Software group report

Facetime Karsten Becker, Agusti Tomas, Farhad Shams

Introduction

Software for Facetime

To provide a great user experience to the customers of Facetime products, a well chosen Software architecture is required. The power of the software determines which functionalities are available to the user. One of the primary goals is to be able to deliver rich functionalities to an easy to use interface. Another requirement is to be able to produce a broad base of products based on this architecture. Of course this functionality should be developed in an economic amount of time without sacrificing the quality. All of those goals have been archived and we will describe here how.

Requirements towards the software

When our product succeeds in the market we are facing a potentially high volume of simultaneous users. Maybe up to hundreds of thousands user try to communicate with each other. The amount of data that is transfered for each message is very low. Maybe some bytes per message. Because we have agreed on using GSM there were 2 possible types to transfer data.

The first one is to use SMS. SMS has the advantage of being delivered securely and within seconds. Unfortunately the telcos charge a lot of money for each SMS. This would require the watch user to pay money for each emoticon that is sent and would make the watch less attractive for impulsive sending.

The other choice is having a steady data connection. In the current stage the telcos have invested much money into their networks for enabling data transfers. The price per transferred volume is declining steadily. Our watch would only use a very small amount of data each day. If we could get this volume for free, our watch would not require any kind of subscription and sending emoticons would be totally free, supporting the impulsive communication.

Getting the traffic for free

Assume a person receives a negative emoticon like :-(. This emoticon does not transport enough information to satisfy the receiver. The receiver thus wants to have more information about the received emoticon. Because the watch does not provide any additional means of communication other than emoticons the receiver choses the next possible means of communication. Which in this case is very likely to be the mobile phone. From a telco point of view a small amount of data initiated a possibly very deep means of communication. Such as a long telephone call or some SMS conversation. This provides some additional revenue for the telco.

Facetime thus could cooperate with the telco to get this little amount of volume for free, while the telco in return gets an increased Revenue. With the current provided amount bandwidth the costs for providing this volume is very low. Facetime would also benefit of having a very strong partner in distribution.

Requirements for the server

As we have chosen to pursue the data communication path the challenges we are facing on the server side look like this:

- Handle possibly millions of simultaneous connected users
- Handle thousands of messages each day
- Provide privacy options
- Be extensible for future additions of features
- Do not require expensive hardware
- Easily interoperate with other services

All those requirements can be satisfied today with existing software. The XMPP provides solutions for those requirements. There are several server implementations for this protocol available. One of this is Jabberd. The code base for this is available in GPL and has been developed for several years.
With XMPP a watch represents a user logged into this server. Upon registration it received a unique name which it uses for communication. The web interface, SMS gateway and the so called server are logged in a regular users too. All communication is packed into the XMPP and relayed through the Jabberd.

