Faculty of Economic Sciences, Communication and IT
Department of Computer Science

Victor Ulhagen

Data visualization on Android

Degree Project of 15 credit points
Computer Science

Date/Term:	2011-06-09
Supervisor:	Katarina Asplund
Examiner:	Donald F. Ross
Serial Number:	C2011:03

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Victor Ulhagen

Approved, 2011-06-09

Advisor: Katarina Asplund

Examiner: Donald F. Ross

Abstract

The project discussed in this thesis was initiated by the Swedish IT company Ninetech in an effort to simplify maintenance of the company's server farm. The proposed method of simplifying maintenance was to create an Android application capable of visualizing key elements of the server farm. Since the internal information that the application should expose is of a classified nature, the whole system needs to be secured to prevent unauthorized access. This thesis describes the development of an Android application as well as a server application handling the communication between the Android application and the server farm information service.

Aknowledgements

Katarina Asplund

For helping me greatly with my dissertation.

Henrik Bäck

For explaining and showing me magic in C#, and for providing LaTeX tools and templates.

Ninetech

For being a generally nice place to be :)

Contents

List of Figures

Chapter 1
Introduction

Ninetech is a company that has a broad spectrum of customers which uses its services. To ensure the future integrity and stability of these services the time it takes to detect problems must be as short as possible. To shorten problem detection time a Android surveillance application was proposed.

1.1 Goal

Figure 1.1: System setup

The goal of the project was to create a system which can report and visualize real time data from a server monitor service in Android smart phones (see Figure 1.1). The Android smart phones should be able to connect to the server monitor service from outside the internal network. Therefore, the connections made must be encrypted to secure the sensitive data sent by the service. It was also important to make sure that the internal network would

stay secure. Therefore a proxy service should be added as a negotiator between the monitoring system and the Android smart phones.

1.2 Requirements

The project had a number of basic requirements that the application needed to fulfill. These requirements are described below.

1.2.1 Security

Since the information that is sent between the surveillance service and the smart phone may be sensitive the connection has to be secure. In section 2.2 we discuss which protocol that was chosen and how it works.
The service also have to be password protected to deny unauthorized users the information.

1.2.2 Data types

The Android application should be able to visualize two types of information, outlined below:

Messages

Messages contain information about problems in the server farm, as well as where and what the problem is. A message could for example indicate that an application crashed or warn about high load.

Measurement data

Measurement data is real time values of different measurable quantities. For example a group of values could be the amount of connected users to a certain service, or the transfer rate through a firewall.

1.2.3 Usability

Since the application will be used to ease the process of detecting errors and system diagnosis it needs to be simpler and faster than the existing error pipeline. To aid the user a set of convince features has been added to the application, these features will be discussed in chapter 3.

1.3 Results

In this Section the final results of the project are briefly discussed.

1.3.1 Proxy service

To ensure the security of the system a server between the server monitor system and the Android phone was created. This service is called the Proxy. The only responsibility of the Proxy service is to act as a negotiator, forwarding requests and responses between the server monitoring service and the Android phone. This will increase the security of the system by hiding most of the functionality of the server monitor service and only exposing the relevant parts through the Proxy. The Proxy also allows the server monitoring service to disregard encryption, since its not exposed outside the internal company network.

1.3.2 Android application

An Android application is able to post requests to the Proxy service and display the responses. Since there are two types of information available the application visualizes these differently. Messages are displayed in a list, allowing easy navigation. While measurement data is viewed as a graph.

1.4 Chapter overview

In chapter 2 we discuss the systems which the Android application and the Proxy service uses. Following, in chapter 3 we give a more technical overview of how the Android application and Proxy service is implemented. Finally, in chapter 4 we discuss the results of the project and the various problems that arose during development.

Chapter 2
Background

In this chapter we will outline underlying systems used by the different parts of the project. In Section 2.1 we give a brief introduction to the Android operating system. we also discuss the chosen encryption method in Section 2.2, the data exchange format in Section 2.3 and the user authentication protocol in Section 2.4. In Section 2.5 we give a brief background on the Windows Communication Foundation and finally in Section 2.6 we briefly discusses what a REST server is and how it is used in this project.

