NILS on Android
– The Wonderful Journey

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Andreas Carlsson
Anton Hjalmarsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2010

Abstract

NILS on Android – The Wonderful Journey

Android is the fastest growing operating system for mobile phones. The development of applications for Android is increasing exponentially, about 2000 new applications per week for the first quarter 2010. [17] And the reason for this trend is particularly Google's openness to everything. Android is open source, and in principle, anyone can freely make their own applications that can be published and used by the entire world. So it was no coincidence that it was just Android that Saab wanted to explore a little deeper to see if this could be something for Saab and their future products.

In this thesis Android was investigated to see if Android was as customizable as Saab wanted, and if the NDK could be modified in such a way that Saab was able to port one of their current products to Android.
The result was that Android does not currently support C + + exceptions which meant that there was no opportunity to continue on that track. Instead an application was developed, NILS (Network Injects Logistic Service), which is part of a project called Crisis Training. CT is a tool for civil training and is suitable for training disaster situations.

Keywords: Android, Crisis Training, NILS, Application, NDK, SDK

Sammanfattning

NILS på Android – Den Underbara Resan

Android är idag det operativsystem till mobiler som växer allra snabbast. Utvecklingen av applikationer till Android ökar lavinartat, ca 2000 nya applikationer/vecka under första kvartalet 2010. [17] Och anledningen till denna utveckling är framförallt Googles öppenhet med allt. Öppen källkod, samt så kan i princip vem som helst helt fritt göra sina egna applikationer som kan publiceras och användas av hela världen. Det var alltså ingen slump att det var just Android som Saab ville undersöka lite djupare för att se om detta kunde vara något för Saab och Saabs framtida produkter.

I detta examensarbete tittades det på om Android var så anpassningsbart som Saab önskade och om man kunde modifiera NDKn på ett sådant sätt så Saab kunde ”porta” en av deras nuvarande produkter till Android.
Resultatet blev att Android inte idag stödjer C++ exceptions vilket gjorde att det inte fanns möjlighet att fortsätta på det spåret. Därför blev det istället att det utvecklades en applikation, NILS (Network Injects Logistic Service), som är en del av ett projekt som går under namnet Crisis Training. CT är ett träningsverktyg för civilträning och lämpar sig för träning vid exempelvis katastrofsituationer.

Nyckelord: Android, CT, NILS, Applikation, NDK, SDK

Foreword

The last thing you do at the Bachelor of Science in Computer Engineering program at LTH is to make a thesis in a subject that connects to your education.
This thesis consists of this document and an application for Android called NILS which is a part of Saab's project CT.

Our tutor at Saab has been Mikael Eriksson and examiner at LTH has been Stefan Nyman.

We want to thank the people at the office in Helsingborg for their support and help during the work with our thesis, and a special thanks to:

Mikael Eriksson, Saab
Stefan Lundmark, Saab
Niklas Andersson, Saab
Stefan Nyman, LTH

List of contents

1 Background 2

2 Problem description 2
2.1 Goals 2
2.2 Delimitations 2

3 Method 2
3.1 Phase 1 3
3.2 Phase 2 3
3.3 Time plan 3

4 What is Android 3
4.1 Architecture 4
4.1.1 Application 4
4.1.2 Application framework 4
4.1.3 Libraries 5
4.1.4 Android runtime 5
4.1.5 Linux kernel 5
4.2 SDK – System Development Kit 5
4.2.1 Android Emulator 5
4.2.2 Android Development Tools plug in 6
4.2.3 DDMS 6
4.3 NDK – Native Development Kit 6
4.3.1 Limitations 6

5 Port native code 7
5.1 Using NDK 7
5.2 Modified NDK 7
5.3 Compile with source code 7
5.4 What will the future bring 8

6 CT - Application for Crisis Training 8
6.1 Background 8
6.1.1 CT Components 8
6.1.2 Exercise Roles 9

7 NILS – Network Injects Logistic Service 10
7.1 Software 10
7.1.1 How to get started with Eclipse 10
7.2 Activities 11
7.2.1 Activity 11
7.2.2 Activity Lifecycle 12
7.2.3 AndroidManifest 14
7.3 User Interface 15

7.3.1 Tab Layout 15
7.3.2 Linear Layout 15
7.3.3 Table Layout 15
7.4 XML 16
8 Web service 17
8.1 JavaScript Object Notation 17
8.2 Server 17
8.2.1 Requests 19
8.2.2 Updates 19
8.3 Client 19
8.3.1 Lists 20
8.3.2 Parsing JSON 20
8.3.3 Container 21
9 Conclusions 21
9.1 Result 21
9.2 Discussion 21
9.3 Future work 23
9.3.1 SQLite 23
9.3.2 Service 23
9.3.3 Notification 23
10 References 24
11 Dictionary 25
12 Appendix A 26

Introduction

We started with defining the project description. In the project description we defined the purpose and expected result of the project. Also, our time plan was described as a Gantt chart. There we specified how many weeks we would spend on each part of the project.