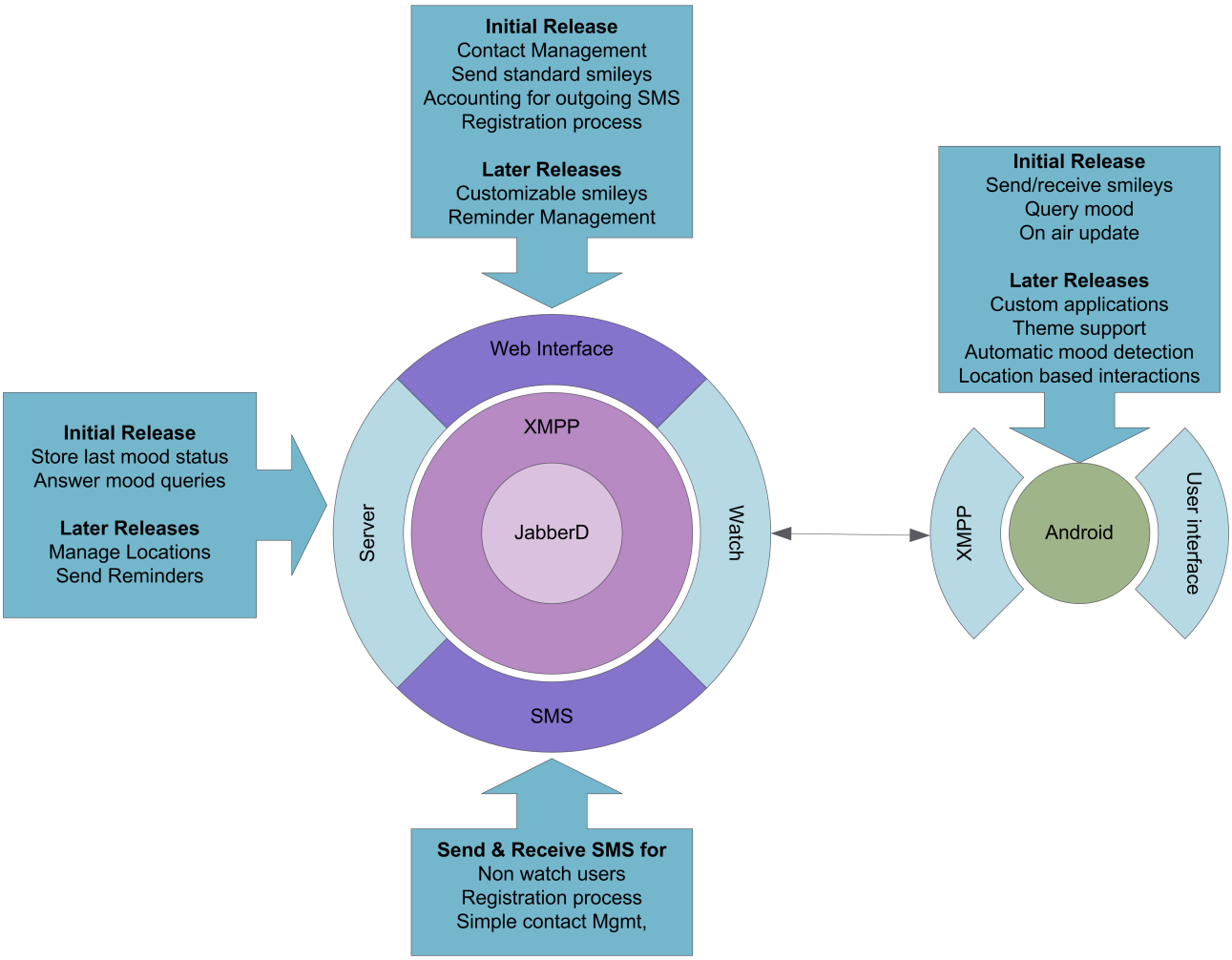
Requirements for the watch

For our watch we need to have a very solid and extensible platform. We have chosen Googles Android platform because it is free, open and supported by an alliance of more than 30 companies. It also has built-in XMPP and advanced graphics support. We will focus on the Android platform in the next chapter.

Our system architecture

With our choices to use the JabberD and the Android platform we saved a significant amount of efforts to develop these main components. With the XMPP we also saved the effort of creating a protocol for communication between the watch and the server. The only components that we need to develop are:

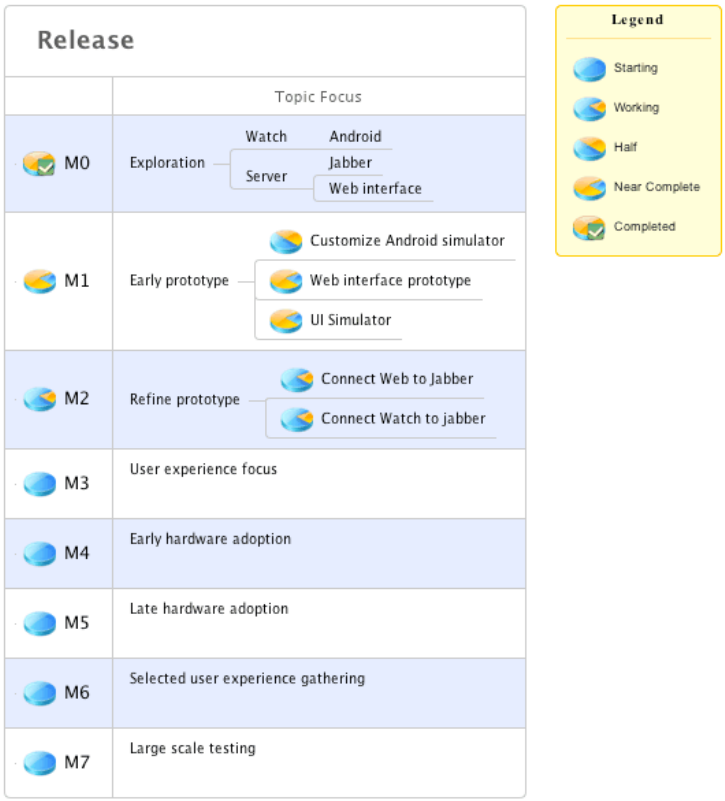
- Web interface/Web API
- Server user
- SMS Gateway
- Android User Interface
- (Sensor integration)



System overview

Roadmap

Our roadmap consists of 8 milestones.