2.1 Android OS

Figure 2.1: Android logo

Android (see Figure 2.1) is a Linux based operating system that last year had 23% (2010) [12] of the market share, and this share is expected to increase to 38% by the end of this year. The operating system was initially developed by Android Inc, but was later bought by Google [4], which is the current owner.

Android OS is licensed as an open source project, and can be used according to the Apache license [3][2]

2.1.1 Limitations

There is a big difference between the performance of a smart phone and consumer computers. The CPU speed for a popular smart phone which was used when developing the application is a single 1GHz core[6], whereas laptop or stationary computer CPUs commonly sports two or four 3.5 Ghz cores[7]. Memorywise there is a big difference as well. The targeted smart phone has 576 MB[6] of internal memory versus the 2-8 GB internal memory for a home computer. These limitations forces the Android OS to handle applications differently from computers, as described in the next section.

2.1.2 Application lifetime

Android has a different way of handling applications compared to ordinary computer operating systems (eg. Windows , Linux and MacOS). When most operating systems let applications run until they are finished, Android has two application types suited for different tasks[5]. Standard applications will use the "Activity" type. This application type can be terminated by Android at any moment when it is not focused by the user, in order to conserve memory. The other type is called "Service". This application type runs until it is finished and it will not be stopped. This service type should be used by processes that run in the background without direct user interaction.

2.2 Secure Socket Layer (SSL)

The Android application contacts a Proxy service (see Figure 1.1) for the information that should be displayed for the user. This data could potentially be classified for the public, in which case the connection needs to be encrypted. For this project we chose to use the SSL encryption protocol. SSL was chosen since its widely used (eg. Https) and objects for managing SSL connections is available in the Android OS as well as in WCF (see Section 2.5). SSL provides a secure connection over the standard socket interface. SSL is an asymmetric encryption protocol. This means that it uses a combination of public and private keys for encryption and decryption (see Figure 2.2). Encryption works by encrypting a message with the recipients public key. When the message is received the recipient then decrypts the message with its private key. SSL can use a multitude of encryption algorithms such as DES and AES[8]. The algorithm to use is determined in the "Negotiation phase" of the connection (see next section). Certificates are also used to assure that the host has the right identity. This is done to secure the connection against Man-In-The-Middle attacks. In essence a Man-In-The-Middle attack is executed by fooling the connecting client that the attacker is the service that the client wants, and then using the information given by the client to gain access to the real service. SSL is widely used since it provides a standardized way of determining encryption algorithms and makes sure that you are in fact talking to the correct recipient The largest field of usage is Internet applications, where HTTPS connections basically uses SSL for its secured communication.

Figure 2.2: SSL connection setup

2.2.1 SSL Connection setup

When an SSL connection is setup the host and the client need to settle on what type of encryption and which version of SSL they should use. This phase is called the "Negotiation phase". During the negotiation phase several facts are established between the client and the server:

* Public keys to use for encryption
* Which cipher to use
* Which compression algorithm to use
* Server (and optionally the client) identity
* Public keys to use for encryption

When the negotiation phase is complete the encrypted communication can commence.

2.3 Information containment format

Since the Smart phone has limited processing ability the data sent between the service and the phone needs to be tailored to the phone (see Section 2.1.1). XML is commonly used as the information containment format (its

most prominent use is for the web as XHTML). XML is very flexible but it contains a lot of redundant data which makes it too slow to consume for a smart phone. As an alternative to XML there is a format called JSON which can be used. JSON is described in the next section

2.3.1 JavaScript Object Notation (JSON)

JSON [10] is a human-readable data exchange format suited for simple data structures. As its name suggests it is derived from the Java Script programming language. Despite this fact, JSON is a language-independent format. Compared to XML, JSON is not hugely different. JSON is a bit less flexible, as an example a JSON node cannot have attributes and children at the same time. However, this is easy to work around.

2.3.2 JSON Example

The figure 2.3 below shows a simple JSON example describing pets.

Ignoring whitespace characters this example takes 100 characters to encode while still being understandable. Figure 2.4 shows the same information encoded with XML.

