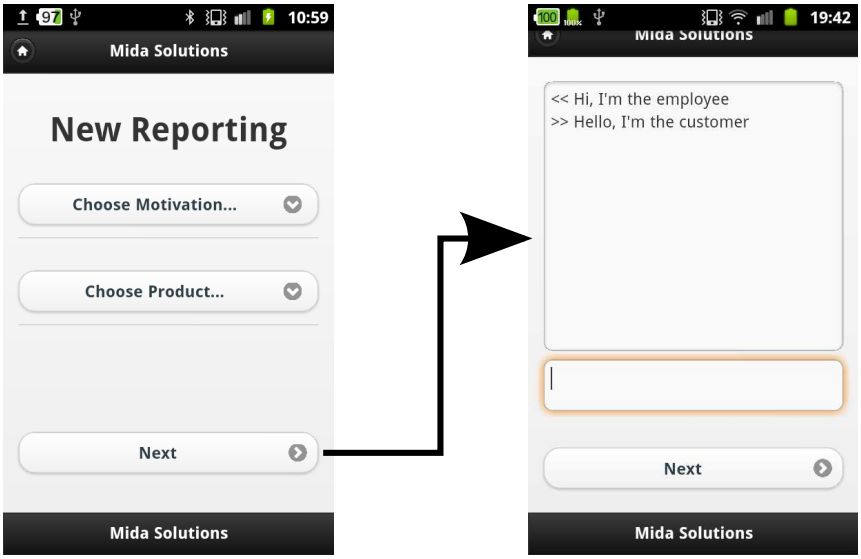

Figure 2.11: Storyboard of the customer chat.

Even the employees of the company must have a chat through which respond to various customers. They need for additional features to keep track of chat and to be able to assign an estimated value at the perceived level of customer satisfaction.

In Figure 2.12 is shown the storyboard of the employee chat. This chat is slightly more complex than the counterparty targeted at customers because it must manage multiple sessions and allow classification of each of them. The first screen shows the list of customers registered to the service, divided into online users and offline users. When a user starts a new chat session, his application sends a notification that the employee side application is capable of handling. The employee click on the button with the name of the user and opens the second screen in which begins the exchange of messages. When the chat ends, the employee press the button to resolve it. This action opens the evaluation screen where is shown the discussion just ended and the possible votes to be assigned. In the design phase, has also been included the ability to assign more votes within the same chat session, because while chatting you can tackle more than one issue. Finally, the last screen shows the ability to consult the chat already closed in order to save time, proposing solutions of problems already solved.

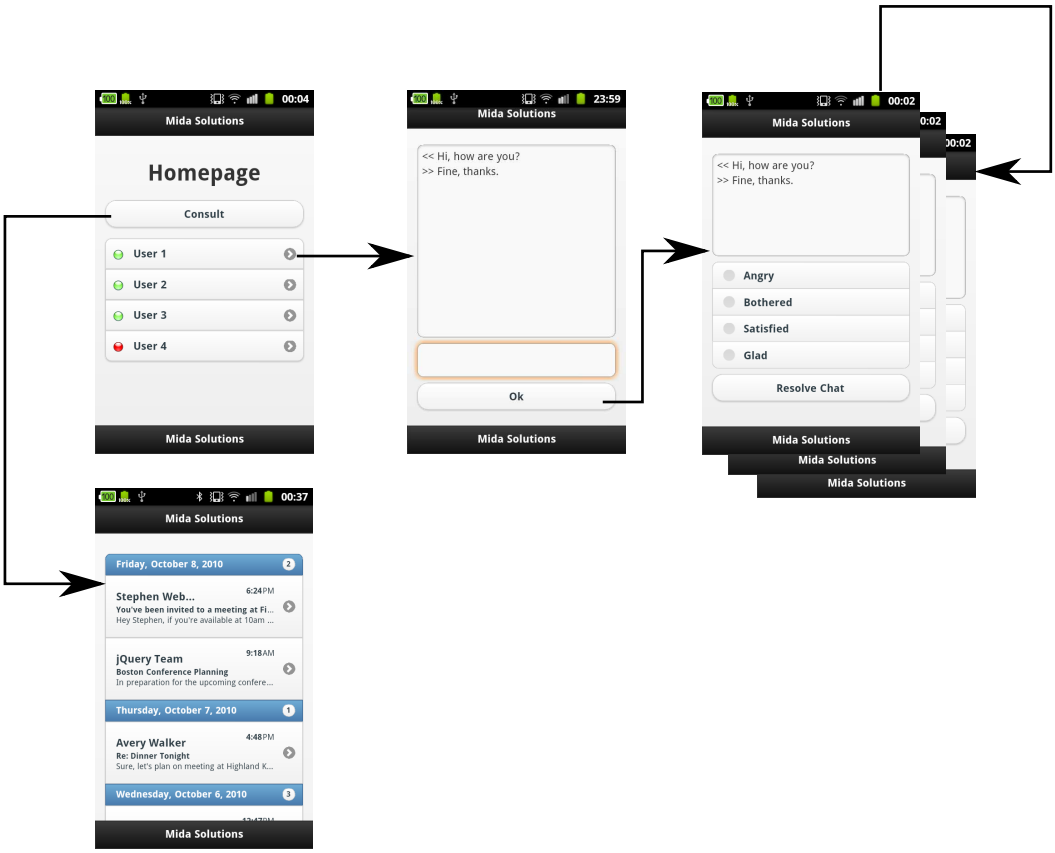

Figure 2.12: Storyboard of the employee chat.

Table 2.1: Access table to chat.

User Type	Access
Not Registered	NO
Registered	YES
Employee	YES

Table 2.2: Actions of the chat available to customers.

Action	Description
Motivate	Gives a reason for opening the chat
Find	Opens a new chat session
Show	Loads and send an image
Resolve	Closes the chat session
Express	Writes in the chat
Vote	Provides a vote on his level of satisfaction

Table 2.3: Actions of the chat available to employees.

Action	Description
Open	Opens a new chat session with the customer
Show	Loads and send meaningful data (e.g. diagrams, images)
Resolve	Closes the chat session
Reply	Writes the answer in the chat
Consult	Views previous chats
Modify	Changes previous chat
Vote	Provides a vote on level of customer satisfaction

2.2.2 Ticket Tracker

One of the tools achievable in a business context is the ticket tracker. The basic idea is to encourage and facilitate the interaction between customer and company taking as a benchmark the immediacy of Twitter.

The ticket tracker thus allows you to send reports of various kinds, in reference to a specific product, in a few easy steps. This information is stored in the form of Mantis bugtracker ticket (also called issue). Through this mechanism, the client is actively involved and contributes to the improvement of the products developed and the services provided. Such as chat, even the ticket tracker is provided only to registered users. The application is designed for mobile devices, allowing users to make their reports at any time from their mobile phones.

The company counterparty of the application allows to manage this informations and assign them assessments, which will be analyzed later by a specific tool. The application used by the employee to classify the reports should provide additional features to perform its task. The employee must be able to:

- view the new messages received (divided by category);
- select them one at a time;
- reply (if necessary);
- give a vote on the level of customer satisfaction perceived.

Table 2.4: Access table to ticket tracker.

User Type	Access
Not Registered	NO
Registered	YES
Employee	YES

Table 2.5: Actions of ticket tracker available to customers.

Action	Description
Motivate	Gives a reason for opening the new report
Express	Writes the report
Consult	Views previous reports

Table 2.6: Actions of the ticket tracker available to employees.

Action	Description
Reply	Writes the reporting reply
Consult	Views previous reportings
Vote	Provides a vote on level of customer satisfaction

2.2.3 Web Area

A crucial tool for the tracking of user actions is certainly the company web area. Through it, the user can know the products offered, solve basic problems and stay informed about the news. The web area is consulted by all types of customers, ie those actual and potential and thus brings inherently a very large information content.

In addition to the kinds of favorite products is possible to reconstruct the navigation paths made by individual users, and from these try to make assumptions about their emotional state: a fast navigation may involve frustration, because the user can not find what he is looking for; a slow browsing, in contrast, may involve satisfaction because the user has found what he was looking for. The

pool of information obtained from this instrument delivers precise, but not easy to analyze, indications. In fact, with only this information, you can get the simplified assumptions, that are hard to verify. To facilitate the phase of data processing, it is useful to exploit the structure of the web area, ie by dividing the Company's website in three main parts: root node, intermediate nodes and end nodes. With this further subdivision, statements can be made more precise: if the user remains stationary on a specific terminal node and then leaves the site, then most likely will he has found what he sought, on the contrary if he remains stationary on an intermediate node for a long time and then leaves the site, this may mean that he did not find what he was looking for.

The real power of this instrument however is in its completeness, in fact it is able to provide also those components for direct evaluations already mentioned above. The user as well as browse through the website, can also make judgments through the addition of comments to the post, clicks on the "+1" at a news, he shares page associated to a product in his Facebook account, etc. Due to its completeness, you can cover the majority of feelings expressed by the user, thus not just limited to establish only interest or indifference toward specific products.

Table 2.7: Access table to web area.

User Type	Access
Not Registered	YES
Registered	YES
Employee	NO

Table 2.8: Actions of web area available to customers.

Action	Description
Search	Browse in web site
Leave	Leaves the web site
Vote	Interacts with voting mechanisms
Motivate	Gives the motivation for the new reporting
Express	Writes the reporting
Consult	Reads web page contents, downloads documentation

2.2.4 Direct Input

The direct insertion of signalings is used to carry out directly any type of report. The instrument has been designed as integration of the website. The basic idea is to provide a tool that enables in a few simple steps the insertion of votes accompanied by motivation, from any terminal node you are: if the user is on the description page of the product that owns and of which wants to propose

the addition of a feature, he will have to press the specific button to open a form where he can:

- select the reason for reporting;
- write the contents of reporting;
- submit the reporting;
- vote the selected product.

In this way the details of reporting, such as the name of the product, will be automatically filled by the application, based on the terminal node from which is carried out the insertion. For messages of a generic nature, the user must simply click the button associated to the feature on the homepage of the company website.

The message can be forwarded to the company, which will use a special software or simply email to receive notification. Signalings by direct input can be imagined as comments already classified based on the cause that generated them.

Figure 2.13 shows the storyboard for direct reports. Once the user arrives at the desidered product page, he clicks the button to the direct reportings and opens a popup window with which he can interact intuitively.

Table 2.9: Access table for direct input.

User Type	Access
Not Registered	NO
Registered	YES
Employee	YES

Table 2.10: Actions of direct input available to customers.

Action	Description
Vote	Gives a vote on product
Motivate	Gives the motivation for the new reporting
Report	Writes the reporting
Consult	Views previous reportings

2.2.5 Twitter

An interesting opportunity comes from the integration of Twitter with the company website. In this way we can provide the opportunity to make comments

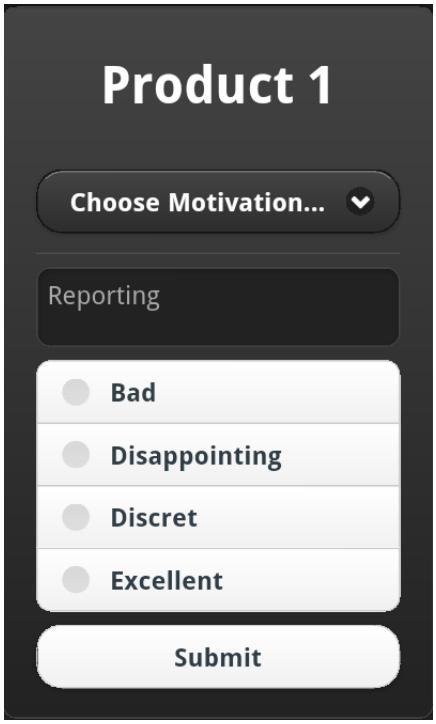

Figure 2.13: Storyboard for direct reportings tool.

related to products in the products website page and then, after a possible filtering operation, publish them in the Twitter company account automatically using specials hashtag.

The hashtag mechanism, in fact, provides a very basic, but at the same time effective, classification because all the comments are in chronological order and then you can rebuild entire conversations. In the transition between the company website and the publication on Twitter we can evaluate the comments on the fly or save them for later analysis. In this case it is necessary to create a suitable software tool for the employee, which is able to take the comments from the site, view them, classify them and eventually post them on Twitter.

So the integration of the two channels, that related to the company website and that on Twitter, through an application bridge enables the analysis of the comments. The main strength of the instrument lies in the simplicity with which the user can interact with it. Conceptually, the mechanism for inserting comments so structured, it is halfway between a forum and a chat, as it allows users to interact with each other, but at the same time the company has control over what is published. As stated for other instruments identified, only registered users have the ability to participate actively in discussions. However, everyone

can see the comments posted.

The implementation of the tool on the customer side is really quite simple, since it consists in the simple addition of a special text area next to the product description.

Turns out to be much more complex the structure of the company software for managing comments. Hereinafter will discuss the storyboard shown in Figure 2.14 When the application starts, a screen appears that summarizes all the new comments to be analyzed, grouped by product. The user selects the product for which wants to manage comments and the application opens the detail page associated with that product. This page lists four main information:

1. the product name;
2. the product description;
3. the list of approved comments with other information such as author and date of submission;
4. the list of comments to be approved with other information such as author and date of submission.

For the management of the new comments, the application provides three buttons:

- **Delete**. Pressing this button will open a popup window in which you are prompted for confirmation before deleting the message.
- **Reply**. The button allows to reply privately to the comment. You will see the appropiate page, which contains the comment and the reporter name. The answer can be sent through the corresponding button.
- **Approve**. When the button is pressed, the application loads a page where you can assign a category and a vote to the comment. Pressing another button the comment is posted to company Twitter account.

Table 2.11: Access table for Twitter integration tool.

User Type	Access
Not Registered	NO
Registered	YES
Employee	YES

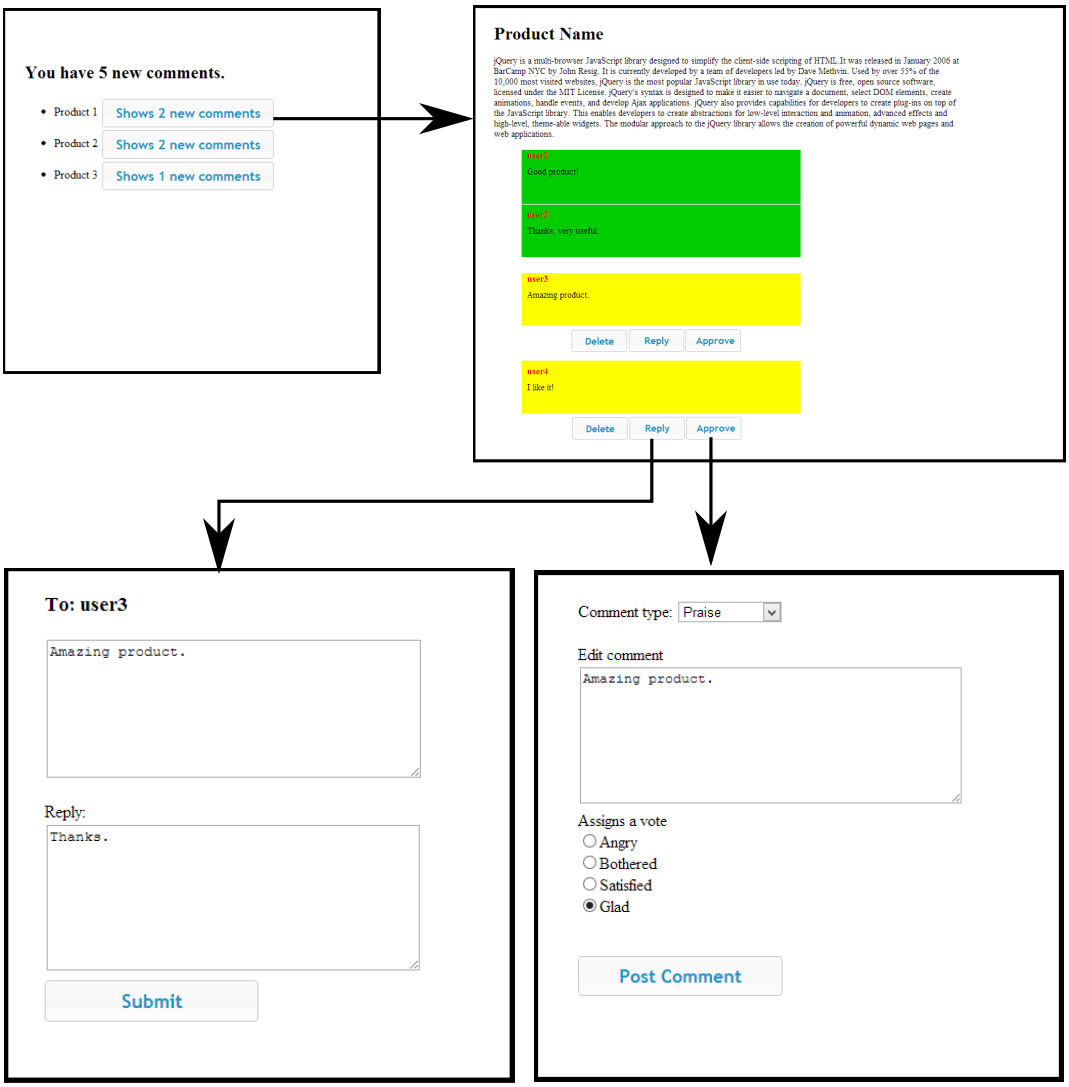

Figure 2.14: Storyboard of the bridge application.