M0 Exploration of concepts

In this phase we explored the possibilities that allow us to achieve our goals. This phase is completed and the chosen components have been presented in the pages before. We assigned the different components to our developers. Agusti is responsible for the Android platform. Farhad is responsible for the Web interface. Karsten is responsible for the UI Simulator and the Jabber part.

M1 Early prototypes

This phase focuses on creating simple prototypes that demonstrate that the main components are working. The UI Simulator is an exploration of the UI challenges. Due to the small size of the screen regular UI methods cannot be used and alternatives have to be found. This has a dramatic impact on the chosen hardware. Also hardware requirements were gathered during that phase. This phase is nearly completed and the results can be found in later pages.

M2 Refine prototype

In this phase we connect the prototypes to the server and check if they are working. For the android platform it has to be evaluated what components can be used and were own code has to be written. We then further develop the prototypes into a real code base and expand their functionality to what we claim to be in the initial release.

M3 User experience focus

In this phase we evaluate how well we reached our goals in terms of ease of use. This is done by deploying the UI simulator as software to our colleagues computers which are free to use the during the day. The UI simulator is used for exploring UI concepts while successful concepts are merged into the Android platform. We also present the UI simulator and Web IF to a very restricted number of people to get comments.

M4 Early hardware adoption

The assumption is that the hardware will still not be available at this point of time. So we start testing the software on another hardware platform that supports the Android platform. This way we can use it in a very mobile environment and see how well it works together with real hardware, and we can start testing means of energy saving.

M5 Late hardware adoption

Here we hope to have the preliminary hardware platform for the watch. So that we can start adopting Android to that platform. This especially focuses on preserving energy.

M6 Selected user experience gathering

In this phase we hope to have a running prototype that we can hand out to some key users and get even more comments. These comments will be incorporated into the product or a future model.

M7 Large scale testing

Our systems will be tested with a large amount of users to see how well it scales and if any quality problems have to be expected when the number of users growth.

Android Platform

Android Overview

Android is a software stack platform developed by Google which includes an operating system, middleware and key applications, according to the definition given by its creators [1].

Some of the more interesting features of Android which can be useful for our product are:

- Application framework which enables reuse and replacement of components.
- Dalvik virtual machine optimized for mobile devices.
- Optimized 2D graphics.
- Media support for common types of image formats.
- GSM telephony support.