Again, if we ignore the whitespace characters, we end up using 142 characters. By sacrificing a bit of flexibility we gain a compression rate of around 70% with JSON without using costly compression algorithms. This property makes JSON ideal for low performance targets, such as smart phones

```
{
animals:
     {
          dog:
          [
               {
                    name:'Rufus',
                    breed:'labrador'
               },
               {
                    name:'Marty',
                    breed:'whippet'
               }
          ],
          cat:
          {
               name:'Matilda'
          }
     }
}
```

Figure 2.3: JSON Example

```
<animals>
     <dog>
          <name>Rufus</name>
          <breed>labrador</breed>
     </dog>
     <dog>
          <name>Marty</name>
          <breed>whippet</breed>
     </dog>
     <cat name="Matilda"/>
</animals>
```

Figure 2.4: XML Example

2.4 Authentication

Authentication of user credentials is handled by the HTTP Basic authorization protocol[13]. HTTP Basic authentication works by appending the user credentials onto a standard HTTP request. Before transmitting, the credentials is concatenated together with a separating colon. The resulting string is encrypted using Base64 encryption, and after encryption the resulting string is attached to the request. For example, given the user name 'Aladdin' and the password 'open sesame', the string 'Aladdin:open sesame' is Base64 encoded, resulting in 'QWxhZGRpbjpvcGVuIHNlc2FtZQ=='[9].

Base64 encoding only disables humans from reading it, against computers it does not provide any security. Base64 encryption is instead used to make sure that no illegal characters make its way into the final URL request[9]. User credentials are still secure however, since the connection this project uses is encrypted using SSL (see section 2.2).

2.5 Windows Communication Foundation

Windows Communication Foundation (WCF) [11] is an application programming interface provided by Microsoft for building service oriented applications. WCF is used through the Microsoft .Net platform. WCF is designed to support easy service consumption by service consumers, this means that a client can consume a multitude of services and a service can be consumed by a multitude of clients. This consumption behavior is ideal for web based services, which caters to a multitude of clients simultaneously.

2.6 Representational State Transfer (REST)

A Representational State Transfer Server [1] is a software architectural style suited for distributed media systems. For a server to be RESTful it needs to abide to a set of constraints, described below:

Client-server model

Clients and servers are separated; servers are not concerned with user interfaces, while clients does not care about the internal state of the server.

Stateless

The server should not store client contextual data, but should instead only react to client stimuli. The statelessness forces clients to send all the required data with each request, making the communication more robust and scalable.

Cacheable data

All data sent from the server should be cacheable, or the client should be able to determine when data received from a server has gotten stale. This constraint enables the clients to reuse data, skipping some communication all together.

Layered structure

The client should not know about the internal structure of the server. This constraint enables a service to be distributed to a multitude of internal servers increasing scalability. It also enables intermediate services between the client and the end service.

Uniform interface

Keeping a uniform interface between the server and the client enables a strong decoupling of the two systems. This constraint enables the client and the server to grow independent from each other.

2.6.1 Connecting to service

Connections to a WCF REST service is simply done using the HTTP or the HTTPS protocol. Each function in the service is mapped to a URL address of the same name. As an example we may have a service named 'students', which is residing on 'kau.se'. In the 'students' service we have a function named 'getStudents'. Calling the 'getStudents' function is simply done by requesting the page 'kau.se/students/getStudents', which in turn would return a XML (or JSON, see section 2.3.1) encoded response.

Chapter 3
System implementation

This chapter describes the actual implementation of the project, and gives a more technical insight into the project. In section 3.1 we give a brief introduction to the Android API and the objects that are most used in it. In section 3.2 we will take a glance at the components that builds the system. Following in section 3.3 we discuss the Proxy service, and in section 3.4 we will look at the Android application.

3.1 Android API

The Android API is a supplement to the standard Java runtime library, including classes that relates to the operating system of the phone and for convenience. Since the Android API is quite vast the application uses only a small subset of its functionality. Following is an outline of the classes that the application uses the most.

Activity

Every application that has a graphical user interface needs at least one class that extends the Activity class. The Activity class handles loading and rendering of user interfaces. A good feature which the project uses extensively is the ability to load user interfaces specified in a XML file. By loading the

interface from an XML file time taken to do the actual design is minimized and it makes it easy to change the interface without changing the underlying functionality.