Table 2.12: Actions of Twitter integration tool available to customers.

Action	Description
Report	Writes the report
Consult	Views previous reports

2.2.6 Forum

Interesting interaction could be achieved putting in relation different parts, not limited to only two entities. An example of this interaction is the forum. The

Table 2.13: Actions of Twitter integration tool available to employees.

Action	Description
Reply	Writes the reporting reply
Consult	Views previous reportings
Approve	Approves comment
Post	Post comments on Twitter account
Classify	Assigns category to comments
Vote	Provides a vote on level of customer satisfaction

forum allows users to send public and private messages with the help of one or more moderators. In this way a solution can be shared among several people, responding with a single post to all those who are in the same situation.

This approach brings together multiple entities going to create a community of people who share interests. In our case, we can get groups of people united by the products purchased. Only registered customers are active participants in a discussion, everyone else can see the threads. With this logic of collaboration, users can find solutions independently and thus indirectly help the company in its activity of support customers.

On the other hand this approach is risky because small problems can be expanded, spreading discontent in many customers and so the company image can be damaged. A solution to this problem is to filter the messages left by users: before being published the message must be approved by a moderator. This involves a cost to the company because an employee has to use some of his time in reading the individual messages.

Table 2.14: Access table for Twitter integration tool.

User Type	Access
Not Registered	NO
Registered	YES
Employee	YES

Table 2.15: Actions of forum tool available to customers.

Action	Description
Report	Writes the comment
Consult	Views previous discussions
Motivate	Gives the motivation for the new discussion
Open	Open a new discussion

Table 2.16: Actions of forum tool available to employees.

Action	Description
Reply	Joins in the discussion
Consult	Views previous discussions
Approve	Approves discussion/comment
Post	Adds approved comments to the discussion
Modify	Edits comment
Classify	Assigns category to comments
Vote	Provides a vote on level of customer satisfaction

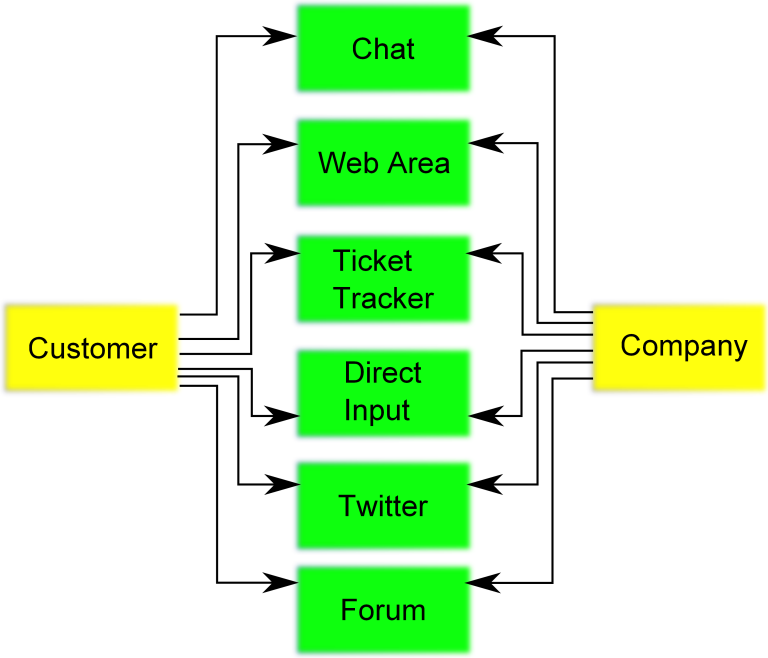

Figure 2.15: Channels found.

2.2.7 System design

Before starting the design phase it is necessary to make a careful analysis to identify the different actors interacting in the system and their role in it.

Intuitively, one can say that the entire application works around the customer. In fact he is the subject from which we obtain information in some way. On an abstract level, it is possible to imagine the user as the source of input data to be analyzed.

The first step of this analysis aims to identify protagonists around which it is possible to develop the entire application. In the studied company context,

the information to be obtained is the level of customer satisfaction, which can be manifested through emotional states that the client expresses directly and indirectly. For these reasons it was decided to create a specific category called `perception`. Examples of values for the category are: satisfaction, anger, appreciation, interest, and so on.

Perceptions can be obtained directly, through evaluations that users provide on a specific product, or indirectly, through the analysis of how they use the tools available to them. Customer feedback can not always be achieved through the use of tools such as mandatory surveys, because this procedure is considered too invasive by the user who generally tends to avoid it. It is therefore important to look for alternative methods to extrapolate judgments about products and services. One of the proposed solutions is to analyze the actions that the user performs with the tools provided. Thinking users who browse in the company website, you can trace the paths that they carry out and make assumptions about their emotional state. From this reasoning it is clear that users interact with the tools through actions. Each tool has its own set of actions, however, most of them are common to several instruments. Hence the definition of action: doing the same thing with different tools. For example, "vote" in the chat means to assign a numerical value to user emotional state, while the same action on the company website means press the "I like" button in correspondence of a product or functionality.

Finally, the last big category is represented by the context of navigation, which provides us more information about the reasons that led customers to use a tool. A user who leaves a comment after navigating repeatedly between the same pages of a specific product tells us that the user will most probably be angry.

Figure 2.16 shows schematically the categories described above.

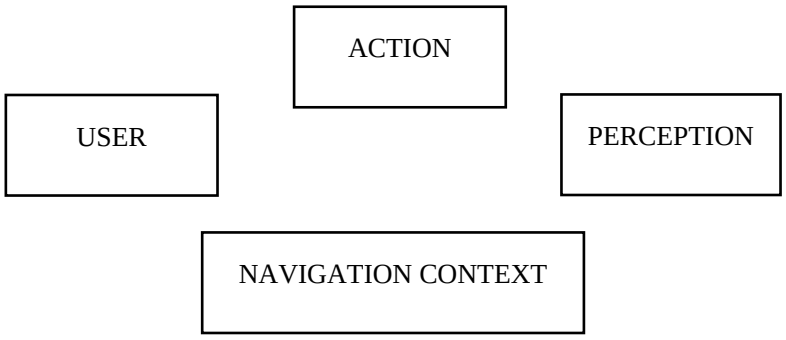

Figure 2.16: Main categories of the system.

The second step is to define the relationships between the various entities through a detailed analysis.

Recalling that the purpose of the system is the calculation of the level of satisfaction, we can say that perceptions are closely linked to actions, since the performed action reflects the mood of the person who executes it. In this sense, the actions are the core of the process of gathering information, in fact, through them, you can extract the assumptions on which assert considerations.

Actions, however, vary in number and typology depending on the tool used. So there is also a relationship between the action and the tool that enabled us to generate it, which allows to know the source of action, and therefore provides important additional information. In general, most of the tools allow the user to provide explicit information about his level of satisfaction (for example, through the "I like" button, popup for voting, etc.). There is, however, another type of information in addition to the direct one, that is, the intrinsic information. From the analysis of the context in which a tool is used, you can perform different considerations: there is a difference between a customer that opens directly a chat and a customer that opens it after consulting several times the web documentation. From this observation it is possible to identify the need for a further relationship between the instrument and its context of use.

As stated above, different tools have different actions, but it is also true that not all users have access to the same tools. The subdivision of the users leads to the definition of the two main categories of users already mentioned (registered and unregistered). The user plays the role of the main actor because he uses the tool, which allows to perform actions from which we extract the desired information. On the other hand the customer can use a tool even after examining a product of interest. In fact, most of the features offered is based on the interaction between the customer and the product, which indirectly generates actions. With this specification we can create two parallel paths where the user can interact with the tool in two ways: directly or through the representation of the product.

The final scheme is described in Figure 2.17, Table 2.17 and Table 2.19.

The entity USER represents the set of all possible users of the tools previously discussed. Registered customers, potential customers and ultimately the employees belong to it. The purpose of this entity is the identification of the user that interacts with the system. It is important to know what actions individual customers perform because, through appropriate software, you can get their profiles. This is the basis of targeted advertising. If the user is not a registered user, will only have access to a limited number of tools. Among all these tools, the most important is the company's website. In the category of non-registered users, there are also potential customers, then you should pay particular attention to them. The data related to this type of users can be analyzed separately by other tools to increase the effectiveness of the company's website.

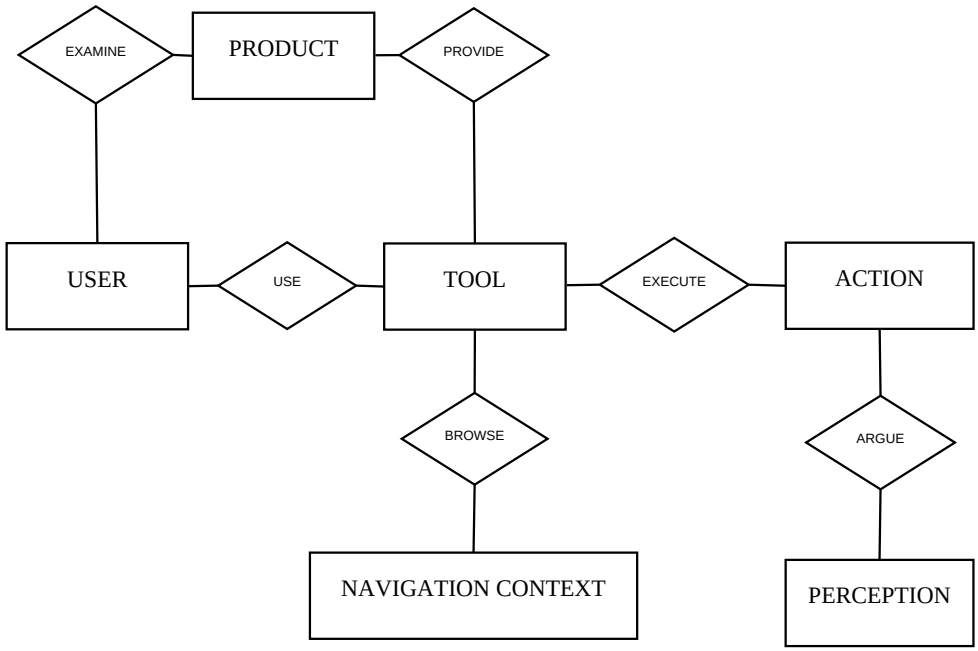

Figure 2.17: Final scheme of the system.

The user can interact with the system through the direct or indirect use of the instruments. Indirect use means perform actions without passing through products representations. For example, the start of the chat can be done by selecting a purchased product or by starting a conversation to make general proposals. The user can also use a product, which in turn allows the use of one of the discussed tool. This relationship between user and product is called `examine`.

The entity PRODUCT allows you to collect all the information related to each product. It plays a key role because the assessments for the products allow for the identification of weaknesses and then the changing of business processes that manage them. In most cases, the evaluation is almost always attributable to a perception arising from use of a product.

If the representation of the product provides a tool then it is possible to automatically reconstruct the association between tool and user that has used it, through the utilization of the relationship `provide`. Instead the direct link between USER and TOOL is made by the relationship `use`.

The entity TOOL is the center of the diagram because it is the missing piece to the connection between the user and perceptions. The tools create independent communication channels through which you can collect the desired

information. The information content is very wide, since the applications are theoretically able to track each user's click. The size of the information content depends on how much the user uses the tool, so we have to provide added value to user's life through the utilization of the instrument.

The information you intend to get is not only the explicit one, provided by the tools themselves, but also the most intrinsic, not immediate, but it turns out to be very important. In general, you can obtain information, analyzing the routes taken before the use of a tool and from where did it come. The tracking of activities carried out by the user before starting a specific tool is made possible by the NAVIGATION CONTEXT entity. Through it is then possible to deduce additional information. The relationship that positions the tool within the navigation context is called `place`.

Among the most interesting relationships appears the relationship `execute` which connects TOOL with ACTION. The latter represents the sets of actions listed in the tables in previous sections. Each tool has its own actions, each of which allows the reconstruction of the user navigation within individual sessions. Effects of actions allow to deduce perceptions. For this reason we have created a relationship called `argue`.

At the end of the process, from the values stored in the entity PERCEPTION is possible to extract an estimate of the level of customer satisfaction. The application, that deals the analysis of the collected data, associates the individuated perceptions to numerical values and from these it is possible to perform various operations.

Table 2.17: Main entities of the system.

Entity	Description
USER	Have been identified three main categories of users that can interact with the system: • **Registered customer**. Customer of which you know the data and you want to keep track of his actions. Generally belong to this category all those customers who have already purchased a product. This type of customer, in addition to access to non-registered customers tools, has at its disposal a set of unique tools, which are used as an additional input source for the assessment of the level of satisfaction. • **Non-registered customer**. Users of which you do not know specific information about the current access. Generally, potential customers belong to this category. This type of customer has access to a limited set of features. Information is extracted through tracking of customer actions. The score thus obtained must have less weight compared to that obtained in an explicit way. • **Employee**. The company's employee has access to specially modified versions of customers tools that allow him to express perceptions directly.
ACTION	In this category there are operations that allow you to run the same interaction with different tools.
PERCEPTION	It represents the emotional state of the user. It can be obtained directly (by filling in the appropriate fields by user or by an employee who interacts with him) or estimated (analyzing the tracking of user actions).
NAV. CONTEXT	An important source of additional information useful to strengthen the analysis of the tracking is given by the navigation context understood as attitudes, paths and actions taken by the user before arriving at a specific page or before launching a tool.
TOOL	Set of tools used by the user to interact with the company.
PRODUCT	This entity represents the products with which the user can interact.

Table 2.19: Relationships between entities.

Relationship	Description
argue	The actions are the core of the system of information gathering as through them you can build hypotheses on client's perceptions about a specific product/service. So is required a relationship between the entities ACTION and PERCEPTION.
execute	The actions, however, are in their turn generated by tools.
browse	The searched information is not only the explicit provided by the tools themselves, but also the most intrinsic, not immediate. In general, you can obtain important information, analyzing the paths taken before the use of a tool and from where did it come to use the tool.
use	Actions are made possible through the use of special tools. These tools are provided to the user based on the category they belong. This connec tion between the user and the tool is described in the relationship `use`.
examine	Most of the features are associated with products: the comments are left on products, the chat is opened as a result of problems/suggestions/questions regarding products, etc. It is therefore important to make explicit this relationship.
provide	The last important relationship is the one that allows you to create the association between product and tool.

2.3 A messaging tool for technical or commercial support

2.3.1 Objectives

"Any tool is a weapon if you hold it right."

Following the analysis carried out previously, it was possible to identify a set of tools capable of perform our goals. This phase was followed by a feasibility study with the purpose of choosing only one of them. By analyzing one by one these instruments we can extract the drawbacks and weaknesses possessed and reach a compromise between available resources and expected results. The following analysis summarizes briefly the reasonings followed for the selection of architecture to develop more suitable to our needs.

The chat is a very interesting tool and able to provide a large information content. Furthermore, it is not difficult to implement and fits perfectly to the context. From the practical point of view, however, it is difficult to use because it requires the constant presence of at least one employee that responds to all chat messages from customers. This involves a excessive workload, difficult to sustain by a company.

The web area is one of the most effective tools, on the one hand because the company website is an obligatory crossing point for all users and on the other because it is able to collect both types of perception. The biggest disadvantage is the difficulty of integrating new functionality on a complex and constantly changing structure.

The mechanism for the direct input of reports requires small additions on the company website and the creation of a mechanism for the management of reporting. The main drawback is the complexity of changing the company website. For the same reason also the integration with Twitter was discarded.

The forum, as well as having difficulty integrating with the web site, has the drawback, already mentioned, to expose the company to the risk of uncontrolled spread of discontent that could damage the image of the company.

The instrument chosen for the implementation is therefore the ticket tracker. In addition to the advantages already mentioned, it enjoys integration, through appropriate arrangements, with Mantis. In this sense, the various reports can be managed as ticket, thereby using an already existing structure as a supporting structure. Through this simplification, there are no costs arising from the adoption of new technologies and their maintenance.

2.3.2 Components

In 2012 smartphones sold in the world have exceeded one billion units: one person out seven has one, confirming the success of what has been called the fastest-spreading technology in history. In one year the number of smartphones in use rose from 708 million in the third quarter of 2011 to 1,038 billion at the end of September 2012. And that number is expected to double from now to 2015. Consequently has risen the number of applications downloaded from various Mobile stores. We can say that a great battle is taking place between two different ways to access the Internet: the "new" app-centered, and the "old" browser-centered. Which one will win? Impossible to say, but just think that, according to the latest estimates, by 2013 the number of users who connect to the Internet via smartphones will exceed those using a computer.

This information indicates that the market for mobile applications, and therefore the entire mobile world, is an attractive market to all and the presence in the field of Mobile Apps is crucial for a company. But what are the reasons for which the presence in the mobile store, such as Apple's App Store and Google Android Market, it is advantageous for a company? Listed below are the main ones.