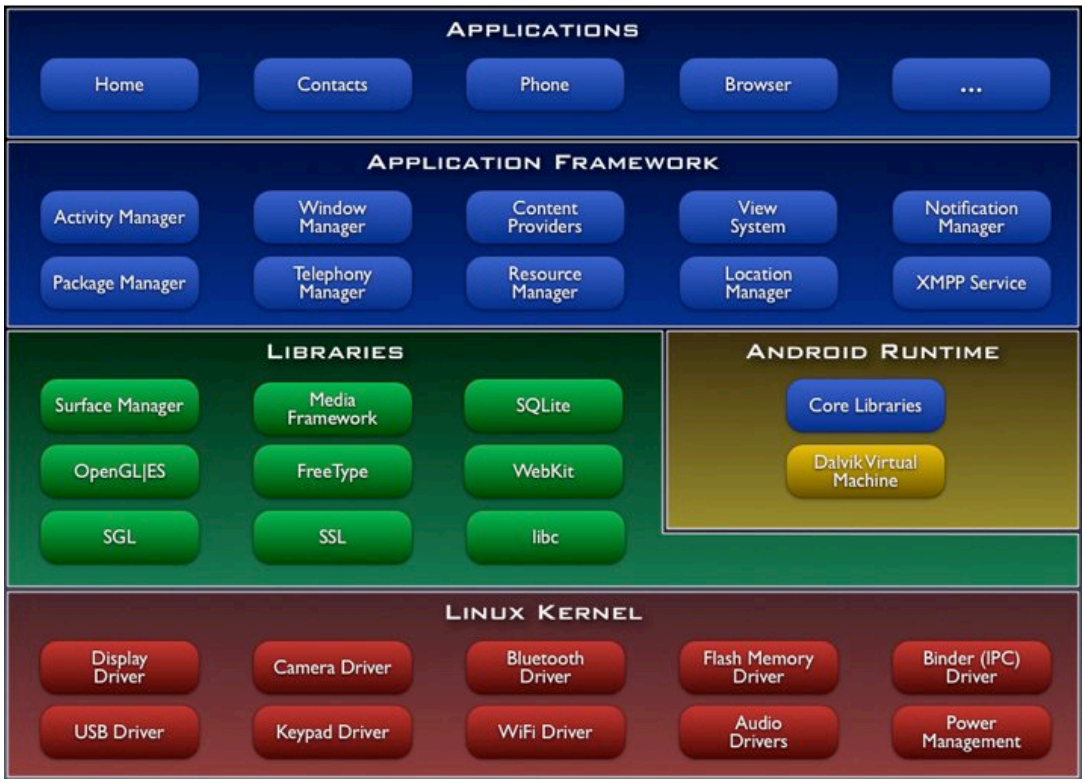


Figure 1. Android architecture.

Architecture

- Figure 1 shows us a diagram of the Android architecture. Android lies down on a 2.6 Linux kernel with drivers for different interfaces that are commonly included in mobile devices, in our case though, we do not require to use all of them.
- There is also a set of C/C++ libraries, common in UNIX systems that are adapted to fit the requirements of the devices where Android is supposed to be used. These libraries are available to the user through the application framework.

- Androids runtime libraries are a set of core libraries which implement almost all the functionalities available in the core libraries of the Java programming language. Java is the language used to develop applications on Android and because of that the implementation of these libraries is vital. Each of the Java applications running on Android runs on a Dalvik Virtual Machine (VM). The Dalvik VM is a register-based VM optimized for low memory requirements and designed to allow multiple VM instances to run at the same time, it was developed specifically for the Android project.
- Finally Android ships with a set of core applications, written in Java, made using the same Application Programming Interface (API) available for the developers.

Reasons for choosing Android

FaceTime has chosen to use the Android platform for a number of reasons. The following sections outlines them.

Big company support

Android is developed by the Open Handset Alliance (OHA) [2]. An alliance formed by important companies in the telecommunications area like Google, Intel, Motorola, NVIDIA, etc. This ensures that the Android platform will be supported throughout the lifetime of the products of our products and beyond.

Open and free

Android is open source and free for everyone. This gives us the advantage of being able, not only to implement our own applications without having to pay any license fees for the underlying platform but also to modify the underlying platform itself to adapt it to our product.

Very adoptable

Refer to the previous section to know why we consider Android as a very adaptable platform.

Good support for graphics

Android manages 2D graphics using a Scene Graph Library (SGL). We do not need to go that deep, using the application framework included with the Software Development Kit (SDK) we can directly use images in the same way they are used in common Java applications.

Stable platform

Android is a finished product which offers us some stability. That means we do not have to worry too much, about the development platform. Also the fact that it is open source helps. There is a big community behind it and the code is revised by much more people than the code of a closed platform, so the errors should, in theory, arise earlier.

Hardware for testing available soon

There are already some prototypes our there [3] and the first commercial products are conceived to get into the market next year [4]. So this means that we should be able to test our Android applications on their final hardware quite soon.

How does Android work?

In Android each application is based on activities. Each activity corresponds to one screen in the application. An activity is made of views. Inside each view you have your own layout with your own components on it.

The set of core applications that come with Android are managed by a home activity which allows us to select between the set of applications. This home activity is always running on our system.

Of course there are some cases where the direct mapping of application to a set of activities, i.e. set of screens, will not work. A media player for example should keep running even if we are using some other application. As in the planned versions of our product we do not need applications running in the background thus we will not go into details of how this is done. In case we would need that we can always modify our home application to support it.

Our use of Android

In the first version of our product we will only have one running application. This application is formed by at least two different activities, one to select the emoticon we would like to send and another to select the contact to who we want to send it to.

In order to do this we should also modify the set of initial core applications that come with Android. We would like to take them away except for the home activity. We should modify the home activity to directly start our unique activity after startup.

In future versions, it might be possible to have personalized applications. To run those we would need to modify the home activity again so that it would allow us to select which application we would like to run at each moment.