IntentService

The 'IntentService' class has a similar function as the Activity class, but is used for services instead of applications. Activities can communicate with 'IntentServices' in an asynchronous fashion to offload work from the application onto the service. Being asynchronous means that when the call to the IntentService is made the caller can continue with its business as usual, and later on get a call with the result from the service. By offloading work to services in a non blocking way applications can focus on keeping its interface responsive while the service does all the work.

Intent

The Intent class is used for a multitude of tasks in the Android API. All communication to the phone or between Activities are handled by Intents. It can be used with 'startActivity' to launch an Activity, with 'broadcastIntent' to send it to any interested 'BroadcastReceiver' components, and with 'startService' to communicate with a background Service.

SharedPreferences

The SharedPreferences class is used to broadcast settings throughout an application. Every Activity within the application can read and modify the SharedPreferences object. SharedPreferences is also saved between uses, adding an easy way to store user settings.

3.2 System overview

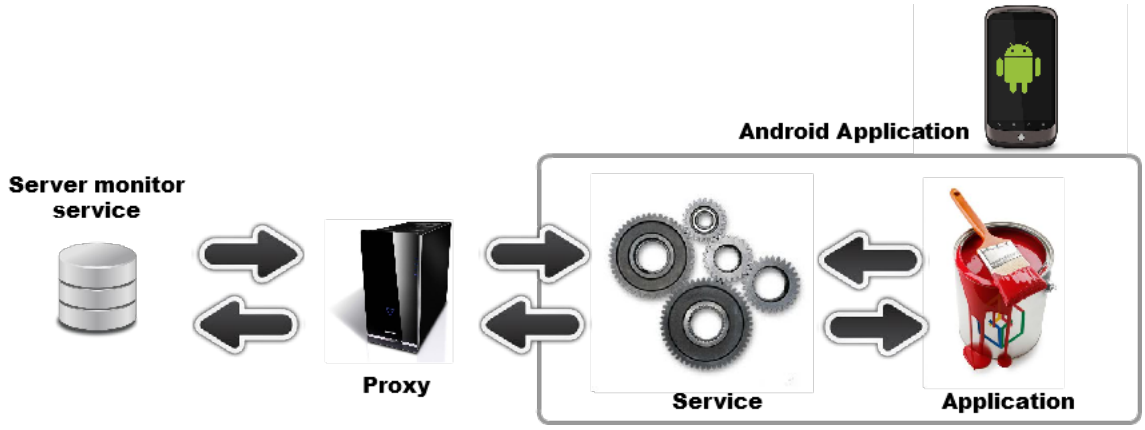

Figure 3.1: Overview of all individual parts of the system

The system consists of two major parts. The Proxy service which handles communication between the server monitor service and the smart phone. The Android application which can be subdivided into two smaller parts, The Service which handles all communication and message caching between the smart phone and the Proxy, and the Application which enables the user to interact and view the information provided by the server monitor service. The connections between the parts can be viewed in figure 3.1. The Proxy service is described in section 3.3 and the Android application in section 3.4.

3.3 Proxy service

The Proxy service is implemented using C# and the WCF.net service architecture. It is residing as a negotiator between the Android application and the server monitor service, as can be seen in figure 3.2. This Proxy service is simply a small set of functions that can be called from the outside. These functions are described in the following sections:

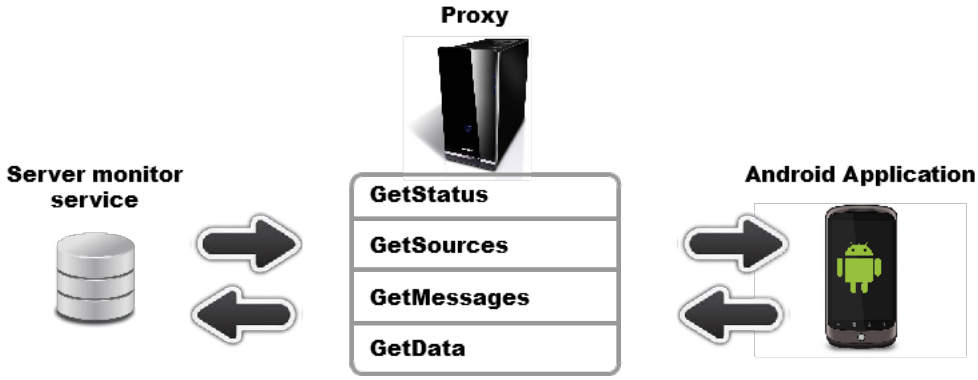

Figure 3.2: Detailed overview of the Proxy service

3.3.1 GetStatus

The GetStatus is used as an initiation function by the Android application, checking if the provided service address is valid. The function does not have any input parameters, and it returns the version of the service as well as the local server time.