- **Visibility**. The number of searches within the Apps Store is very high and therefore, appear among the search results, is good publicity for the company.
- **Increase the value of the brand**. Have its own application is a sign of a dynamic company, always attentive to the evolution of technology and abreast with the times.
- **Media coverage**. 75% of the planet is not yet connected to the Internet, but with the mobile world we can reach the users with poor internet connectivity, therefore catching every opportunity for company business.
- **Bring the brand directly in the customer's smartphone**. Smartphones are instruments that are always available to users, and therefore potentially always accessible.

The benefits deriving from the creation of mobile applications are varied, but an application must be used to deliver added value to customers, not only for advertising purposes otherwise you risk to reduce the value instead of increase it.

In summary, we can say that the rules to achieve success in the Apps Store are the following:

- **User Experience**. Applications must add value to the user's life;

- **Ease of use**. Applications should be intuitive and not too complex to be used;
- **Completeness**. The services must be fully functional and in continuous improvement;

As a result of these considerations, it was decided to implement the ticket tracker as an application for mobile devices, using the PhoneGap framework. The ticket tracker realizes direct communication between customer and company, making it an active player in the business life: he can praise the work, propose solutions and criticize the choices that are not shared. All in a few simple steps. In fact, the reference model is the speed of information that characterizes Twitter, which allows to create a sort of asynchronous chat.

The designed system is made up of three main components:

- customers application;
- employees application;
- application for the analysis of collected data.

Each of them has its own characteristics that can perform several tasks under its jurisdiction, but the common feature that applications must have is the ability to interface with Mantis. With this requirement, the user is able to make a report which is stored as a ticket in the Mantis database. The employee regularly monitors from its application the list of new reports and manages them individually. In the end, all the information stored in the database are processed by the employee through the third application.

Analyzing at the abstract level the functioning of the system, it is interesting to note that the components do not communicate directly with each other. The individual applications communicate with each other only through the use of the Mantis database, which thus assumes a central role in the architecture.

Here below is proposed a first overview of the individual components that serves to understand in detail the technological needs of each of them.

Customer App

The application used by the customer is structurally very simple. But what user needs for accomplish his tasks? Surely he must be able to view his reports and to place new ones. To encourage the use of the application, priority must be the speed of use. Once established these characteristics it is possible to propose a first application's storyboard that shows the three main screens. As shown in Figure 2.18 storyboard, user must first fill out a login screen with the credentials

assigned him by the company. In addition to traditional username and password, it is possible to also enter the address of the server that provides the services necessary for the application. If the login is successful, the application will display the main screen that shows two lists: one containing the reports in resolution and another containing the closed reports.

Figure 2.18: Login page in customer ticket tracker app.

The communications between customer and employee involve the exchange of multiple messages and only when both parties consider ended the conversation, the employee may end reporting. From creation to closing the report is in the list of reports in resolution. Since the closure, the reporting is in the list of closed reports. Therefore the purpose of this subdivision is to provide a conceptual ordering of data that allows user to develop a global view of the current state. In order to obtain also a visual separation of the lists were chosen different colors for each list.

Every element of each list contains the basic information that summarize the report. This information is:

- Title. It allows to summarize in a few words the content of the message;
- Description. It represents the body of the report;

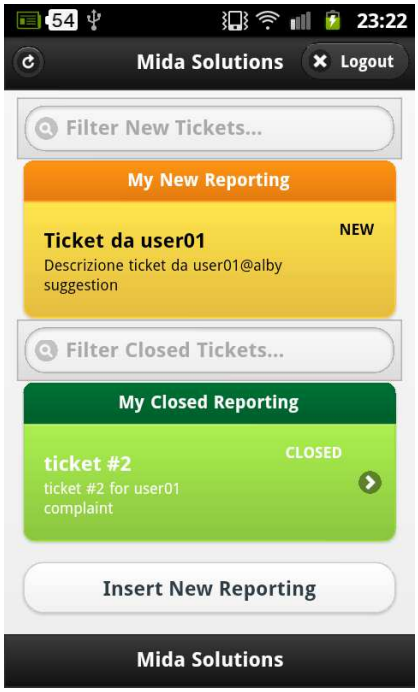

Figure 2.19: Homepage in customer ticket tracker app.

- Type of reporting. The purpose of this value is to provide a first logical division of the data;
- .Report Status. It is used to control how the signal evolves.

The messages are sorted according to the date of the last update in order to keep under control the reports that receive answers. It is possible also, in the implementation phase, add an additional view with report details. In this screen, it's possible add all the informations that describe in detail the ticket. Among all these, the application must display a special section that shows the entire conversation between reporter and handler. In order to exploit the potential of mobile devices, it has been added the ability to scroll through the list of ticket details using the swipe action.

The last screen is for the insertion of a new alert. Within it the user must first select the type of signaling between three possible values: praise, suggestion and complaint. The second value to specify is the product for which you want make the alert. This value and the previous one are chosen by the appropriate drop-down menu, in order to avoid input errors. The third and fourth field are respectively title and description. With these four values the message can be sent. Through this procedure, the user can constantly monitor the progress of the reports.

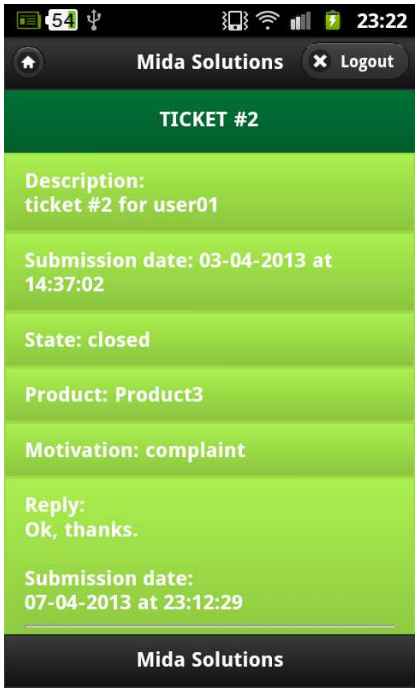

Figure 2.20: Page for report details in customer ticket tracker app.

To facilitate the search for reports in the lists, the application provides a filter to search for keywords within the information displayed. The filter searches the inserted keyword into its bar and displays real-time list items that contain it.

Employee App

The employee app is much more complex because it must be able to handle all the reports made by customers. Again the question arises: what the employee needs to accomplish its tasks? The employee must be able to:

- view new messages (in addition to those already closed);
- consult the list of reports, divided by category;
- display the detail page of the report;
- respond to the report;
- close the report;
- evaluate the report.

Figure 2.21: Storyboard for customer ticket tracker app.

The application will then provide a set of pages that can perform all these actions. At this point, it has therefore been proposed the storyboard in Figure 2.23, which shows the main screens.

After the usual login page, the application will automatically download all tickets stored in Mantis. Then the messages are grouped according to type, in three separate lists. A further subdivision of the tickets based on the value of the state: the main page shows the number of new ticket with the status for each category. From this page you can access pages of reports and resolved in resolution by pressing the corresponding buttons. These pages have a structure similar to that of the main page.

Selecting a type from the list you will see the screen shown in Figure XX containing the list of messages associated with that particular type. Even in this case, for each ticket are only reported basic information (title, description, status and type). The list elements are sorted according to the date of the last update, placing the most recent items at the top of the list. The employee can scroll through the list and decide which signaling resolve.

After clicking the desired item in the list, the page containing ticket details is opened. Among the displayed information appear the name of the product, the reporter, submission date, the name of the instrument by which it was made reporting and motivation. In the final part of the page there is a section containing the messages exchanged between the client and the employee, sorted from oldest

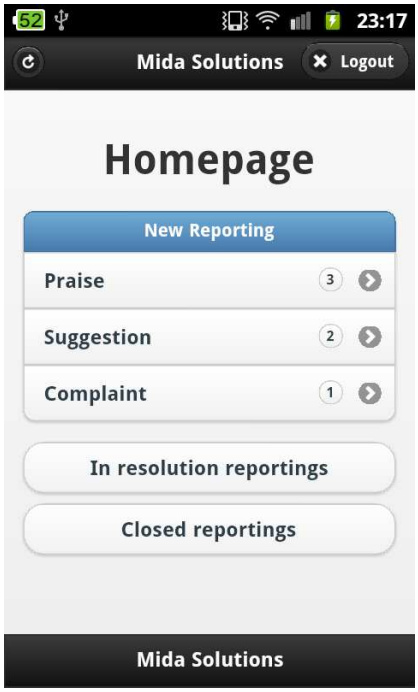

Figure 2.22: Homepage for employee ticket tracker app.

to newest, so you can automatically build the conversations that took place. In the case of reports that have not yet been answered, in this area is displayed a string that notifies the lack of answers. If the user wants to answer, he can insert his reply in the text box at the bottom of the page and then press the "Reply". If you want to directly close the alert, he presses the "Close" button. In the latter case, before updating the ticket in Mantis, the application opens a popup window which asks the employee to assess the state of the customer. After performing one of the two operations, the user is redirected to the main page that is re-created dynamically with the changes.

The process for the management of conversations in resolution is similar to that for new ones. Slightly different is the process of consultation of alerts resolved, as they can not be changed, but only consulted.

Analysis tool

After collecting the data through the application used by the customer, and after providing scores to the level of customer satisfaction for each reports, it is necessary to process the data in order to extract information of interest for company. On the collected data it's possible to perform various types of analysis, that can provide a wide variety of information contents, for example, the application is

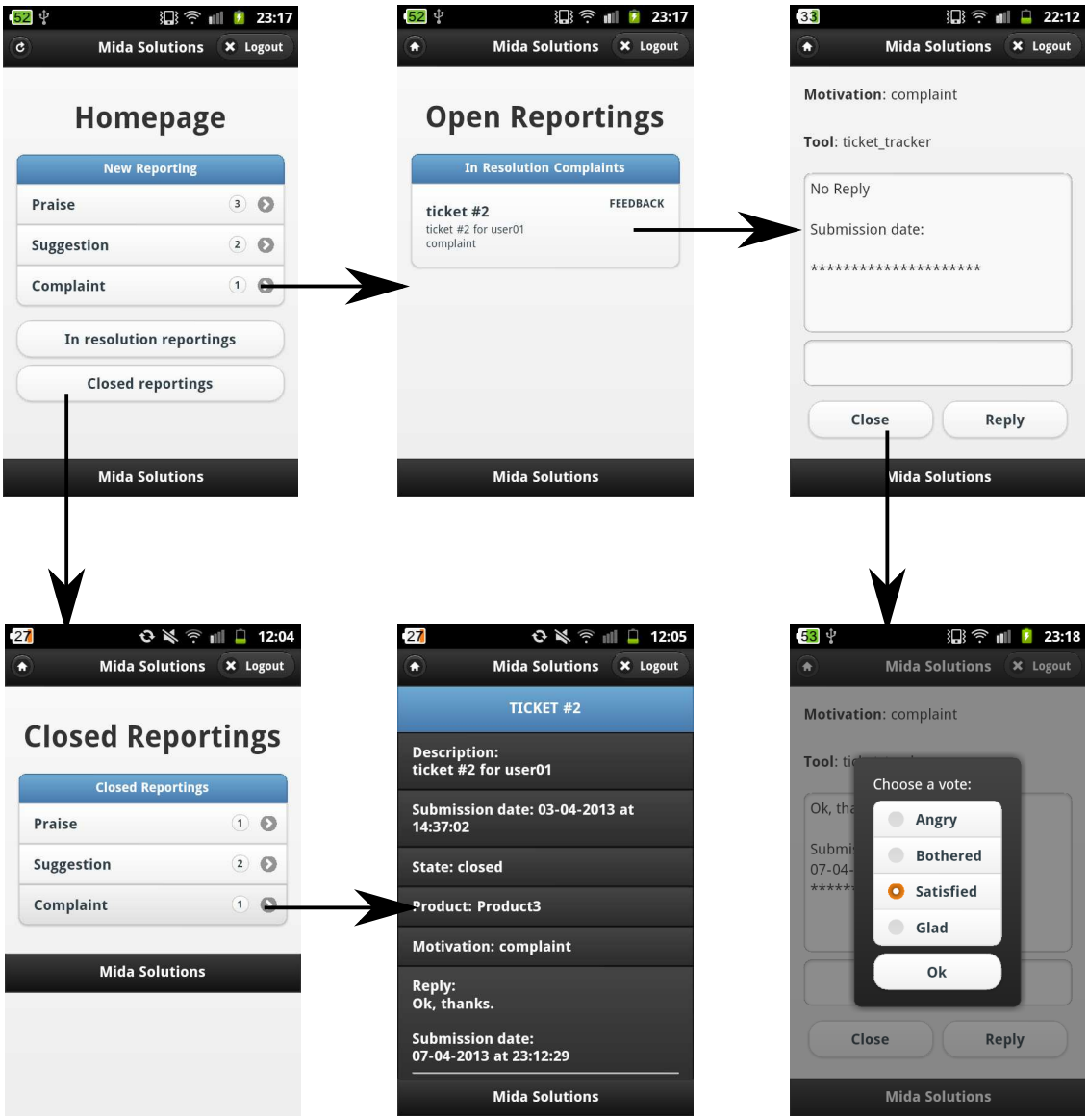

Figure 2.23: Storyboard for employee ticket tracker app.

able to extract the types of products preferred by each user, through the analysis of the reports. On the other hand, this tool does not have the ambition to perform complex analysis, but it is meant to show how it is possible to gather and process the data and then among all possible information it has been chosen only a small subset of them.

The information returned in output by the tool are:

- the total scores associated with each product;
- the best and the worst product;
- reports grouped by product and by users;
- reports made by individual users.

After fixing these objectives it is possible to construct a storyboard of the application. Unlike the other two applications, this has not been implemented for mobile devices, as it is not necessary to have it always available. It is used regularly by employees to monitor the performance of the products and other parameters. So the application has been designed as a web interface, consisting of a single page in which are gathered all the data of interest. The basic idea is to make available two tabs: one related to the products and the Users. Each tab consists of a summary table which lists the main data. In both cases, the table contains the number of reports made divided by type. It also needs a special area where to show all comments divided on the basis of the main entities of the belonging tab, for example in the products tab will be reported all of the comments assigned to each product divided into three sets, one for each type signaling (praise, suggestion and complaint). The proposed storyboard is shown in Figure 2.24.

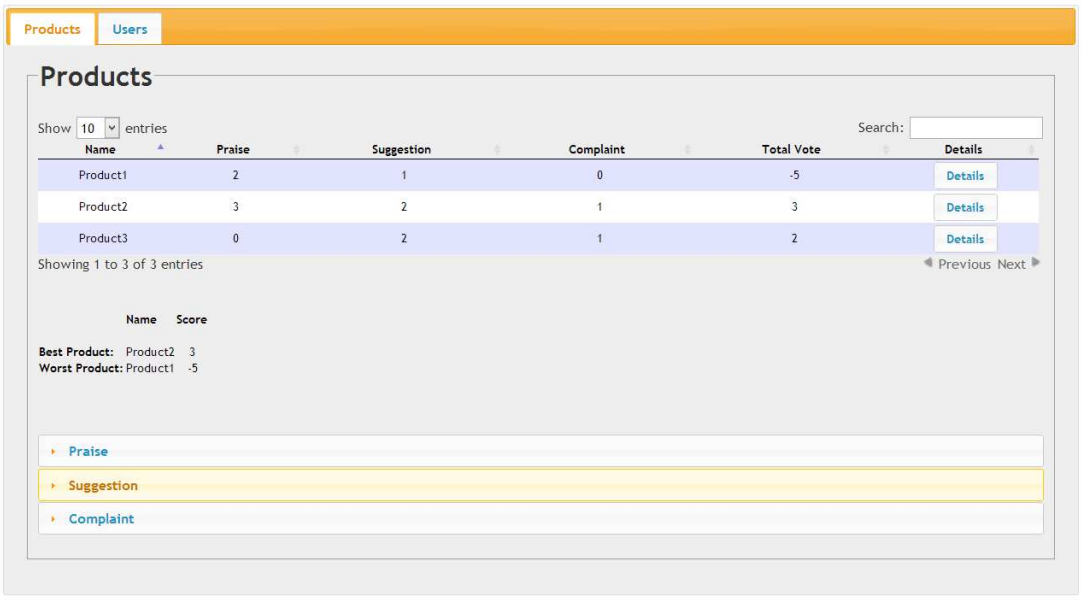

Figure 2.24: Storyboard for the analysis tool.

The instrument able to immediately understand the time trend of any phenomenon is the plot. Therefore it was decided to provide the application the ability to dynamically create charts. In this way, by selecting any product, the employee sees the time trend of the votes allocated to it, allowing him to highlight any critical issues and determine which company processes need adjustment. An example of chart for the representation of the temporal evolution of a product is shown in Figure 2.25.

Figure 2.25: Chart example.

Chapter 3

Part II - Technologies

After defining the objectives, the analysis moves to the development environment necessary to achieve them. The development environment consists of a set of suitable tools for our needs, which were chosen to make the application the most user friendly possible.