Implied hardware

In our product we are using four important hardware components which have to be compatible with Android. That means that the customized linux kernel should support them and that we have access to them through the application framework. We will discuss about these components in the following subsections.

Processor

The minimum processor required to run Android is a 200 Mhz processor according to [4]. We have read in not totally reliable sources that the processor has to be compatible with the ARM v5 instruction set or above.

The planned Broadcom processor fulfills this requirement by using ARMv11.

Memory

The minimum amount of memory necessary to run Android is 32 megabytes according to [4].

Flash

The minimum amount of flash memory necessary to run Android is 32 megabytes according to [4].

Our product will have 256 megabytes of flash memory.

Touch screen

We could not find any information about the supported types of touch screens supported by Android. However in the class index we could find some classes which imply the support of touch screens by Android. For example in the description of the MotionEvent class it states:*"Object used to report movement (mouse, pen, finger) events".* So we can see that they take into account pen and finger events which implies a touch screen available. Also one of the companies which is part of the OHA is Synaptics, Inc. a company which develops touch screen interfaces.

We are not assuming that only touch screen devices made by Synaptics, Inc. will be compatible with Android. Instead, we are assuming that any device which meets the minimum requirements of the device driver will work.

Web interface

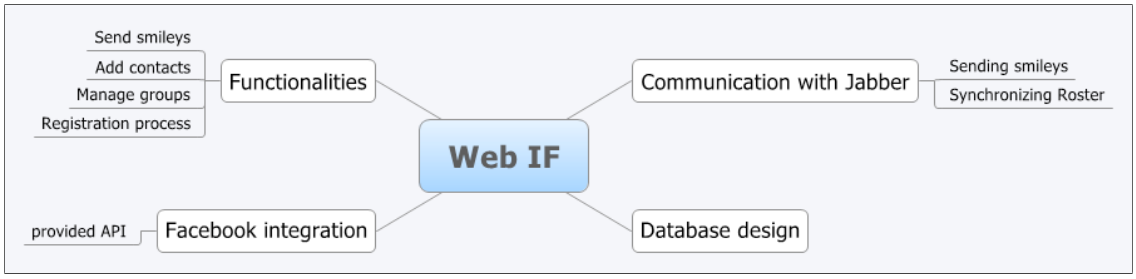
Overview

We decided to equip our Face Time watch whit a web interface because of the the reasons explained in the following subsections.

Registration

The first time the user wants to use the watch, he or she has to go through the registration process using the web interface. This registration process is done using watch's MAC address, the user can use its own jabber account to sign up. In case the MAC address is valid then a response number will be generated and will be used for the authentication of the watch.

After the user inputs the response number to prove he is the owner of the watch the web interface server will connect to the jabber database and check the availability of the username and password. After the availability has been checked the web interface will check if the watch has stored in its internal memory the same number it has generated, this is done in order to check if the watch can be accepted as a client.



Communication with others

The web interface allows the users to communicate between them using the following characteristics:

1. Retrieve friend list of user from jabber database.
2. Update them (add ,delete ,modify).
3. Send and receive instant messages.
4. Send and receive emoticons.

Design of the web interface

Requirements of the web interface server

In order to properly run our web interface server it will need to support the technologies shown in Table 1.

Functionalities

Through the web interface users can easily interact together like other social network website (ex: FaceBook ,Orkut, Myspace). The functionalities available for the users are:

Facetime

- In this environment any user has a **profile** which can be managed adding images or some text, also his/her friends will be visible in the profile and a small image of them is also available.
- They can easily **search** through the web interface server database and find their friends. After finding the desired person he/she can be easily added in the friends list of the user and they can start to interact with each other.
- Anyone can create a **community (group)** and manage it. Other users can join created communities and post or read messages on them.
- Of course, sending and receiving manages to and from users using watches or using the web interface will be also possible. In case the user is using the web interface the user will have some predefined emoticons available.

In future versions we would like to expand the last functionality allowing the user to create his/her own emoticons and sending them to his/her friends.

Database Design

For the log in phase we connect to the jabber.org database as we are using jabber accounts to register the users.

Our is accessed for the rest of functionalities.

UI Simulator

Purpose

The UI simulator was created for 2 reasons. The first one is that the screen of the watch is very constrained in size. Usual concepts such as buttons, scrollbars and menus do not work with that screen size. Therefor a new specialized UI has to be created. In order to start with developing these UI concepts even before any prototype of the hardware is available the simulator was created. This ensures to have a satisfying UI concept once the hardware is available.