3.3.2 GetSources

The GetSources function is used to construct the menu inside the Android application. The function has no input parameters and it returns a list of data sources, where a source is a combination of provider id and type of data.

3.3.3 GetMessages

The GetMessages function is used to collect error messages. The function has source id as input parameters and it returns a list of messages.

3.3.4 GetData

The GetData function is used to collect visualization data. The function has source id and a number representing the targeted timespan as input parameters and it returns a list of data values.

3.4 Android application

Figure 3.3: Android application setup

The Android application is implemented as two parts to complement the application/service model Android employs (see section 2.1.2) as can be seen in figure 3.3. This design allows the system to alert the user even when the main application is not active. In the following sections we will describe the individual parts of the Android application in more detail. In Section 3.4.1 we talk about the application manifest. Following, in Section 3.4.2 we describe the Service. And finally we discuss the Application in Section 3.4.3.

3.4.1 Manifest

Coupled with the application and the service is a manifest. The manifest contains settings and information about the application and it is a integral part of any Android application. The manifest also contains a list of all activities that the application can start, the events which the application is interested in as well as permissions. Registering events in the manifest enables an application to, for example, start when the phone starts, or handle viewing of specific file types. The manifest can register that it wants to react to PDF intents, enabling other applications to just tell the phone that it wants to view a PDF. The phone can then check if there are any applications registered for the PDF event and start the application accordingly. Another example is that the service registers its interest in the LAUNCHER event to start when the phone starts. The LAUNCHER event is provided by the Android OS, and it is broadcast when the phone starts. Having the manifest specify which permissions the application wants adds to the security of the phone. For example, the manifest can have the permission for Internet access. When installing an application the user can review the permissions used and cancel the installation if the permissions seems invalid (for example a calculator application will not need Internet access, and would alert the user that the application may be malicious)

3.4.2 Service

The service part of the Android application starts when the phone is turned on and will not stop unless the user explicitly terminates it. Its main purpose is to periodically check the Proxy service for new error messages and display these to the user. This means that the user does not have to check the application

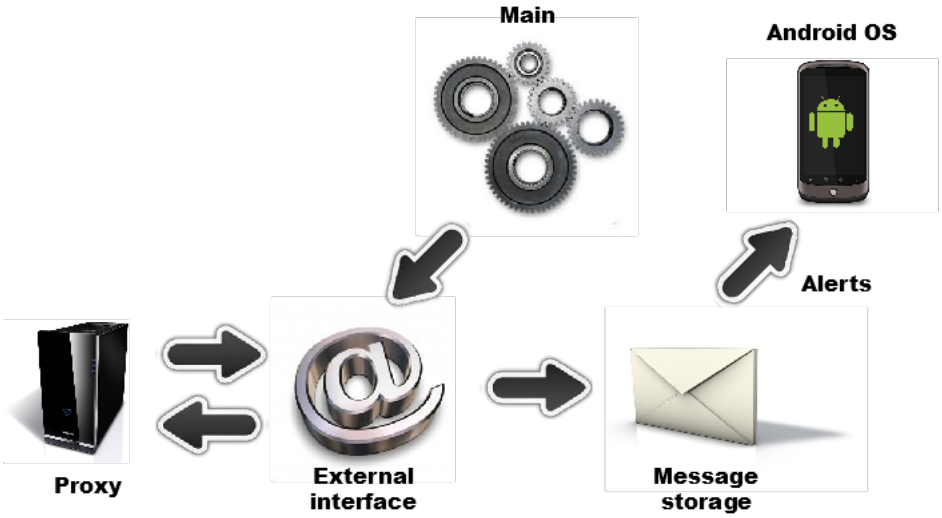

Figure 3.4: Android service setup

periodically, which adds to user friendliness.