Before writing code, it is necessary to collect some libraries and some tools. These are:

- **jQuery**: The jQuery library makes dealing with HTML and CSS a breeze, and it is also extremely handy for manipulating XML, and therefore, XMPP stanzas.
- **jQuery UI**: The jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library.
- **jQuery Mobile**: A unified, HTML5-based user interface system for all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily themeable design.
- **Mantis**: MantisBT is a free popular web-based bugtracking system. It is written in the PHP scripting language and works with MySQL, MS SQL, and PostgreSQL databases and a webserver. It is used for storage and management of reports.
- **PhoneGap**: PhoneGap is a free and open source framework that allows you to create mobile apps using standardized web APIs for the platforms you care about.
- **XMPP**: Extensible Messaging and Presence Protocol is a communications protocol for message-oriented middleware based on XML (Extensible Markup Language). The protocol was developed by the Jabber open-source community in 1999 for near real-time, instant messaging, presence

information, and contact list maintenance. Designed to be extensible, the protocol has also been used for publish-subscribe systems, signalling for VoIP, video, file transfer, gaming, Internet of Things applications such as the smart grid, and Social networking services.

- **Smack**: Smack is an Open Source XMPP client library for instant messaging and presence. A pure Java library, it can be embedded into your applications to create anything from a full XMPP client to simple XMPP integrations such as sending notification messages and presence-enabling devices.

- **Servlet and JSP**: A servlet is a Java programming language class used to extend the capabilities of a server. JavaServer Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. The difference between servlets and JSP is that servlets typically embed HTML inside Java code, while JSPs embed Java code in HTML.

This chapter analyzes in detail these instruments.

3.1 jQuery and jQuery UI

The jQuery and jQuery UI libraries are available from `http://jquery.com` and `http://ui.jquery.com` respectively. jQuery, like many JavaScript libraries, is available in normal and minified version, that is a compressed version of the file obtained through the operation of minification. Minification is the practice of removing unnecessary characters from code to reduce its size thereby improving load times. When code is minified all comments are removed, as well as unneeded white space characters (space, newline, and tab). In the case of JavaScript, this improves response time performance because the size of the downloaded file is reduced.

Google has made available many libraries through its AJAX Library API, and among them also appear jQuery and jQuery UI. So, if the application is designed to run on devices constantly connected to internet you do not need to download and store the libraries, but just connect to one of the Google's super fast servers.

Alternatively, it is possible to download the libraries and include them in your HTML page. The minified versions can significantly reduce the space required for storing, in this way the inclusion does not affect the application's performances. To include jQuery, jQuery UI and the UI theme CSS in HTML code, it's necessary to insert the following lines:

```
<link rel="stylesheet" href="scripts/jquery-ui-1.9.2.custom.css" />

<script src="scripts/jquery-1.8.3.js"></script>
<script src="scripts/jquery-ui-1.9.2.custom.js"></script>
```

3.2 jQuery Mobile

jQuery Mobile is a set of jQuery plug-ins and widgets that aim to provide a crossplatform API for creating mobile web applications. In terms of code implementation, jQuery Mobile is very similar to jQuery UI, but while jQuery UI is focused on desktop applications, jQuery Mobile is built with mobile devices in mind. In a matter of minutes, you can create mobile applications (apps) that run on every phone, tablet, desktop, and e–reader device available today. That's right, with a single jQuery Mobile codebase we can create a unified experience for nearly all consumers.

We will now list the major strengths that have made possible the affirmation of jQuery:

- jQuery Mobile apps are universally available to all devices with a browser and are optimized to run on nearly every phone, tablet, desktop, and e–reader device available today.
- jQuery Mobile applications can take advantage of the instant deployment capabilities we have grown accustomed to on the Web.
- A single jQuery Mobile code base will render consistently across all supported platforms without customizations. This is a very costeffective solution when compared to the alternative of building an app for each OS or client.
- jQuery Mobile is a simplified markup-driven framework that should appear very familiar to Web designers and developers.
- jQuery Mobile utilizes progressive enhancement techniques to render a very rich experience for all A-grade devices and provides a usable experience for older C–grade browsers.
- A jQuery Mobile UI will render responsively across devices of various sizes including phones, tablets, desktops, or TV's.
- jQuery Mobile supports a themable design that allows designers to quickly re–style their UI globally.

Figure 3.1: jQuery provides a consistent experience across all platforms.

3.3 Mantis Bug Tracker

The achievement of Quality certification has been made possible by the use of a management tool developed on an existing bug tracking software, called Mantis Bug Tracker (hereinafter referred to as MantisBT). Through specific adjustments, the company has obtained an interesting mechanism able to tracking the activities of the employees. The following description illustrates the structure of the system MantisBT considering all the settings that were applied for the adaptation to the company needs.

MantisBT is a web based bug tracking system that was first made available to the public in November 2000. MantisBT is developed in PHP, with support to multiple database backends including MySQL, MS SQL, PostgreSQL and DB2.

MantisBT, as a PHP script, can run on any operating system that is supported by PHP and has support for one of the DBMSes that are supported. MantisBT is known to run fine on Windows, Linux, OS/2, Mac OS X, System i and a variety of Unix operating systems.

Like in all bug trackers, the main entity of the system is the issue. The life cycle of an issue starts with its creation. It can be created via one of the following channels:

- MantisBT Web Interface. This is where a user logs into MantisBT and reports a new issue.
- SOAP API. Where an application automatically reports an issue into MantisBT using the SOAP API web services interfaces.

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 3.1:

Table 3.1: Nomenclature changes adopted.

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

Thus, from now on the new nomenclature is adopted.

3.3.1 Authorization and Access Levels

MantisBT uses access levels to define what a user can do. Each user account has a global or default access level that is associated with it. This access level is used as the access level for such users for all actions associated with public projects as well as actions that are not related to a specific project.

The default access levels supplied as standard with MantisBT are:

- viewer
- reporter
- updater
- developer
- manager
- administrator

Each features has several configuration options associated with it and identifies the required access level to do certain actions. So for a user to be able to report an issue against a public project, the user must have a project-specific or a global access level that is greater than or equal to REPORTER. The default value for the available access levels is '10:viewer, 25:reporter, 40:updater, 55:developer, 70:manager, 90:administrator'.

3.3.2 Ticket Status

An important part of ticket tracking is to classify tickets as per their status. Each team may decide to have a different set of categorization for the status of the tickets, and hence, MantisBT provides the ability to customize the list of statuses. MantisBT assumes that an ticket can be in one of three stages:

- opened
- resolved
- closed

Hence, the customized statuses list will be mapped to 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 to opened, "resolved" means resolved and "closed" means closed.

The Figure 3.2 shows schematically the status transitions.

Following is the explanation of what the standard statuses that are shipped with MantisBT means.

Table 3.2: Mantis ticket status.

Ticket Status	Description
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 reporter is suggesting in an ticket report, although they didn't yet attempt to reproduce what the reporter is referring 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 reporter 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. An ticket can be resolved with one of many resolutions (customizable). For example, an ticket can be resolved as "fixed", "duplicate", "won't fix", "no change required", etc. 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. It also typically hides the ticket from the View Tickets page. Some teams use "closed" to reflect sign-off by the reporter and others use it to reflect the fact that the fix has been released to customers.

3.3.3 Workflow

Now that we have covered how an issue gets created, and what are the different statuses during the life cycle of such issues, the next step is to define the workflow. The workflow dictates the valid transitions between statuses and the user access level required of the user who triggers such transitions; in other words, how issues move from one status to another and who is authorized to trigger such transitions.

MantisBT provides the ability for teams to define their own custom workflow which works on top of their custom status.

By default, there is no workflow defined, which means that all states are accessible from any other, by anyone, but the "Manage > Manage Configuration > Workflow Transitions" page allows users with ADMINISTRATOR access level to do the following tasks:

- Define the valid next statuses for each status.

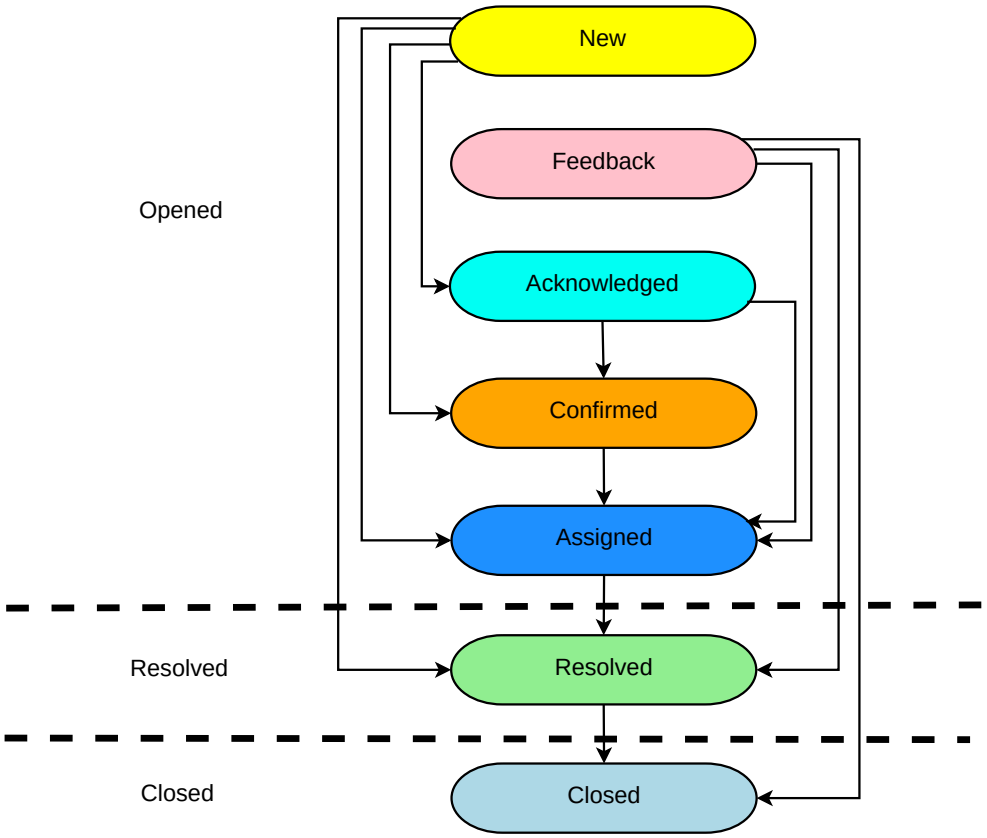

Figure 3.2: Diagram of status transitions.

- Define the default next status for each status.
- Define the minimum access level required for a user to transition to each status.
- Define the default status for newly created issues.
- Define the status at which the issue is considered resolved. Any issues a status code greater than or equal to the specified status will be considered resolved.
- Define the status which is assigned to issues that are re-opened.
- Define the required access level to change the workflow.

Note that the scope of the applied change is dependent on the selected project. If "All Projects" is selected, then the configuration is to be used as the default for all projects, unless overidden by a specific project.

3.3.4 Mantis Adjustments

Particular attention has requested the mapping between the system designed and the Mantis database structure. The correct path would be the addition of new tables and modifying existing ones to create a custom extension of the bug tracker. This solution, however, is risky and expensive for a company, as it requires the migration of data stored in the old structure into the new one, introducing the risk of data loss and any other inconsistencies due to the complex structure of the database.

The alternative solution was to simulate the designed architecture through the use of the tools provided by the basic version of Mantis, for example imagining the ticket like tables to which add custom fields and create special projects for the storage of data relating to products . Through these simple considerations, it is possible to obtain a structure that can contain all the information we need. On the other hand, analyzing in detail the possible scenarios that may be developed within the company, these simplifications are reasonable.

Below is the mapping between the entities identified in the design phase and the custom fields added in the Mantis ticket.

An additional custom field has been created to enable a more detailed classification of the collected data. The name of this field is "motivation" and is used to indicate the type of reporting, which may take only one among the following values: praise, suggestion and complaint.

Figure 3.3: Custom fields added.

A different approach was adopted for the management of products. In this case it was decided to use fake projects, created specifically to represent the products. In fact, unlike the components analyzed previously, the products are continuously updated and can not be managed by static structures. In this

Entity	Custom Field	Description
Action	action	The field, of enumeration type, allows the choice of value only within a defined and limited set of values. The value is assigned automatically by the instrument that deals with the management of the ticket.
Product	product	This field allows the inclusion of the product to which the reporting is related. The possible values to be assigned are handled at the application level through a list created on the basis of a query to the Mantis database, which returns the names of particular types of projects.
Tool	tool	The field represents the instrument from which the reporting arrives. Again, this field is populated automatically by the application that generates the reporting and its value is chosen from a finite set of possible values.
User	requestfrom	To allow the identification of the user who issued the alert has been added a special field that is filled with the customer xmpp username that makes the reporting.
Perception	score	In this field are represented the values that can assume the perceptions, from which you can extract the numerical score.

Table 3.3: Mapping between the designed system and the Mantis database structure.

way, when a user will make a report for a specific product, the application will simply have to download the list of projects that have the value "product" in the description field. Another advantage of this choice is the possibility of adding more sub-projects to each project, allowing the addition of properties to the project, for example the features characterizing the product.

The next step is the definition of workflow. The application relies on the exchange of messages between employee and customer. This allows the creation of real conversations, which are managed by the addition of notes in the ticket. The workflow is very simple and involves the use of only three states:

- **new**, when the message is created and has not yet received a reply;
- **feedback**, the ticket goes into this state when it receives the first response by the employee;
- **closed**, when the ticket is closed by the employee.

A reporting remains in the feedback state for the entire duration of the conversation. When the conversation ends, the status is set to closed. To the employee is given the ability to start the conversation and close it, as opposed to the customer who can only create reports and respond.

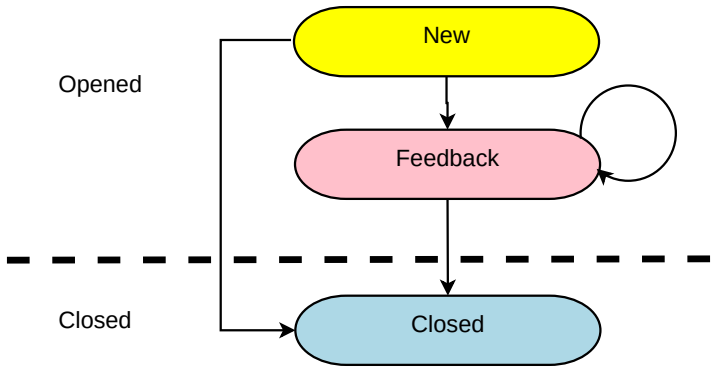

Figure 3.4: System workflow.

3.4 PhoneGap

"PhoneGap is an open source development tool for building fast, easy mobile apps with JavaScript."

The spread of the large number of operating systems has introduced a number of challenges associated with the development of mobile applications. First, there were BlackBerry and Symbian smartphones, then came the powerful iPhone and Android platforms. Recently, Microsoft has launched Windows 7 Phone and Samsung is releasing Bada. This leads to significant technical challenges for launching mobile applications on all platforms.

The browser world was largely fragmented until just a few years ago. Newer browser, 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. All of these mobile platforms support embedding browsers in applications.

This means that it is possible to imagine an application such as a web browser that displays HTML pages, realizing therefore a cross-platform technology. These embedded browsers are often referred as webviews. PhoneGap was made possible due to a commonality between all of the mobile platforms. If it were not for this common component, PhoneGap would not have been possible.

PhoneGap is a 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 such as Objective-C. The resulting applications are hybrid, meaning that they are neither truly native (because all layout rendering is done via web views instead of the platform's native UI framework) nor purely web-based (because they are not just web apps, but are packaged as apps for distribution and have access to native device APIs).

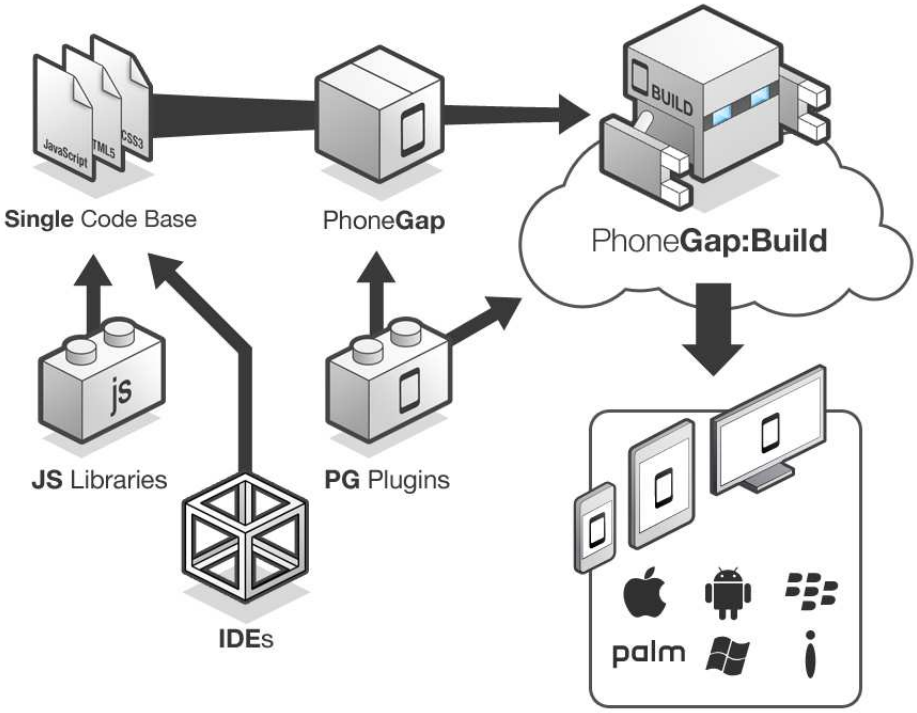

Figure 3.5: PhoneGap application architecture.