Also the UI simulator can be deployed to a regular PC and can be used for communication in our team. This is useful to evaluate if our expectations of our overall concept have been met. Because most PC do not have a touch screen as our target hardware will have, it was necessary to emulate it. This was done using the mouse. The UI Simulator is fully connected to the server and can thus test the use of the XMPP.

UI Features

During the development of the UI simulator the following solutions were found. Still they do not yet represent the final interface.

Navigation

Because of the lack of screen estate special ways of input had to be found. The solution was to use 5 different gestures.

- Tap - a very short contact with the screen as in a click
- Roll up - moving the finger in a vertical direction upwards
- Roll down - moving the finger in a vertical direction downwards
- Slide right - moving the finger in a horizontal direction to the right
- Slide left - moving the finger in a horizontal direction to the left

While the slide left functionality in most cases will return to the previous screen, the other 4 functionalities can have different meanings depending on the screen contents.

Communicate the expected behavior

In order to let the user know what will happen when he execute a certain gesture, an area dedicated to instruct the user has been introduced. The lower 16 pixels are used to explain in plain English what will happen.

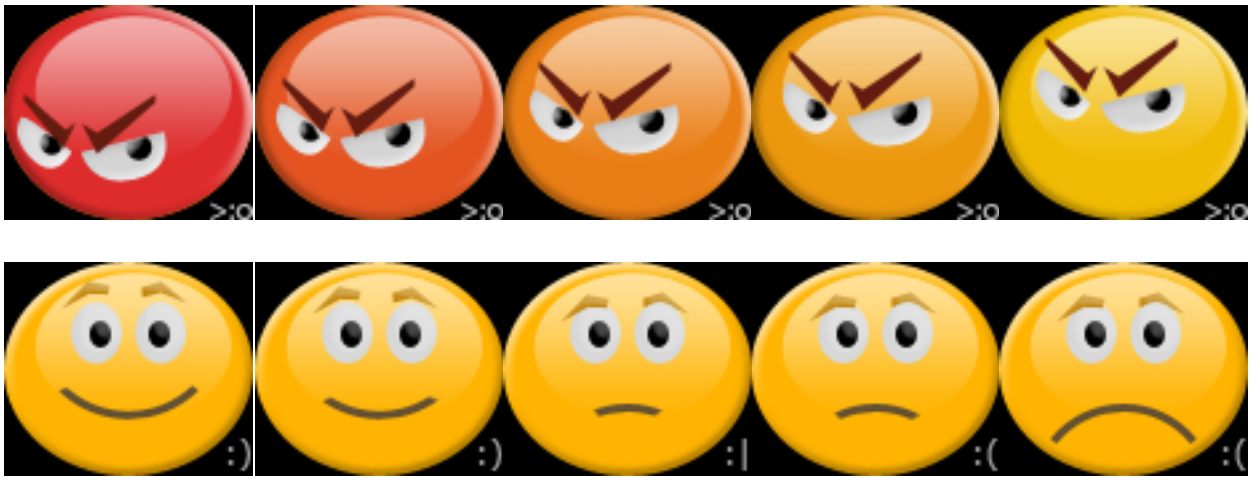
For example: “Slide to send” or “Slide to return”. While those are easy to understand they do not provide enough information to the user to know what exactly he has to do. The reason is that there are to slide gestures. One to the right and one to the left. The user gets a very intuitive hint:

The instruction “slide to send” is animated. A highlight is moving across the text in the direction to indicate what the user has to do. In this case he has to slide to the right. The colors were chosen to be very soft and the animation is very smooth so that the user does not get distracted too much.

The same technique is used for the vertical roll. The tap is represented using very short black period in a longer period of regular visibility.

Expressing the amplitude of your mood

Because mood is not a digital feature of our mind several grades of one mood can occur. A user can be a bit angry, angry, even more angry, or very angry. Just to name a few. In order to express those intermediate steps the user can modify an emoticon using the roll gesture. This way he can express 90 stages of angriness or happiness.