The service is composed of three parts (see figure 3.4), which are described below:

3.4.2.1 Main

The main part of the service handles service startup and tear down, as well as initialization of the other two systems. It also acts as an entry point for the application, exposing the two other parts. Startup of the service can be done in two ways. The first option is that the service registers itself to receive an event when the phone is turned on. By doing so the user will get the functionality the service provides without having to start the application first. The application can also start the service if it is not started by the time the application launches. When the application wants to couple with the service an Intent (see section 3.1) is generated and then sent to the service (creating the service if it is not available). The service responds by sending an event to the application with a reference to itself. When the communication finishes

the application ends up with a usable reference to the service, and the service is locked to the application that requested the coupling. Finally when the connection is not needed the application drops the connection and the service is ready for a new coupling.

3.4.2.2 External Interface

The External Interface initializes and maintains the connection to the Proxy service. It also provides an abstraction layer between the phone and the Proxy, enabling easier Proxy handling. To create a valid connection a series of requests must be made. Firstly, the interface needs a URL path locating the Proxy service. When a path is given and validated to be an active Proxy, the interface then needs valid user credentials. When the Proxy and user is validated the connection is open, and other requests can now be sent. The available requests for the external interface are outlined below:

Connect

The Connect function takes a URL address and stores it for later use. This function is called first in the connection setup to specify which address the following calls should be directed.

Authorize

Calling Authorize makes the external interface try to establish a working connection to a Proxy.

Refresh Sources

By calling Refresh Sources the interface will update the list of available sources for later use. The sources are stored inside the interface so that the application

request sources only one time and then reuse the cached response.

Refresh Messages

When calling Refresh Messages the interface retrieves a list of the currently active messages from the Proxy. It also sorts the messages and compare them to the currently cached list of messages. If a new message is detected the caller is notified by the function response. When the check for new messages is complete the new list is sent to the Message Storage.

Get Data

Get Data needs a source id and the number of seconds back in time which the caller is interested in. In return Get Data will give the caller a list of data points and a value label that can be used to draw a graph.

3.4.2.3 Message Storage

Messages collected from the Proxy are stored in the Message Storage. Since the phone only can request a list of all active messages, this part needs to sort these messages and also detect when a new message is received so that an alert can be sent to the user.

3.4.3 Application

The main application is treated as a collection of smaller programs (or states) which henceforth will be called pages. Each page is its own Activity (see section 3.1) following the implementation guidelines for the Android. Using an Activity for each page allows automatic backwards navigation and resource handling, but it also introduces the risk of navigational infinite loops. Navi-

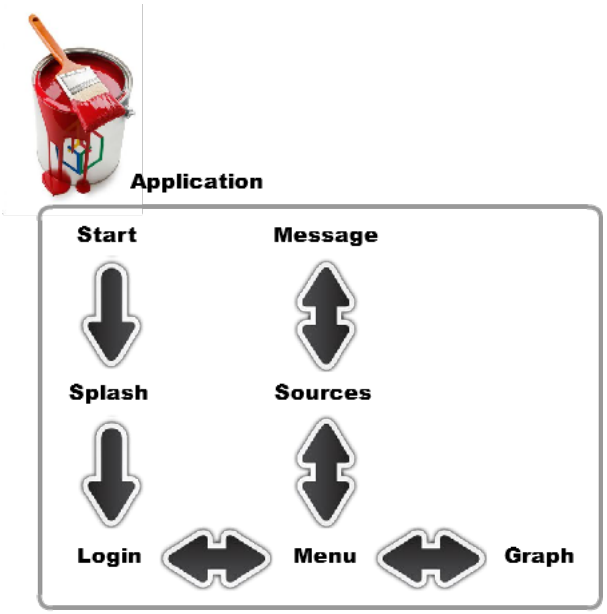

Figure 3.5: Android application overview

gational loops means that, if not careful, an application can enable the user to navigate in circles. By navigating in circles more and more pages will be allocated in memory, draining the phone of its resources. The problem with the navigational loops are solved by having a strict navigational scheme (see figure 3.5). Below follows a description of the pages that the application is composed of.

3.4.3.1 Splash

The Splash page (see Figure 3.6) acts as the startup page for the application. Its only responsibility is to display the application logo and after a few seconds send the user to the Login page.