The software underlying PhoneGap is Apache Cordova. The core of PhoneGap applications use HTML5 and CSS3 for their rendering, and JavaScript for their logic. Using HTML5 alone would prevent the use of underlying hardware

such as the accelerometer or GPS, and browser support for HTML5 is not consistent across mobile browsers. To overcome these limitations, the PhoneGap framework embeds HTML5 code inside a native WebView on the device, using a Foreign Function Interface to access the native resources of the device.

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. As shown in Figure 3.6, an application build using PhoneGap will primarily have to parts:

1. The JavaScript Business Logic Part, which drives the UI and its functionality.

2. 2. The JavaScript Part, which accesses and controls the device.

3.5 XMPP

" XMPP is an XML-based set of technologies for real-time applications."

The eXtensible Messaging and Presence Protocol (XMPP) is, at its most basic level, a protocol for moving small, structured pieces of data between two places. From this humble basis, it has been used to build large-scale instant messaging systems, Internet gaming platforms, search engines, collaboration spaces, and voice and video conferencing systems.

Most social media constructs that have propelled web sites like Facebook, MySpace, and Twitter into the forefront are also baked into XMPP. Within XMPP, you'll find rosters full of contacts that create a social graph with directed or undirected edges. Presence notifications are sent automatically when contacts come online and go offline, and private and public messages are the bread and butter application of XMPP systems. Developers will sometimes choose XMPP as the underlying technology layer simply because it gives them many social features for free, leaving them to concentrate on the unique pieces of their application.

XMPP, like all protocols, defines a format for moving data between two or more communicating entities. In XMPP's case, the entities are normally a client and a server, although it also allows for peer-to-peer communication between two servers or two clients.

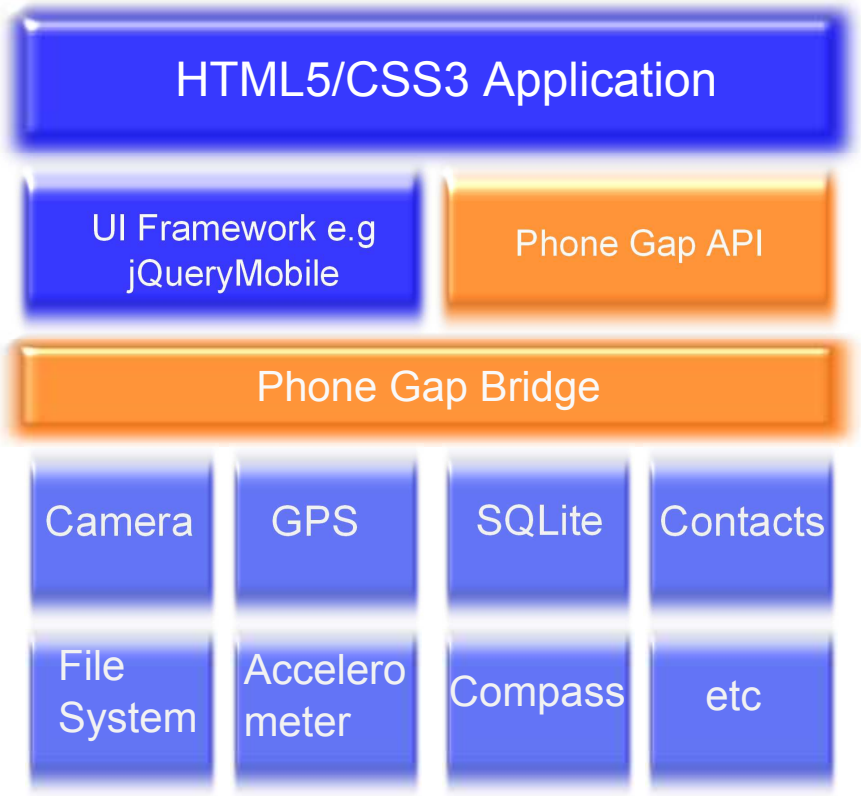

Figure 3.6: PhoneGap application architecture.

Data exchanged over XMPP is in XML, giving the communication a rich, extensible structure. One major feature XMPP gets by using XML is XML's extensibility. It is extremely easy to add new features to the protocol that are both backward and forward compatible.

XML is known primarily as a document format, but in XMPP, XML data is organized as a pair of streams, one stream for each direction of communication. Each XML stream consists of an opening element, followed by XMPP stanzas and other top-level elements, and then a closing element. Each XMPP stanza is a first-level child element of the stream with all its descendent elements and attributes. At the end of an XMPP connection, the two streams form a pair of valid XML documents.

XMPP stanzas make up the core part of the protocol, and XMPP applications are concerned with sending and responding to various kinds of stanzas. Stanzas may contain information about other entities availability on the network, personal messages similar to email, or structured communication intended for computer processing. An example stanza is shown here:

Listing 3.1: Example of stanza.

```
<message  to='elizabeth@longbourn.lit'
          from='darcy@pemberley.lit/dance'
          type='chat'>
  <body>What think you of books?</body>
</message>
```

XMPP is designed for the exchange of small bits of information, not large blobs of binary data. XMPP can, however, be used to negotiate and set up out-of-band or in-band transports, which can move large blocks from point to point.

The focus on small, structured bits of data gives the XMPP protocol extremely low latency and makes it extremely useful for real-time applications. These applications, which include collaborative spaces, games, and synchronization, are driving XMPP's growth in popularity as developers experiment with the real-time Web.

3.5.1 XMPP architecture

All good Internet technologies have an "architecture" (a way that various entities fit together, link up, and communicate). For example, the World Wide Web consists of millions of web servers running software like Apache, and many more millions of web clients (browsers) running software like Firefox, all using standard protocols and data formats like HTTP and HTML. As another example, the email infrastructure consists of millions of email servers running software like Postfix, and many more millions of email clients running software like Thunderbird, all using standard protocols like SMTP, POP, and IMAP.

Similarly, the Internet's infrastructure for instant messaging, presence, and other forms of real-time communication increasingly consists of hundreds of thousands of Jabber servers running software like ejabberd and Openfire, and millions of Jabber clients running software like Adium, Gajim, Pidgin, and Psi, all using the standard protocol called XMPP.

XMPP technologies use a decentralized client-server architecture similar to the architectures used for the World Wide Web and the email network. However, there are some important architectural differences between the Web, email, and Jabber.

When an user visits a website, his browser connects to a web server, but web servers typically do not connect to each other in order to complete a transaction (see Figure 3.7). Instead, the HTML of the web page may refer to other web servers (e.g., to load images or scripts), and user's browser opens sessions with those web servers to load the full page. Thus, the Web typically does not involve inter-domain connections (often called federation).

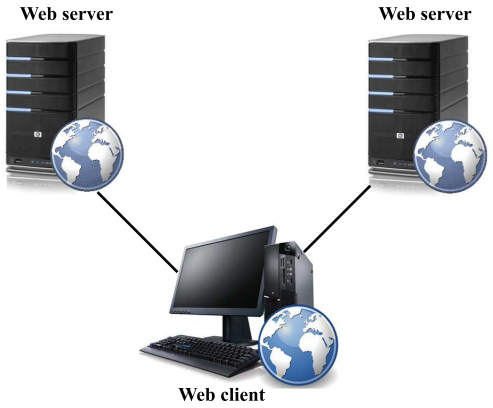

Figure 3.7: Web architecture.

When an user sends an email to one of his contacts at a different domain, his email client connects to his "home" email server, which then seeks to route the message to his contact. Thus, unlike the Web, the email system consists of a federated network of servers (see Figure 3.8).

However, user's message might be routed through multiple intermediate email servers before it reaches its final destination. Thus, the email network uses multiple hops between servers to deliver messages.

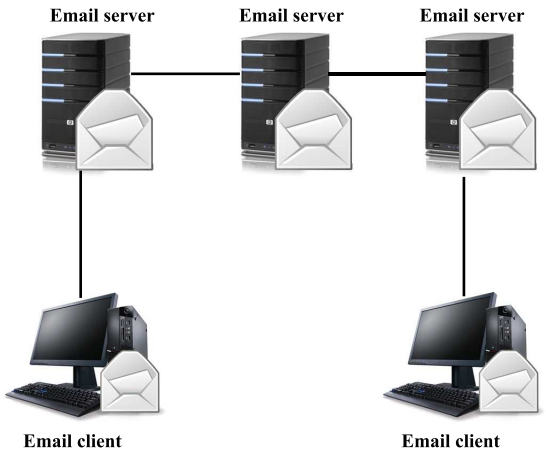

Figure 3.8: Email architecture.

Like email, but unlike the Web, XMPP systems involve a great deal of inter-domain connections. However, when an user sends an XMPP message to one

of his contacts at a different domain, his client connects to his "home" server, which then connects directly to his contact's server without intermediate hops (see Figure 3.9). This direct communication eliminates some common vectors for spam and unauthorized messages.

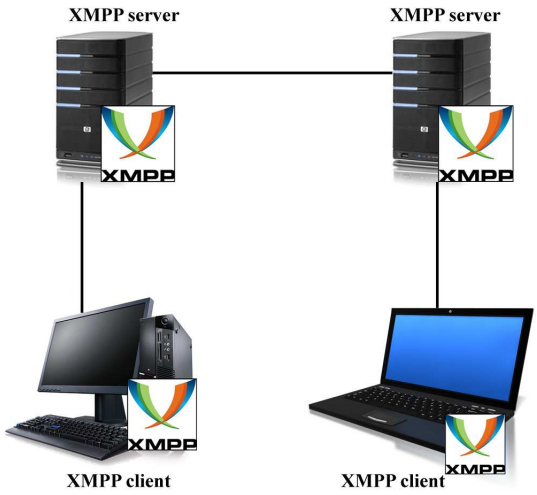

Figure 3.9: XMPP architecture.

This is just one of the many ways in which XMPP is designed for security. It also supports encrypted communications between endpoints through use of Transport Layer Security (TLS) and strong authentication mechanisms via Simple Authentication and Security Layers (SASL).

3.5.2 XMPP addressing

Every entity on an XMPP network will have one or more addresses, or JIDs. JIDs (short for jabber identifiers) can take a variety of forms, but they normally look just like email addresses. Each JID is made up of up to three pieces, the local part, the domain, and the resource. The domain portion is always required, but the other two pieces are optional, depending on their context.

The domain is the resolvable DNS name of the entity (a server, component, or plug-in). A JID consisting of just a domain is valid and addresses a server. Stanzas addressed to a domain are handled by the server itself and potentially routed to a component or plug-in.

The local part usually identifies a particular user at a domain. It appears at the beginning of a JID, before the domain, and it is separated from the rest of the JID by the @ character, just like the local part of an email address. The local part can also be used to identify other objects; a multi-user chat service will expose each room as a JID where the local part references the room.

A JID's resource part most often identifies a particular XMPP connection of a client. For XMPP clients, each connection is assigned a resource. If Mr. Darcy, whose JID is `darcy@pemberley.lit`, is connected both from his study and his library, his connections will be addressable as `darcy@pemberley.lit/study` and `darcy@pemberley.lit/library`. Like the local part, a resource can be used to identify other things; on a multi-user chat service, the resource part of the JID is used to identify a particular user of a chat room.

JIDs are divided into two categories, bare JIDs and full JIDs. The full JID is always the most specific address for a particular entity, and the bare JID is simply the full JID with any resource part removed. For example, if a client's full JID is `darcy@pemberley.lit/library`, its bare JID would be `darcy@pemberley.lit`. In some cases, the bare JID and the full JID are the same, such as when addressing a server or a specific multi-user chat room.

3.5.3 XMPP stanzas

A real-time messaging system using XMPP comprises three broad categories of communication:

- **Messaging**, where data is transferred between parties;
- **Presence**, which allows users to broadcast their online status and availability;
- **Info/query requests**, which allow an XMPP entity to make a request and receive a reply from another entity;

Each of them is delivered through a complete XML stanza (a discrete item of information expressed in XML). These stanzas are `<presence>`, `<message>`, and `<iq>`. The three basic stanzas make up the core XMPP toolset. Each type of stanza has its place and purpose, and by composing the right kinds of quantities of these stanzas, sophisticated behaviors can be achieved.

XMPP stanzas of all three types have the following attributes in common:

- `from`, the JID of the source XMPP entity;
- `to`, the JID of the intended recipient;
- `id`, an optional identifier for the conversation;
- `type`, this attribute specifies the specific kind of `<presence>`, `<message>` or `<iq>` stanza;
- `xml:lang`, if the contenti s human-readable, description of the messagge's language.

Presence stanzas

The `<presence>` stanza controls and reports the availability of an entity. This availability can range from simple online and offline to the more complex away and do not disturb. In addition, `<presence>` stanzas are used to establish and terminate presence subscriptions to other entities.

There are mainly four types of `<presence>` stanza:

- normal presence stanzas
- extending presence stanzas
- presence subscriptions
- directed presence

Hereinafter will be analyzed in detail all types of `<presence>` stanza, describing the basic features through examples.

Normal presence stanzas A normal `<presence>` stanza contains no type attribute or a type attribute that has the value unavailable or error. These stanzas set or indicate an entity's presence or availability for communication.

Users manipulate their own presence status by sending `<presence>` stanzas without a `to` attribute, addressing the server directly. Some examples are listed below:

- The first two stanzas set a user's presence status to online or offline, respectively. These are also typically the first and last presence stanzas sent during an XMPP session.

Listing 3.2: Example of presence stanza.

```
<presence />

<presence type='unavailable' />
```

- The `<show>` element is used to communicate the nature of the user's availability. The element is named "show" because it requests that the recipient's client use this information to update a visual indicator of the sender's presence. Only one `<show>` child is allowed in a <presence> stanza, and this element may only contain the following possible values: away, chat, dnd, and xa. These values communicate that a user is away, is interested in chatting, does not wish to be disturbed, or is away for an extended period.

Listing 3.3: Example of show tag.

```
<presence>
  <show>away</show>
</presence>
```

- A `<status>` element is a human-readable string that the user can set to any value in order to comunicate presence information. This string is generally displayed next to the contact's name in the recipient's chat client.

Listing 3.4: Example of status tag.

```
<presence>
  <show>away</show>
  <status>at the bathroom</status>
</presence>
```

- Each connected resource of a user has a priority between -128 and 127. This priority is set to zero by default, but can be manipulated by including a `<priority>` element in `<presence>` stanzas. Users with multiple simultaneous connections may use this to indicate which resource should receive chat messages addressed to their bare JID.

Listing 3.5: Example of priority tag.

```
<presence>
  <priority>10</priority>
</presence>
```

Extending presence stanzas It is tempting for developers to want to extend `<presence>` stanzas to include more detailed information such as the song the user is currently listening to or the person's mood. Because `<presence>` stanzas are broadcast to all contacts (even those that may not have an interest in the information) and constitute a large share of the network traffic in the XMPP network, this practice is discouraged.

Presence subscriptions The user's server automatically broadcasts presence information to contacts that have a presence subscription to the user. Similarly, users receive presence updates from all contacts for which they have a presence

subscription. Presence subscriptions are established and controlled by use of `<presence>` stanzas.

Unlike some social network and IM systems, presence subscriptions in XMPP are directional. If Elizabeth has a subscription to Mr. Darcy's presence information, this does not imply that Mr. Darcy has a subscription to Elizabeth. If a bidirectional subscription is desired, a subscription must be separately established in both directions. Bidirectional subscriptions are often the norm for human communicators, but many services (and even some users) are interested in only one of the directions.

Presence subscription stanzas can be identified by a `type` attribute that has a value of `subscribe`, `unsubscribe`, `subscribed`, or `unsubscribed`. The first two values request that a new presence subscription be established or an existing subscription be removed, and the other two are the answers to such requests.

The following example shows Elizabeth and Mr. Darcy establishing a mutual presence subscription:

Listing 3.6: Example of mutual presence subscription.

```
<presence from='elizabeth@longbourn.lit/outside'
          to='darcy@pemberley.lit'
          type='subscribe'/>

<presence from='darcy@pemberley.lit/library'
          to='elizabeth@longbourn.lit/outside'
          type='subscribed'/>

<presence from='darcy@pemberley.lit/library'
          to='elizabeth@longbourn.lit'
          type='subscribe'/>

<presence from='elizabeth@longbourn.lit/outside'
          to='darcy@pemberley.lit/library'
          type='subscribed'/>
```

Directed presence The final kind of `<presence>` stanza is directed presence. A directed presence stanza is a normal `<presence>` stanza addressed directly to another user or some other entity. These can be used to communicate presence to entities that do not have a presence subscription, usually because the presence information is needed only temporarily.

One important feature of directed presence is that the recipient of the presence information is automatically notified when the sender becomes unavailable

even if the sender forgets to notify the recipient explicitly. Services can use directed presence to establish temporary knowledge of a user's availability that won't accidentally get out of date.

Message stanzas

As their name implies, `<message>` stanzas are used to send messages from one entity to another. These messages may be simple chat messages that you are familiar with from other IM systems, but they can also be used to transport any kind of structured information.

A `<message>` stanza is *fire and forget*; there is no built in reliability, similar to email messages. Once the message has been sent, the sender has no information on whether it was delivered or when it was received. In some cases, such as when sending to a non-existent server, the sender may receive an error stanza alerting them to the problem.