Communicating with non watch users

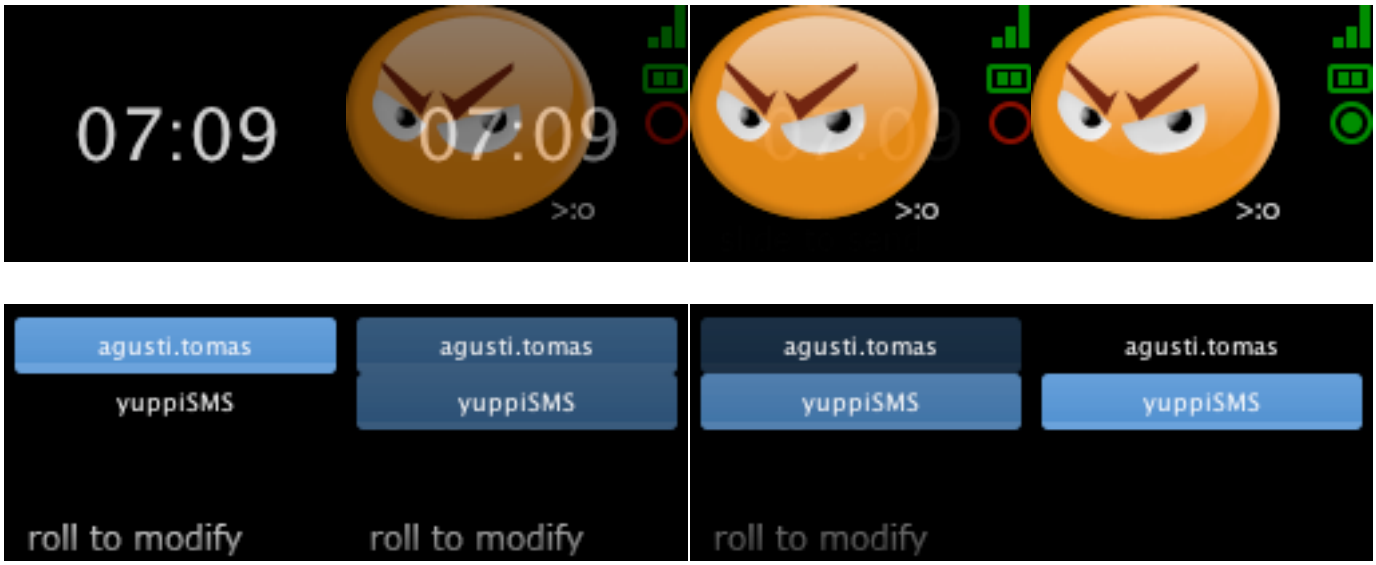
All of those granularities of the emoticon can be transported to another watch user. But what will happen when the user will send that emoticon to a non watch user? The answer is always visible in the lower right corner of that emoticon. This way the user knows what the emoticon will be mapped to. For angriness there is only one expression. For emoticons from happy to sad there are 3.

Using the most of the screen estate

Because of the small screen size every piece of screen has to be used carefully. That is why the emoticons are not round but elliptical.

Make a nice appealing

When the user is changing the screens, for example from the time screen to the mood selection screen, a hard change between those frames appears very rough. However our watch should be smooth to use. That is why transitions are faded.



It can also been seen that the selection indicator has round edges and features a very gentle gradient. This makes the selection appear very sophisticated and rich.

Indicators

In the emoticon selection screen several indicators are shown to give the user an idea of the current situation. The first bars are the network connection. The second one the battery life and the third one is the jabber connection.

The jabber connection indicator uses two features to indicate its state. The first one is that if not connected the circle is red. Red is usually associated with a malfunction. The second feature is the filling. If the connection is ready the circle is filled. This is known to the user from LEDs to indicate a status. If the circle is filled everything is okay and the emoticon can be sent.

References

[1] What is Android? Overview of the Android platform. Available at: http://code.google.com/android/what-is-android.html

[2] Open Handset Alliance homepage. OHA is the company behind the Android platform. Available at: http://www.openhandsetalliance.com/

[3] Willcom Android prototype page. Information about the development board made by Willcom running Android. Available at: http://google.com/translate?u=http%3A%2F%2Fk-tai.impress.co.jp%2Fcda%2Farticle%2Fnews_toppage%2F37548.html&langpair=ja%7Cen&hl=en&ie=UTF8

[4] Interview with Andy Rubin, director of mobile platforms at Google. Available at: http://www.news.com/2008-1039_3-6218126.html

[5] The ARM instruction set architecture. Available at: http://www.arm.com/products/CPUs/architecture.html