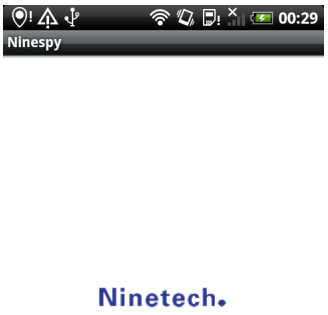

Figure 3.6: Application splash screen

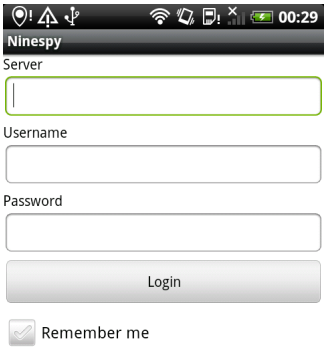

Figure 3.7: Application login screen

3.4.3.2 Login

The Login page (see Figure 3.7) enables the user to enter the credentials needed to connect to the Proxy service. It also stores this information if the user wishes, allowing for easier login on subsequent uses. The Login page is also responsible for testing the given credentials, only advancing to the menu page when the External Interface (see section 3.4.2.2) has a working connection. When the user has specified that the credentials should be saved, they are saved in a Shared Preferences object. Such an object enables all other parts of the application and the service to access this information and the credentials can be used to create a valid connection when the service starts. When the user has submitted credentials they are used to create a connection in the external interface. To setup the connection a call to Connect (see section 3.4.2.2) is made, testing if the supplied URL is valid. When a Proxy is validated the username and password is used in a call to Authorize (see section 3.4.2.2). If both these calls are successful a valid connection has been established and the application can proceed to the next page.

3.4.3.3 Sources

The Source page (see Figure 3.8) lists all available message sources, enabling navigation to message pages for each source.

3.4.3.4 Message

The Message page (see Figure 3.9) lists all messages tied to a single source. Each message is composed of a time stamp, an error level and the actual message text.

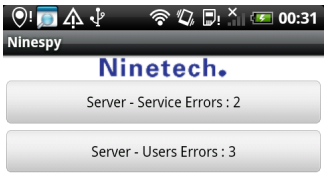

Figure 3.8: Application sources screen

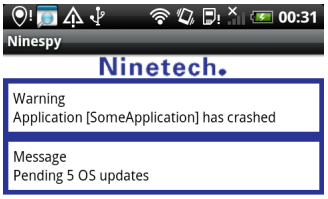

Figure 3.9: Application messages screen

3.4.3.5 Graph

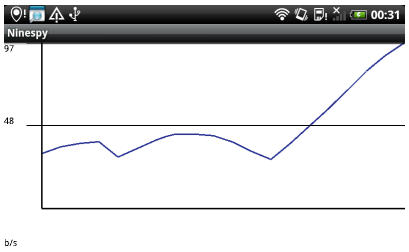

Figure 3.10: Application graph screen

The Graph page (see Figure 3.10) handles the visualization of a data source in the form of a graph. The graph is continually updated automatically, scaling the vertical axis depending on the currently largest data point. This page also features a user controlled "ruler". The ruler works by finding the data point nearest to where the user presses the screen (see Figure 3.11). This function enables the user to view the value of interesting data points.

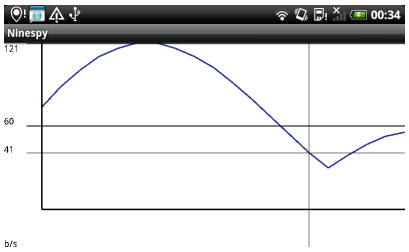

Figure 3.11: Application graph screen, with ruler

Chapter 4
Results and evaluation

In this chapter we present the final results of the project in section 4.1 we also discuss the various problems encountered while developing the Android application as well as the Proxy service in section 4.2.

4.1 Results

When the project finished we had created three systems, the Android application as well as two implementations of the Proxy service. To uncouple the development of the server information system and the Android application a Proxy simulator was created. The Proxy simulator is used to test connectivity and functionality of the Android application. The simulator works by acting like a Proxy but instead of gathering data from the server information service it generates it when requested. In the end of the project we also created a Proxy which connects to a server information service to generate data. The Android application was also finished. The finished application can communicate through an encrypted connection with a Proxy, and display the data received.