Here is an easy example of `<message>` stanza:

Listing 3.7: Example of simple message stanza.

```
<message from='bingley@netherfield.lit/drawing_room'
         to='darcy@pemberley.lit'
         type='chat'>

    <body>Come, Darcy, I must have you dance.</body>
    <thread>4fd61b376fbc4950b9433f031a5595ab</thread>

</message>
```

The example shows a typical `<message>` stanza for a private chat, including a thread identifier.

Message types Several different types of `<message>` stanzas exist. These types are indicated with the type attribute, and this attribute can have the value `chat`, `error`, `normal`, `groupchat`, or `headline`. Sometimes the message's type is used to inform a user's client how best to present the message, but some XMPP extensions, multi-user chat being a prime example, use the `type` attribute to disambiguate context.

The type attribute of a `<message>` stanza is optional, but it is recommended that applications provide one. Also, any reply `<message>` stanza should mirror the type attribute received. If no type attribute is specified, the `<message>` stanza is interpreted as if it had a type attribute set to `normal`.

Table 3.4: Possible values for type attribute of a <message> stanza

Type	Description
chat	Messages of type chat are sent in the context of a one-to-one chat conversation. This type is the most common in IM applications, which are primarily concerned with private, one-to-one communication.
error	This type is used in reply to a message that generated an error. These are commonly seen in response to malferme addressing; sending a <message> stanza to a non-existent domain or user results in a reply stanza with the type attribute set to error.
normal	A <message> stanza with a type of normal has been sent outside the context of a one-to-one chat.
groupchat	The groupchat type is used for messages sent from multi-user chats. It is used to disambiguate direct, private messages from a multi-user chat participant from the broadcast messages that participant sends to everyone in the room. A private message has the type attribute set to chat, whereas a message sent to everyone in the room contains a type attribute set to groupchat.
headline	Messages of type headline are used mostly by automated services that do not expert or support replies. If automatically generated email had a type attribute, it would use a value of headline.

Message contents Though <message> stanzas are allowed to contain arbitrary extension elements, the <body> and <thread> elements are the normal mechanisms provided for adding content to messages. Both of these child elements are optional. The <body> element contains the human-readable contents of the message.

Conversations, like email, can form threads, where each message in a thread is related to the same conversation. Threads are created by adding a <thread> element to a <message> stanza. The content of the <thread> element is some unique identifier that distinguishes the thread. A reply stanza should contain the same <thread> element as the one it is a reply to.

IQ stanzas

The <iq> stanza stands for Info/Query and provides a request and response mechanism for XMPP communication. It is very similar to the basic workings

of the HTTP protocol, allowing both get and set queries, similar to the GET and POST actions of HTTP.

Each `<iq>` stanza is required to have a response, and, as mentioned previously, the stanza's required id attribute is used to associate a response with the request that caused it. The `<iq>` stanza comes in four flavors differentiated by the stanza's type attribute. There are two types of `<iq>` stanza requests, `get` and `set`, and two types of responses, `result` and `error`. Throughout the thesis these are often abbreviated as IQ-get, IQ-set, IQ-result, and IQ-error.

Every IQ-get or IQ-set must receive a response IQ-result or IQ-error. The following examples show some common `<iq>` stanzas and their possible responses. Note that unlike `<message>` and `<presence>` stanzas, which have defined children elements, `<iq>` stanzas typically contain only extension elements relating to their function. Also, each pair of `<iq>` stanzas has a matching id attribute.

Listing 3.8: Example of iq stanza.

```
<iq from='jane@longbourn.lit/garden'
    type='get'
    id='roster2'>

    <query xmlns='jabber:iq:roster'/>
</iq>

<iq to='jane@longbourn.lit/garden'
  type='result'
  id='roster2'>

    <query xmlns='jabber:iq:roster'>
        <item jid='elizabeth@longbourn.lit' name='Elizabeth'/>
        <item jid='bingley@netherfield.lit' name='Bingley'/>
    </query>
</iq>
```

Jane sent a roster retrieval request to her server and the server replied to Jane with her small roster.

The `<iq>` stanza is quite useful in any case where result data or simple acknowledgment is required. Most XMPP extension protocols use a mix of `<iq>` and `<message>` stanzas to accomplish their goals. The `<iq>` stanzas are used for things like configuration and state changes, whereas `<message>` stanzas are used for regular communication.

3.5.4 The connection life cycle

The three stanzas can accomplish virtually any task in XMPP when combined properly. However, sending stanzas usually requires an authenticated XMPP session be established. This section describes the other portions of an XMPP connection's life cycle, that are connection, stream set up, authentication, and disconnection.

Connection

Before any stanzas are sent, an XMPP stream is necessary. Before an XMPP stream can exist, a connection must be made to an XMPP server. XMPP includes some sophisticated support for establishing connections to the right servers.

Stream set up

Once a connection is established to a given XMPP server, an XMPP stream is started. An XMPP stream is a set of two XML documents, one for each direction of communication. The XMPP stream is opened by sending the opening `<stream:stream>` element, that is the root element of the XML document, to the server. The server responds by sending the response stream's opening `<stream:stream>` tag.

Once XMPP streams are open in both directions, elements can be sent back and forth. At this stage of the connection life cycle, these elements will be related to the stream and the stream's features.

First, the client sends the opening element to the server:

Listing 3.9: Client request for XMPP stream creation.

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               to='pemberley.lit'>
```

The `to` attribute contains the domain part of Alice's JID. It is the logical domain name, which might not be the same as the physical machine used for connecting.

Elements qualified by the http://etherx.jabber.org/streams namespace must always be prefixed with `stream:`.

The xmlns attribute specifies the default namespace for all XML sent over the stream (i.e., the namespace that applies if no other namespace is noted). Because this is a client-to-server stream, the default namespace is `jabber:client`.

The server first sends a `<stream:features>` element and immediately it tries to reach agreement with the client on how the connection will proceed. So, the server tells the client about the stream feature it support. These mostly relate to encryption and authentication options that are available. The server reply is listed below:

Listing 3.10: Server reply for XMPP stream creation.

```
<stream:stream xmlns='jabber:client'
               xmlns:stream='http://etherx.jabber.org/streams'
               from='pemberley.lit'
               id='893ca401f5ff2ec29499984e9b7e8afc'
               xml:lang='en' >

<stream:features>

    <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls' />
    <compression xmlns='http://jabber.org/features/compress'>
        <method>zlib</method>
    </compression>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
        <mechanism>DIGEST-MD5</mechanism>
        <mechanism>PLAIN</mechanism>
    </mechanisms>

</stream:features>
```

In this case, the server supports the following features:

- Encrypted connections with the XMPP profile of Transport Layer Security (TLS).
- Authentication via the Simple Authentication and Security Layer (SASL). In this case, the only supported authentication methods are the PLAIN mechanism and the DIGEST-MD5 mechanism.
- Stream compression via zlib.

Authentication

XMPP allows for Transport Layer Security (TLS) encryption, and most clients use this by default. Once TLS support is advertised by the server, the client starts the TLS connection and upgrades the current socket to an encrypted one without disconnecting. Once TLS encryption is established, a new pair of XMPP streams is created.

Authentication in XMPP uses the Simple Authentication and Security Layers (SASL) protocol, and depending on the server involved, can support a number of authentication mechanisms. Normally servers provide plain text authentication and MD5 digest-based authentication, but some servers support authenticating via Kerberos or special tokens.

These same encryption and authentication technologies are also used in many other protocols (email and LDAP are two examples) and common libraries exist for supporting TLS and SASL that can be used equally well for XMPP.

Once authentication is complete, a client must bind a resource for the connection and start a session. If you are watching XMPP traffic on the wire, you will see `<bind>` and `<session>` elements (inside `<iq>` stanzas) being sent to do these jobs. If the client does not provide a resource to bind, the server chooses one for it, usually randomly. Also, the server may alter the user's chosen resource even if the client provides one.

Disconnection

When users are done with their XMPP sessions, they terminate the sessions and disconnect. The most polite way to terminate a session is to first send unavailable presence and then close the `<stream:stream>` element.

By sending a final unavailable presence, the user's contacts can be informed about the reasons for the user's departure. Closing the stream explicitly allows any in-flight stanzas to arrive safely.

A polite disconnection would look like this:

Listing 3.11: Correct policy of disconnection.

```
    <presence type='unavailable'/>
</stream:stream>
```

3.6 Smack API

Smack is an Open Source XMPP client library for instant messaging and presence. Smack is meant to be easily embedded into any existing Java application. It has no external dependencies and is optimized to be as small as possible. We use this pure Java library into our applications to create simple XMPP integrations such as sending notification messages and presence-enabling devices.

The power of this library lies in its simplicity of use. Below, we illustrate its main functionality through few and intuitive lines of code.

The `XMPPConnection` class is used to create a connection to an XMPP server. Below is code example for making a connection:

Listing 3.12: Establishing a connection.

```
// Create a connection to the jabber.org server
// on a specific port.
ConnectionConfiguration config =
            new ConnectionConfiguration("jabber.org", 5222);
Connection connection = new XMPPConnection(config);
connection.login("username", "password");
connection.connect();
```

Once you've logged in, you can being chatting with other users by creating new Chat or GroupChat objects.

Each message to the XMPP server from a client is called a packet and is sent as XML. The `org.jivesoftware.smack.packet` package contains classes that encapsulate the three different basic packet types allowed by XMPP (message, presence, and IQ). Classes such as `Chat` and `GroupChat` provide higher-level constructs that manage creating and sending packets automatically, but you can also create and send packets directly. Below is a code example for changing your presence to let people know you're unavailable and "out fishing":

Listing 3.13: Writing packets.

```
Presence presence = new Presence(Presence.Type.unavailable);
presence.setStatus("Gone fishing");
connection.sendPacket(presence);
```

Smack provides two ways to read incoming packets: `PacketListener`, and `PacketCollector`. Both use `PacketFilter` instances to determine which packets should be processed. A packet listener is used for event style programming, while a packet collector has a result queue of packets that you can do polling and blocking operations on. So, a packet listener is useful when you want to take some action whenever a packet happens to come in, while a packet collector is useful when you want to wait for a specific packet to arrive. Packet collectors and listeners can be created using an `Connection` instance.

3.7 Java Servlet

Servlets are the Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building Web-based applications. Java Servlets are part of the Java Enterprise Edition (Java EE), so we will need to run our Java Servlets inside a Servlet compatible "Servlet Container" (e.g. web server) in order for them to work.

A Java Servlet is a Java object that responds to HTTP requests. It runs inside a Servlet container. A Java web application can contain other components

than servlets. It can also contain Java Server Pages (JSP), Java Server Faces (JSF) and Web Services.

The browser sends an HTTP request to the Java web server. The web server checks if the request is for a servlet. If it is, the servlet container is passed the request. The servlet container will then find out which servlet the request is for, and activate that servlet. The servlet is activated by calling the `Servlet.service()` method.

Once the servlet has been activated via the service() method, the servlet processes the request, and generates a response. The response is then sent back to the browser.

A servlet follows a certain life cycle. The servlet life cycle is managed by the servlet container and contains the following steps:

1. Load Servlet Class.
2. Create Instance of Servlet.
3. Call the servlets `init()` method.
4. Call the servlets `service()` method.
5. Call the servlets `destroy()` method.

Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. Step 4 is executed multiple times (once for every HTTP request to the servlet). Step 5 is executed when the servlet container unloads the servlet.

For every request received to the servlet, the servlets `service()` method is called. Servlets typically extend `HttpServlet` and override `doGet` or `doPost`, depending on whether the data is being sent by `GET` or by `POST`. If you want a servlet to take the same action for both `GET` and `POST` requests, simply have `doGet` call `doPost`, or vice versa. Here is how:

Listing 3.14: A simple Servlet.

```
1  public class SimpleHttpServlet extends HttpServlet {
2
3    protected void doGet( HttpServletRequest request,
4                          HttpServletResponse response)
5          throws ServletException, IOException {
6
7        doPost(request, response);
8    }
9
10   protected void doPost( HttpServletRequest request,
11                          HttpServletResponse response)
```

```
12          throws ServletException, IOException {
13
14        response.getWriter().write("GET/POST response");
15    }
16
17  }
```

Both doGet and doPost take two arguments: an `HttpServletRequest` and an `HttpServletResponse`. The `HttpServletRequest` lets you get at all of the incoming data; the class has methods by which you can find out about information such as form data, HTTP request headers, and the client's hostname. The `HttpServletResponse` lets you specify outgoing information such as HTTP status codes and response headers (`Content-Type`, `Set-Cookie`, etc.).

Most importantly, HttpServletResponse lets you obtain a PrintWriter that you use to send document content back to the client. For simple servlets, most of the effort is spent in println statements that generate the desired page.

Most servlets generate HTML. To generate HTML, you need of three steps:

1. Tell the browser that you're sending it HTML.
2. Modify the `println` statements to build a legal Web page.
3. Check your HTML with a formal syntax validator.

You accomplish the first step by setting the HTTP `Content-Type` response header to `text/html`, so the code would look like this:

```
response.setContentType("text/html");
```

Although HTML is the most common kind of document that servlets create, it is not unusual for servlets to create other document types.

You should note that you need to set response headers before actually returning any of the content with the `PrintWriter`. That's because an HTTP response consists of the status line, one or more headers, a blank line, and the actual document, in that order.

The second step in writing a servlet that builds an HTML document is to have your println statements output HTML. An example is shown below:

Listing 3.15: Reading and writing packets.

```
1  import java.io.*;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4
5  public class HelloServlet extends HttpServlet {
6    public void doGet(HttpServletRequest request,
```