4.2 Problems

In this section we discuss the problems which was not foreseen to happen when implementing the system during planning.

4.2.1 Developing on windows

The Proxy service was developed using Microsoft Visual Studio 2010. The development service server shipped with Visual Studio does not accept connections from other sources than localhost. Since the phone is an external source with its own IP address it was denied access.
This problem was solved by using a port forwarding application, and in turn mask the external connection as a local connection.

4.2.2 Battery versus workload

Since the background service on the Android runs indefinite, making requests to the Proxy service whenever it can, it adds a constant processor load overhead to the phone. Since requesting messages and then sorting these messages takes a lot of processing the processor does not have time to rest. This would be an acceptable solution on a computer since the thread that handles message updating could be set to a low priority, maintaining the performance integrity of the system. Having the processor work all the time on a smart phone, on the other hand, is a very bad idea, because constantly updating drained the phone's battery in a couple of hours.
The problem was mitigated by adding a timer to the update routine, so updating would only happen every other minute.

4.2.3 Self signed certificates

As described in section 2.2, a SSL connection uses certificates to ensure the identity of the server. These certificates costs money to acquire unless you create a self signed certificate. A self signed certificate does not provide any security since anyone can forge a dubious copy. When developing it is not viable to pay for a real certificate, which lead to the use of a self signed certificate being used. The only problem with using a self signed certificate was that the implementation of SSL connections on Android by default terminates when it registers the self signed certificate.

This was solved by bluntly forcing the SSL implementation trust every certificate, signed or not. This effectively renders the SSL encryption useless until a valid certificate is placed, and this override is removed.

Chapter 5
Conclusion

There are still room for extensions and improvements to the application and Proxy, but the completed version should cover the basic needs of server farm surveillance. From a personal point of view it has been interesting to work with an application that hopefully will see practical use in the future. It has also been interesting and rewarding working with newer technology. Especially since this project has been centered around smart phone development, which in the coming years will be commonplace. In the end I am satisfied with the finished product, and hope that Ninetech will be too.

5.1 Future work

In this section we will discuss a few options for improving the system. Adding a system to change different properties of the Android application would help usability further. For example, changing update frequency or alert methods could make the application more convenient for the user. The graph is currently rendered in software mode. Changing the rendering to use OpenGL instead would drastically improve graph performance and visual quality of the graph.

References

[1] IBM Alex Rodriguez. Restful web services: The basics. 2011-06-12. Available from: https://www.ibm.com/developerworks/webservices/library/ws-restful/.

[2] Apache. Apache license, version 2.0. 2011-06-12. Available from: http://www.apache.org/licenses/LICENSE-2.0.

[3] Dave Bort. Android is now available as open source. 2011-06-12. Available from: https://sites.google.com/a/android.com/opensource/posts/opensource.

[4] Ben Elgin. Google buys android for its mobile arsenal. 2011-06-12. Available from: http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm.

[5] Google. Application fundamentals. 2011-06-12. Available from: http://developer.android.com/guide/topics/fundamentals.html.

[6] HTC. Htc desire. 2011-06-12. Available from: http://www.htc.com/www/product/desire/specification.html.

[7] Intel. Intel processor comparison. 2011-06-12. Available from: http://www.intel.com/consumer/products/processors/compare-processors.htm?select=desktop.

[8] Intel. Transport layer security (tls) parameters. 2011-06-12. Available from: http://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-3.

[9] S. Josefsson. The base16, base32, and base64 data encodings. 2011-06-12. Available from: http://tools.ietf.org/html/rfc4648.

[10] JSON. Introducing json. 2011-06-12. Available from: http://json.org/.

[11] Microsoft. Windows communication foundation is... . 2011-06-12. Available from: http://msdn.microsoft.com/en-us/netframework/aa663324.aspx.

[12] Don Reisinger. Gartner: Android market share to near 50 percent. 2011-06-12. Available from: http://news.cnet.com/8301-13506_3-20051610-17.html.

[13] UC Irvine H. Frystyk T. Berners-Lee, R. Fielding. Hypertext transfer protocol – http/1.0. 2011-06-12. Available from: http://www.ietf.org/rfc/rfc1945.txt.