```
7                       HttpServletResponse response)
8               throws ServletException, IOException {
9       response.setContentType("text/html");
10      PrintWriter out = response.getWriter();
11      out.println("<HTML>\n" +
12      "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
13      "<BODY>\n" +
14      "<H1>Hello</H1>\n" +
15      "</BODY></HTML>");
16    }
17  }
```

3.8 JSP

JavaServer Pages (JSP) technology enables Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems. As part of the Java technology family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

JSP technology uses XML-like tags that encapsulate the logic that generates the content for the page. The application logic can reside in server-based resources that the page accesses with these tags. Any and all formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build Web-based applications.

Servlets are indeed useful, and JSP by no means makes them obsolete. So, why use JSP? They are all tasks related to programming or data processing. But servlets are not so good at presentation. Servlets have the following deficiencies when it comes to generating the output:

- It is hard to write and maintain the HTML. Using print statements to generate HTML? Hardly convenient: you have to use parentheses and semicolons, have to insert backslashes in front of embedded double quotes, and have to use string concatenation to put the content together.
- You cannot use standard HTML tools. All those great Web-site development tools you have are of little use when you are writing Java code.

- The HTML is inaccessible to non-Java developers. If the HTML is embedded within Java code, a Web development expert who does not know the Java programming language will have trouble reviewing and changing the HTML.

JSP pages are translated into servlets. So, fundamentally, any task JSP pages can perform could also be accomplished by servlets. However, this underlying equivalence does not mean that servlets and JSP pages are equally appropriate in all scenarios. The issue is not the power of the technology, it is the convenience, productivity, and maintainability of one or the other. After all, anything you can do on a particular computer platform in the Java programming language you could also do in assembly language. But it still matters which you choose. JSP provides the following benefits over servlets alone:

- It is easier to write and maintain the HTML. Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.
- You can use standard Web-site development tools. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.
- You can divide up your development team. The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

JavaServer Pages technology is an extension of the Java Servlet technology. Servlets are platform-independent, server-side modules that fit seamlessly into a Web server framework and can be used to extend the capabilities of a Web server with minimal overhead, maintenance, and support. Unlike other scripting languages, servlets involve no platform-specific consideration or modifications; they are application components that are downloaded, on demand, to the part of the system that needs them. Together, JSP technology and servlets provide an attractive alternative to other types of dynamic Web scripting/programming by offering: platform independence; enhanced performance; separation of logic from display; ease of administration; extensibility into the enterprise; and, most importantly, ease of use.

Now, this discussion is not to say that you should stop using servlets and use only JSP instead. By no means. Almost all projects will use both. For some requests in your project, you will use servlets. For others, you will use JSP. For still others, you will combine them. You want the appropriate tool for the job, and servlets, by themselves, do not complete your toolkit.

Chapter 4

Capter III - Implementation

With the identification of technologies that can implement the system designed, starts the last phase of the work. But before starting the implementation, we need a general reflection to precisely define all the components that have an active role in the system. During the design phase were discussed three main actors who exchange messages, thus realizing the processes of collection and classification of data. But these three applications are the only players present in the architecture?

For answer this question we must take a small step back. To briefly summarize the key steps constituting the interaction between the various entities, we can say that no component communicates directly with each other, but all communications pass through the Mantis database, which stores the ticket representing the various reports. Communications between applications and Mantis are directed. What happens in case of changes to the Mantis database? What happens if you decide not to use Mantis as a support structure? The answer to these questions is simple and not pleasant: the applications are not able to cope with these eventualities and therefore to every change, applications should be heavily modified. To address these not unlikely possibility, has been realized the architecture that will be discussed below.

The basic idea is to abstract of some level the entire system to make applications independent from the technology used for managing the database. In this way, we try to mask the particular implementation of the database to allow interaction in any context. At this point it is necessary the addition of a new component to the system: the Java daemon. This component acts as an intermediary between applications and Mantis. Through the use of the Java daemon, applications can call generic methods regardless of the technology used.

How can app call the single methods of the Java daemon?

The communication between applications and Java daemon takes place through the exchange of XMPP messages. The daemon then appears on the outside as a

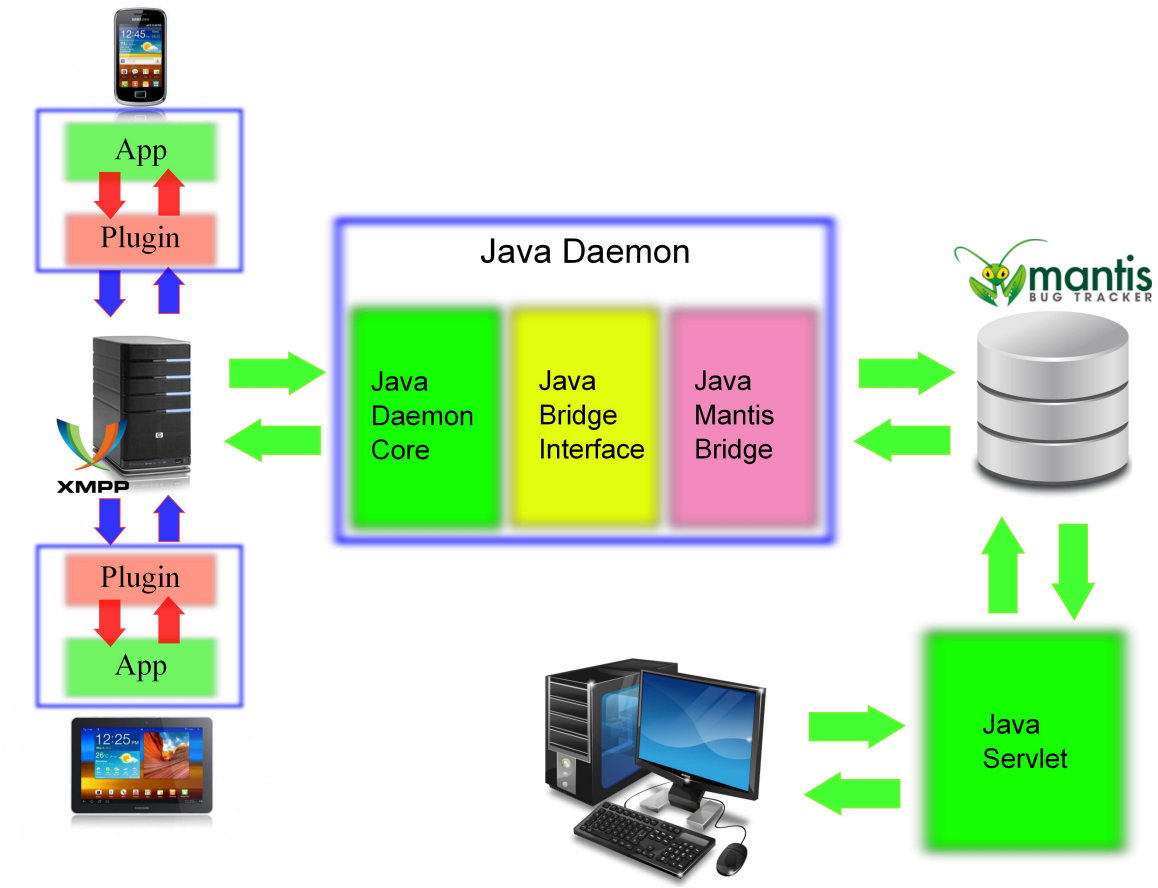

Figure 4.1: Architecture.

normal XMPP user. It runs continuously and is able to interpret the messages because they were written in a form known to him. After decoding the message and extract required parameters, daemon may invoke the desired method and return the result of processing through another XMPP message.

It is essential to find a common data format, to ensure that both parties are able to interpret the messages of the other. The chosen format is JSON. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is a text format that is completely language independent.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array,

vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by ,(comma).

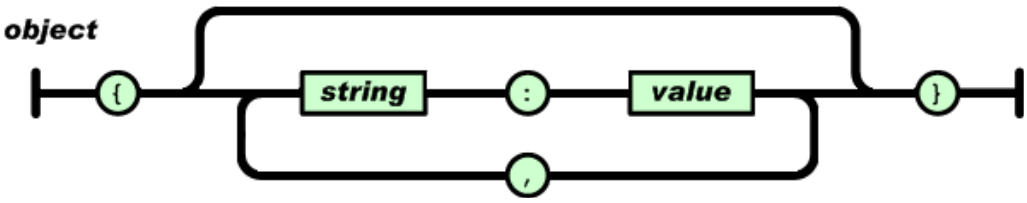

Figure 4.2: JSON object.

An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).

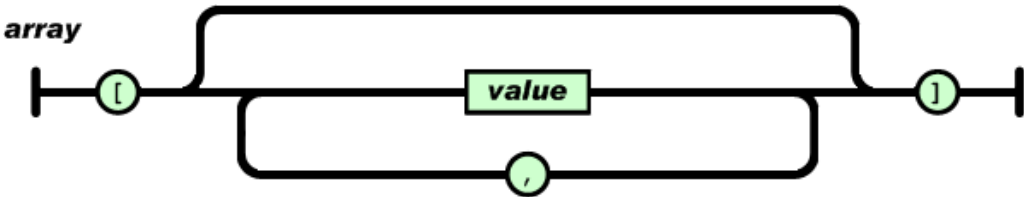

Figure 4.3: JSON array.

With the ability to convert JSON objects to strings, applications can exchange XMPP messages containing JSON objects in the message body. At the receiving is performed the reverse process, ie the app builds the JSON object starting from extracted string from the message body.

An example of JSON object contained inside an XMPP message is shown in Listing4.1.

Listing 4.1: JSON object contained inside XMPP message.

```
1
2 <message to="user02@alby" from="user03@alby" type="chat">
3     <body>
```

```
4           {"action":"insert_ticket",
5           "param":{"reporter":"user03@alby",
6           "product":"Product1",
7           "motivation":"suggestion",
8           "title":"Problema telefono",
9           "description":"La segreteria non parte! ",
10          "tool":"ticket_tracker"}}
11      </body>
12      <thread>sa9JR5</thread>
13  </message>
```

This allows the addition of custom fields that provide a structure to the message. For example, the selection of the method to invoke is made possible by controlling the value of the `action` field, set at the time of the request by individual applications.

A further level of abstraction was introduced through the creation of a bridge class between Java daemon and Mantis database. This has the purpose to collect in a single class the portions of code relating to the specific implementation. In case of technological variations, it is sufficient to change only this class and the entire architecture continues to operate.

Finally, the last great component is the tool for the analysis of the collected data. The mechanism used for other instruments is able to manage also this communication but the approach chosen is a bit different. Considering the purposes of the instrument, has been identified the most suitable technology. To summarize briefly the functioning, the user must log in to access the service and, if successful, the application will display the returned data. These requests are made on a frequent basis and only by company employees. From this analysis it can be said that the right technology for us are Java servlets used in combination with JSP. Through their use it is possible to implement a communication structure similar to that of other applications. The servlet is able to interpret the needs and uses a support class to query the database.

The operating system on which all mobile apps have been developed is Android. In order to manage XMPP communications in such devices it is necessary to use a modified version of the Java Smack library. This library is called asmack. The difficulty arose in this context is due to the fact that the application is written in Javascript and the library in Java. How do these two technologies communicate? The solution to this problem is provided by PhoneGap. It offers the possibility of developing plugins specific for our needs. Plugins are composed of a single JavaScript interface used across all platforms, and native implementations following platform-specific `Plugin` interfaces that the JavaScript will call

into. It should be noted that all of the core Cordova APIs are implemented using this exact architecture.

The entry point for any plugin is JavaScript. The reason developers use Cordova is so they can use and write JavaScript, not Objective-C, not Java, not C#. The JavaScript interface for your plugin is the front-facing and arguably most important part of Cordova plugin.

The one thing we must use to communicate between the Cordova JavaScript and native environments is the cordova.exec function. Here is an example:

```
1
2 cordova.exec( function(winParam) {},
3               function(error) {}, "service", "action",
4               ["firstArgument", "secondArgument", 42, false]);
```

The parameters explained in more detail:

1. `function(winParam) {}` Success function callback. Assuming your exec call completes successfully, this function will be invoked (optionally with any parameters you pass back to it).

2. `function(error) {}` Error function callback. If the operation does not complete successfully, this function will be invoked (optionally with an error parameter).

3. `service` The service name to call into on the native side. This will be mapped to a native class. More on this in the native guides below.

4. `action` The action name to call into. This is picked up by the native class receiving the exec call, and, depending on the platform, essentially maps to a class's method.

5. `[/* arguments */]` Arguments to get passed into the native environment.

An example is shown below:

```
1
2 function insertTicket( param, callbackInsertTicket ) {
3     cordova.exec( callbackInsertTicket, function(err) {
4         alert( 'Unable to insert ticket!' );
5     }, "AppPlugin", "insertTicket", [param]);
6 };
```

First, let's take a look at the last three arguments to the `exec` function. We will be calling the `AppPlugin` "service", requesting the `insertTicket` "action",

and passing an array of arguments containing only the JSON object with the required parameters, which is the first parameter into the `insertTicket` function.

App would then use it as follows:

```
1
2 insertTicket( param , function( result ){
3     alert("Success!");
4 });
```

The success callback passed into `exec` is simply a reference to the callback function defined as second parameter in the `insertTicket` call.

In Android, a plugin will consist of at least a single Java class that extends the `CordovaPlugin` class. A plugin must override one of the execute methods from `CordovaPlugin`. In addition to this, there is a best practice that the plugin should handle pause and resume events, and should handle message passing between plugins.

The JavaScript portion of a plugin always uses the cordova.exec method as follows:

```
exec(<successFunction>,<failFunction>,<service>,<action>,[<args>]);
```

This will marshal a request from the WebView to the Android native side, more or less boiling down to calling the action method on the service class, with the arguments passed in the args Array.

We have JavaScript to fire off a plugin request to the native side. So what does the final Android Java Plugin class look like?

What gets dispatched to the plugin via JavaScript's `exec` function gets passed into the `Plugin` class's `execute` method. The `execute` implementation look like this:

```
1
2 public boolean execute( String action,
3                         JSONArray args,
4                         CallbackContext callbackContext )
5                         throws JSONException {
6
7     if ( action.equals( "insertTicket" ) ){
8         JSONObject param = args.getJSONObject(0);
9         this.insertTicket( param, callbackContext );
10        return true;
11     }
12     return false;
13 }
```

We override the `execute()` method in order to receive messages from `exec()`. Our method first compares against action: this plugin only supports one action, the `insertTicket` action. Any other action will return false, which results in an error of type INVALID_ACTION (this will translate into an error callback invocation on the JavaScript side).

We compare the value of the action parameter, and dispatch the request off to a (private) method in the class, optionally passing some of the parameters to the method.

JavaScript in the WebView does not run on the UI thread. It runs on the WebCore thread. The `execute` method also runs on the WebCore thread. In order to interact with the GUI while waiting for a response from the plugin we need to implement the various methods as thread.

The final schema of the architecture is shown in Figure 4.1.

In the following section, we describe some implementation details regarding the individual components of the system, with the aim to highlight the peculiarities added in phase of realization.

Customer App

The customer application has been realized following the prototype proposed in the design phase. It consists of a total of five views. During its development, it was necessary to make some small additions.

The first improvement is the change of screen for ticket detail of reports in resolution, through discrimination between report that have already been answered and reports still unanswered. Depending on the status of the ticket, the application determines whether the user can respond to the reporting. So, if the state is `feedback`, then is added button to access the screen, otherwise, the button is not created. The two screens are shown in Figure 4.4.

Another enhancement is the addition of a system for receiving notifications that arrive when a report is updated by the employee. In this way, if the user is online at the time of receipt of the notification, he is informed of the update.

A characterizing feature of the application is the plugin, as it offers a set of actions that implement the functionalities identified in the design phase. You can call these actions by Javascript code through the function cordova.exec. They are summarized briefly in Table 4.1.

Besides the realization of the plugin, another significant challenge is the management of the GUI. The goal is to create an interface as user-friendly as possible. Among the small adjustments adopted appears the presence of loaders that are displayed when the user waits for responses from the plugin. They are used to inform the user about actions being performed. Another little detail is the management text areas on response screen. Text areas, in contrast to other

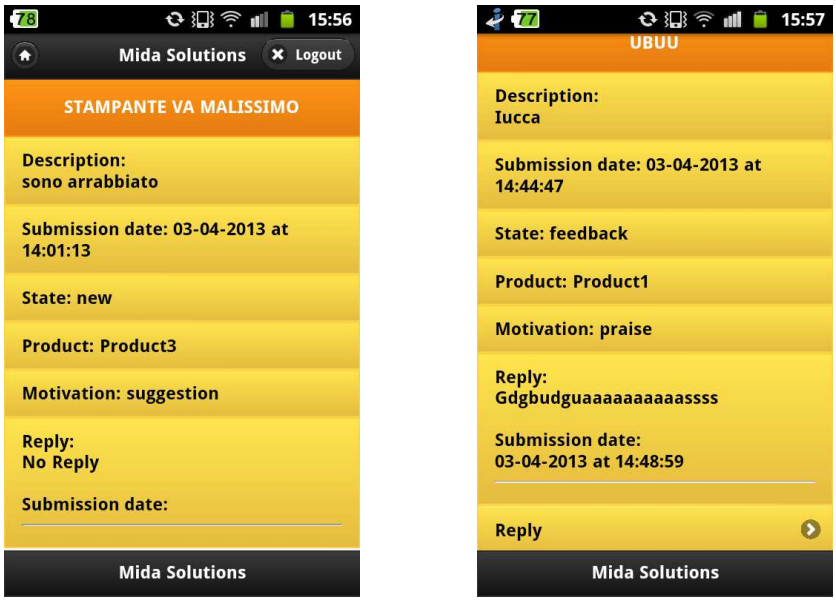

Figure 4.4: Different screens for reports with status new and feedback.

Table 4.1: Actions acknowledged by plugin of customer app.

Action	Description
connectXMPP	Establishes the connection to the XMPP server and login into it.
disconnectXMPP	Disconnects the current user from the XMPP server.
getMyTickets	Requires reports made by the current user.
getProducts	Requires the list of available products.
insertTicket	Performs the insertion of a new reporting.
updateTicket	Updates a reporting by adding the new user reply.
listenNews	Waits to receive notifications about updating of reports made.

HTML elements, allow the correct formatting of the text and the ability to copy the text contained in them. It was also added a specific plugin that takes care

of dynamically adapt the size of the text area according to its content.

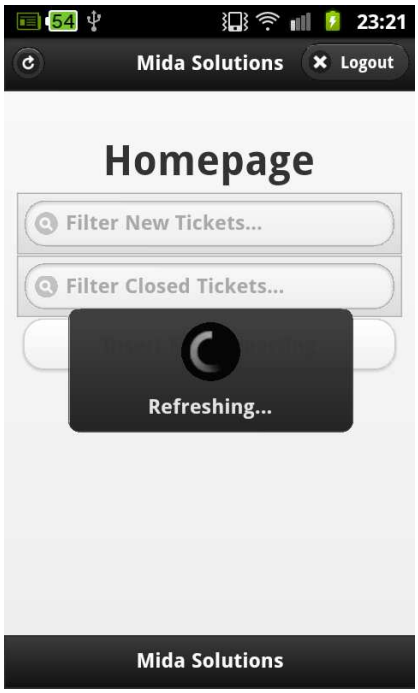

Figure 4.5: Loaders used to show what application is doing.

Employee App

Even the employee app reflects the storyboard suggested during the design phase. Despite this, the management of some components has required special attention in development phase.

One of the main difficulties faced is the realization of the system for receiving notifications. The main problem is caused by the need to leave continuously in listening the method that handles the notifications. The default mechanism that regulates the communication from plugin towards application is characterized by the invocation of the method `callbackContext.success()` in case of success and the method `callbackContext.error()` in case of error. This type of approach, however, does not fit to our case because the call to these methods involves the end of the method which satisfies the request.

The technique used is slightly more complex. Each time the application invokes a plugin method, plugin stores the id of the calling method callback. After processing, the evoked method returns the result going to call the callback

through the id previously stored. In this way it unambiguously identifies the method that made the request. Once developed this system, you can leave running the method for handling notifications through the use of a specific object.

After receiving the new message you must notify the user somehow. It was decided to implement this functionality in a complete manner, through the use of an Android plugin for displaying messages in the notification bar. Thus, for each new message received, notification plugin displays reporting title and reporter. The added value provided by the plugin, however, lies not only in the notifications, but also in the ability to run in the background the app. The user in this way can start the application and continue to carry out its activities. As soon as a message arrives, a notification signals the event, allowing user to open application.

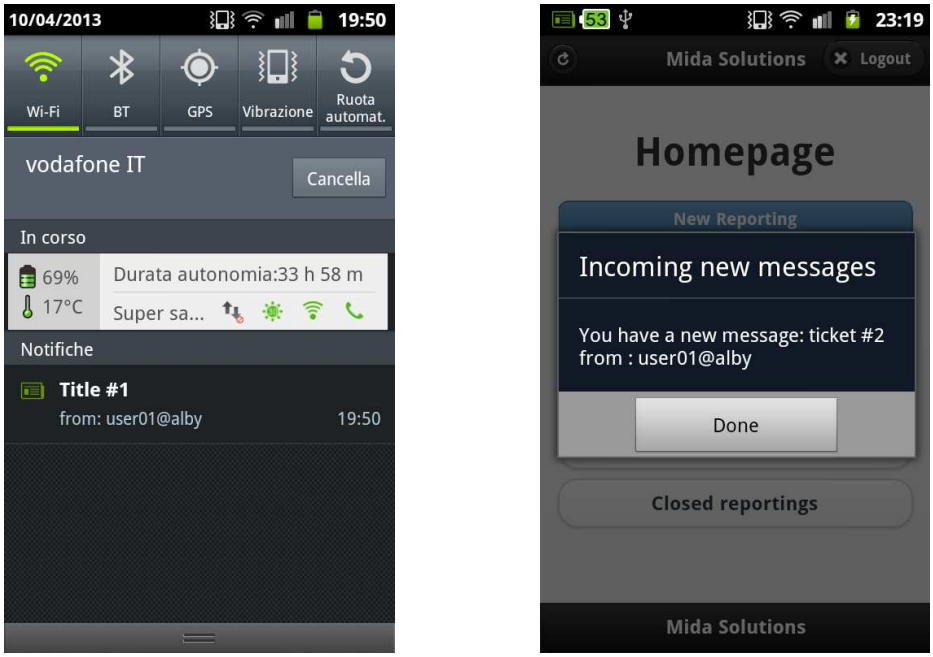

Figure 4.6: Notification system for incoming messages.

Like for user app, employee app has a plugin characterized by a series of actions that can be called from JavaScript code. The list of these actions is shown in Table 4.2.

Analysis Tool

The logic of adaptation to technological change has also been applied to the instrument of analysis of the collected data. In fact, the servlet to handle requests

Table 4.2: Actions acknowledged by plugin of employee app.

Action	Description
connectXMPP	Establishes the connection to the XMPP server and login into it.
disconnectXMPP	Disconnects the current user from the XMPP server.
getAllTickets	Requires all reports.
getProducts	Requires the list of available products.
updateTicket	Updates a reporting by adding a new reply or by setting the status to closed.
listenNews	Waits to receive notifications of new reports.

uses an object that implements a generic interface. This object contains the code for the specific technology. In the face of technological changes is therefore sufficient to modify this item.

The functioning of the servlet is very intuitive, as it provides a number of methods that can query the Mantis database and return data appropriately grouped. The selection of the method to invoke takes place through the use of the parameter `action`. The method of interaction with the servlet is similar to that for the interaction with the Java daemon. The possible values that can take the field `action` are mainly three:

- **login**. It logs in to the Mantis database with the credentials passed by the user through the login form;
- **get_products**. It returns all reports divided by products.
- **get_users**. It returns all reports divided by users.

The values returned by the servlet change according to evoked action. At the first access, user calls logging method, which, if successful, returns an html page. In later stages, the application communicates with the servlet through ajax calls, made by JavaScript code.

AJAX (an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With Ajax, web applications can send data to, and

retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data can be retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not required (JSON is often used instead), and the requests do not need to be asynchronous.

Fortunately for us, jQuery provides several methods for AJAX functionality. It is possible to make an ajax call as follows:

```
$.ajax( [settings] )
```

where `settings` is a set of key/value pairs that configure the Ajax request. All settings are optional. The following are listed in Table 4.3:

Table 4.3: Settings used in ajax calls.

Name	Description
async	By default, all requests are sent asynchronously (i.e. this is set to true by default). However, in our application were used only synchronous calls.
data	Data to be sent to the server. It is converted to a query string, if not already a string. It's appended to the url for GET-requests.
dataType	The type of data that you're expecting back from the server. The available types (and the result passed as the first argument to your success callback) are: xml, html, script, json, jsonp and text.
error	A function to be called if the request fails. The function receives three arguments: the jqXHR object, a string describing the type of error that occurred and an optional exception object, if one occurred.
success	A function to be called if the request succeeds. The function gets passed three arguments: the data returned from the server; a string describing the status; and the jqXHR object.
type	The type of request to make (`POST` or `GET`), default is `GET`.
url	A string containing the URL to which the request is sent.

One of the ajax calls made by the application is shown below:

```
$.ajax({
    type: "POST",
    url: "Analyzer",
    data: dataString,
    dataType: "json",
    async: false,

    success: function( data, textStatus, jqXHR){
        products = new Array();
        $.each(data, function(k, product){
            products.push(product);
        });
    },

    error: function(jqXHR, textStatus, errorThrown){
        alert("Error in products request");
    }
});
```

In the implementation phase, it was decided to equip the application of a tool to select time intervals within which to evaluate the performance of the individual products. It is thus possible to analyze the score of a product after the occurrence of an event, or in a given month. When you select one of the two dates, automatically sets the selectable values in the other, in order to avoid the selection of invalid intervals. The filter uses these intervals to display only reports that have the date of submission within the selected range.

Java Daemon

At startup, the Java daemon connects to the XMPP server as a normal user. Subsequently it creates a packets listener to read incoming messages. The method, that is executed on receipt of each new message, performs an initial check on the type of incoming packets. In the current version of the application, the only XMPP messages analyzed are those of type `chat`, but the extension to other types is not difficult to achieve. From the content of the message body is removed the string from which you can create the JSON object representing the request. The value extracted from the `action` field of the object is used to select the desired method.

Regardless of the request, the application creates an object of type MantisBridge which in turn implements the Bridge interface. With this object the application interacts with Mantis. Once it has finished querying, the results are

organized into JSON objects that application will forward to the senders. In the case of insertions and updates, notifications are shipped respectively to employee and user.

To interact with Mantis has been used a special Java library called mantis axis soap client. One of the advantages of using Mantis is that it does not require the creation of sessions because every call is an end in itself. So each request creates an object that allows you to interact with Mantis. The code required to do this is shown in Listing4.2.

Listing 4.2: Creation of the object responsible of communication with Mantis.

```
1 url=new URL(server);
2 MantisConnectLocator mcl=new MantisConnectLocator();
3 MantisConnectPortType portType=mcl.getMantisConnectPort(url);
```

Through the `portType` object you can make calls to methods. The main difficulty in working with Mantis is given by the management of the custom fields, since Mantis can not know a priori how many and what are the fields added by administrator.

4.1 Technical Analysis

To better understand the architecture thus far discussed, it was decided to observe the entire life cycle of a request, through the analysis of the code portions associated to the various steps.

Will be studied below the steps constituting the two main interactions: communication between mobile applications and Mantis, and the communication between the analysis tool and Mantis.

4.1.1 Communication between customer app and Mantis

Regarding the first interaction, the action choice for this trip in the code is the request of reports made by the specific user. This request is made automatically at app startup or when the user decide to update its homepage. It goes through all the main tools and therefore allows the construction of a general view enough detailed.

Java Daemon

The first step needed to the configuration of the system is the startup of the Java daemon. This action is composed of three sub-phases, whose purpose is to manage XMPP communications. They are:

- connecting to the XMPP server;
- sending a presence stanza;
- adding a packets listener.

To implement these actions has been used Smack library, which provides all the methods we need. The connection is made through the creation of a global `XMPPConnection` object. It manages the session and therefore all operations must pass through it.

Listing 4.3: XMPP connection configurations.

```
1 ConnectionConfiguration config =
2       new ConnectionConfiguration("192.168.1.11", 5222);
3
4 XMPPConnection connection = new XMPPConnection(config);
5 connection.connect();
6 connection.login("myUsername","myPassword");
```

The first statement is used to create an object that contains the address of the server and the port with which you want to connect. If the login is successful, then the Java daemon is connected to the server.

Once logged to the server you need to send a presence stanza to communicate to the server that you are available to receive messages. To do this, the library comes to help us by providing the Presence class, created specifically to manage the presence stanza in the most intuitive way.

Listing 4.4: Presence notification.

```
1 Presence presence=new Presence(Presence.Type.available);
2 presence.setMode(Mode.available);
3 connection.sendPacket(presence);
```

The last operation of daemon initialization consists in adding a listener for the management of messages. It's necessary register a packet listener with this connection. Another main component is the packet filter, that determines which packets will be delivered to the listener. It was decided to only accept messages of chat type because the operations associated with the messages require the presence of both parties. The listener method, that will be called every time a new packet is found, is `processPacket`. In our case, it consists of a single call to `onPacketReceived` method.

Listing 4.5: Adding packet listener.

```
connection.addPacketListener( new PacketListener() {
    public void processPacket( Packet packet ) {
        onPacketReceived( packet );
    }
}, new MessageTypeFilter(Message.Type.chat));
```

`onPacketReceived` extracts the text contained in the message body and builds the JSON object associated, from which is derived the value of the `action` field, which is used for the selection of the method to evoke. After performing these preliminary operations, the daemon is ready to receive requests.

Customer App

The first page displayed by app is the login page. The user inserts his credentials to login. At this point, the application invokes the method `connectXMPP` of its plugin, passing as parameters the information entered by the user in the JSON form and the function to be performed in case of success.

Listing 4.6: client.js

```
var conn = new Object();
conn.username = $('#usernameXMPP').val();
conn.password = $('#passwordXMPP').val();
conn.server = $('#serverXMPP').val();

connectXMPP( conn, function( result )
{
    // Javascript code here.
});
```

The called method is defined, within the file plugin.js, in the following way:

Listing 4.7: plugin.js

```
function connectXMPP( loginInfo, callbackLogin ){
    cordova.exec( callbackLogin,
        function(err){ alert( 'Connection error!!!' ); },
          "AppPlugin", "connectXMPP", [ loginInfo ] );
};
```

Through this definition, we create a correspondence between the function called by the JavaScript code and the plugin method.

Plugin

The plugin intercepts the call to its execute method, and compares the value of the action with the names of its methods. When it finds a match, it extracts the parameters and calls the method for the connection with XMPP server.

Listing 4.8: AppPlugin.java

```
1 if ( CONNECT_XMPP.equals( action ) ){
2     JSONObject loginInfo = args.getJSONObject(0);
3     this.connectXMPP(loginInfo, callbackContext);
4     return true;
5 }
```

`connectXMPP` performs the same operations carried out in the initialization phase of the Java daemon. As already said, the most important task of this step is the addition of the packets listener. If the login is successful, it returns a JSON object having only the reply field with `success` value.

Customer App

The app performs a success callback which controls the value of the `reply` field and calls the function `getMyTickets` to download all the reports made by the current user. This function has a single parameter: the function to execute on success.

Plugin

The plugin evokes the method associated with the action, which creates a JSON object that contains a field with the desired action and another containing the XMPP username of the applicant. The so composed message is sent to the Java daemon. The last line of code stores the id of the current callback, so afterwards you can send the reply to the correct method.

Listing 4.9: AppPlugin.java

```
1  String name =
2      StringUtils.parseBareAddress(connection.getUser());
3
4  JSONObject request = new JSONObject();
5  request.put("action","get_tickets");
6  request.put("username",name);
7  String stringRequest = request.toString();
8  chat.sendMessage(stringRequest);
9
10 getCallbackID = callbackContext.getCallbackId();
```

Java Daemon

Java daemon intercepts the new message and parses it, creating the JSON object from the string contained in the body of the message. Of this object, it is first analyzed the `action` field for the identification of the corresponding method.

Listing 4.10: JavaDaemon.java

```
1  Message message = (Message)packet;
2  String json = message.getBody();
3
4  JSONObject received = new JSONObject(json);
5  String action = received.getString("action");
6
7  if (action.equalsIgnoreCase("get_tickets"))
8  {
9      String name = received.getString("username");
10     getTickets(message.getFrom(), name);
11 }
```

`getTickets` takes care of retrieval of the requested information. In our case, it instantiates an object of `MantisBridge` class to which passes a JSON object containing two information: the credentials to log in as administrator on Mantis and the JID of the applicant.

MantisBridge.java

A first difficulty faced has occurred in the restitution of ticket. The default behavior of Mantis is to return all tickets, except those that were closed. This is due to the original nature of the bug tracker. So it was necessary to create a special filter called `all_issues`, in Mantis configurations, able to return all tickets. As a preliminary step, the method checks the presence of this filter and, if successful, it returns all tickets stored in Mantis.

Listing 4.11: MantisBridge.java

```
1  IssueData[] allIssues =
2    portType.mc_filter_get_issues( user,
3      pwd,
4      new BigInteger("" + ProjectId ),
5      filter_id,
6      new BigInteger("" + PageNumber ),
7      new BigInteger("" + PerPage ) );
```

The next step is the extraction of the interest values and storing them in a JSONArray, that is an array of JSON objects. Tickets are filtered based on the use of the XMPP username passed as a parameter to the method: are added to JsonArray only those ticket whose `requestfrom` field coincides with the XMPP username.

Listing 4.12: MantisBridge.java

```
JSONArray tickets = new JSONArray();

for (IssueData ticket : allIssues){
  JSONObject item = new JSONObject();

  // Custom fields analysis

  if(item.getString("reporter").equals(usernameXMPP)){
    item.put("id",ticket.getId());
    item.put("title", ticket.getSummary());
    item.put("project",
      new JSONObject().put("name",
                ticket.getProject().getName()));
          ...
    }
    tickets.put(item);
}
```

Java Daemon

If all goes well, the method returns the JsonArray. After being converted to a string, it is placed in the `reply` field of a JSON object, to which is also added the `action` attribute with value `return_tickets`. The response object is converted to a string and sent via an XMPP message to the user who made the request.

Plugin

On receiving a new message, the packet listener of the plugin creates the JSON object from the message and it extracts the value of the `action` attribute. Based on this value, it performs the corresponding operation. In this case, it is limited to the delivery of the JSONArray containing all the reports.

Listing 4.13: AppPlugin.java

```
if(message.getType() == Message.Type.chat){
    JSONObject reply = new JSONObject( message.getBody());
```

```
3       String action = reply.getString("action");
4
5       if(action.equalsIgnoreCase("return_tickets")){
6         JSONArray tickets =
7         new JSONArray(reply.getString("reply"));
8       PluginResult pluginResult =
9         new PluginResult(PluginResult.Status.OK, tickets);
10
11      pluginResult.setKeepCallback(false);
12      webView.sendPluginResult(pluginResult, getCallbackID);
13  }
```

Customer App

When you receive an object containing all the tickets you can perform all the calculations directly from JavaScript code. At this point, the route is complete. As you can see, the code is very intuitive and is based on a non-rigid structure which is translated in the ease of adding new functionality. The life cycle of the application for employee is very similar to that described.

4.1.2 Communication between analysis tool and Mantis

The other major system interaction occurs between the tool for data analysis and Mantis database. This type of interaction is made possible by the use of a servlet that works in a very similar way to the Java daemon: based on the passed parameters is evoked a specific method. In this case the involved entities are only two, and then the communications management is much simpler than the case analyzed above.

The process, in part already described, starts with login of the employee. The servlet, still running on the server, checks if there is a user with these credentials and if he has permission to work with Mantis database. In any case, application forwards the request from servlet to `index.jsp`.

Listing 4.14: Analyzer.java

```
1  username = request.getParameter("username");
2  password = request.getParameter("password");
3  action = request.getParameter("action");
4
5  ...
6
7  switch(action){
8    case "login":
9         request.getRequestDispatcher("index.jsp")
10             .forward(request, response);
11        JSONObject login_object = new JSONObject();
12        login_object.put("username", username);
13        login_object.put("password", password);
14        login_object.put("server", server);
15
16        LoginInterface login = new LoginInfo();
17        isConnected = login.login(login_object);
18        ...
19        break;
20 ...
21
22 }
```

The user can then interact with the page, querying the servlet through the javascript code. Requests are made by Javascript Ajax calls. At startup, the application requires the list of comments grouped by product and by user. The servlet processes requests and uses `PrintWriter` object, returned by the `getWriter()` method , to send data to the applicant.

Listing 4.15: Analyzer.java

```
case "get_products":
    JSONArray products = new JSONArray();
    products = proc.getProducts();
    PrintWriter out = response.getWriter();
    out.print(products);
    out.close();
    break;
```

Once we have received information requested, through JavaScript code, the application dynamically creates the various components of the page, allowing the user to perform the required checks.

Bibliography

[1] J. Moffitt, *Professional XMPP Programming with JavaScript and jQuery*. Wiley Publishing, 2010. ISBN: 978-0-470-54071-8.

[2] P. Saint-Andre, K. Smith, and R. Troncon, *XMPP: The Definitive Guide*. O'Reilly, 2009.

[3] R. Gatol, *Beginning PhoneGap - Mobile Web Framework for JavaScript and HTML 5*. Apress, 2011.

[4] "PhoneGap – a framework for creating mobile apps." http://phonegap.com.

[5] "JQuery – a library for writing javascript." http://jquery.com.

[6] "JQueryUI – a library for creating gui for desktop applications." http://jqueryui.com.

[7] "JQuery Mobile – a library for creating gui for mobile apps." http://jquerymobile.com.

[8] "Mantis bug tracker, a free popular web-based bugtracking system." http://www.mantisbt.org.

[9] "Smack API – an open source xmpp library client." http://www.igniterealtime.org/projects/smack/index.jsp.

[10] "Java servlet and jsp: Technology overview." http://www.oracle.com/technetwork/java/index.html.

List of Tables

List of Figures