The first part was an analysis of Android and its tools. The main questions we focused on were, what potentials Android's SDK and NDK has and if it was possible to port some of Saab's products to Android.

The result from the analysis part helped us choosing orientation of the development part.

The second part was the development part where we separated the job between us. Andreas focused on the underlying processing of information in the application and the communication between the application and the web service and from there, down to the database. Anton focused on the applications user interface and the applications functions and functionality.

The final report was written continually during the entire project and we tried to keep the writing in phase with the progression of the project. The presentation of the thesis that was held at Campus Helsingborg was based on the content in final report.

1 Background

“Training and Simulation – Software Products” is a division within Saab Business Area Security and Defence Solutions who develop training systems for military and civil education. They work to create service based solutions through integration of system applications and components.

In 2010, Android will be introduced in a couple of mobile phones and mini pc’s. Saab is interested in evaluating how Google’s new platform, Android, can be used in applications for integration, training and civil security.

2 Problem description

Saab Training and Simulation, who is a leading developer of systems in military training, are interested to use Android for their products and want to know if it has the needed capacity for porting an existing product, or if not, develop a prototype of a new Saab product for Android. The goals are divided into a main and a secondary goal.

2.1 Goals

The main goal is to determine if Android got the capacity and abilities to run a Saab product in the sense that it will be of any use for Saab to go ahead with future projects that are oriented towards Android.

A secondary goal is to get a working prototype of a Saab product to execute on an Android platform.

2.2 Delimitations

This thesis is not supposed to deliver a product with commercial product quality.

If the analysis part shows that it is possible for a Saab product to run on the Android platform, the development part should focus on porting one of Saab’s products to Android.

3 Method

The work will be divided into two phases. The first phase will be an analysis phase. And the second phase will be a development phase.

3.1 Phase 1

During phase 1 an analysis of Android will be made. The result from this phase will determine the orientation of the next phase, the development phase.

3.2 Phase 2

This phase will be a development phase. Depending on the result in phase 1, a decision will be made on what kind of application that will be developed.

3.3 Time plan

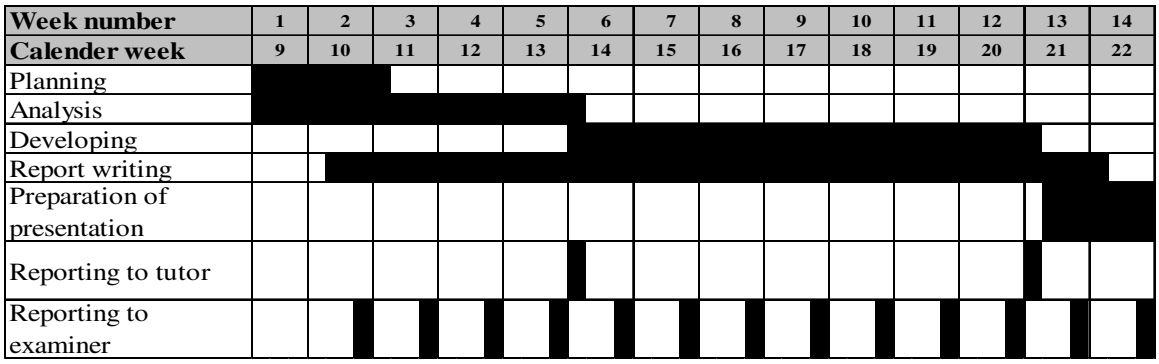

Figure 1 – Time plan described in a form of a Gantt chart

4 What is Android

Android is a mobile platform developed by Open Handset Alliance, where Google is one of the big actors. The main word for Android is “open” and they push a lot for that word. It makes it possible for a developer to use the core functions in the phone, like making calls, sending messages, using the camera and so on.
Android is using a custom virtual machine called Dalvik, which is designed to optimize memory and hardware resources in the mobile environment. [12]
There is a lot of innovative features in Android. To mention one, in January 2010 they released Android 2.1 and one of the new functions was “Speech-To-Text” that allows the user to talk into the mobile microphone and the phone will convert it to text, then for example you can use it in a SMS.

4.1 Architecture

Android is built up by five major components; application, application framework, libraries, Android runtime and a Linux kernel.

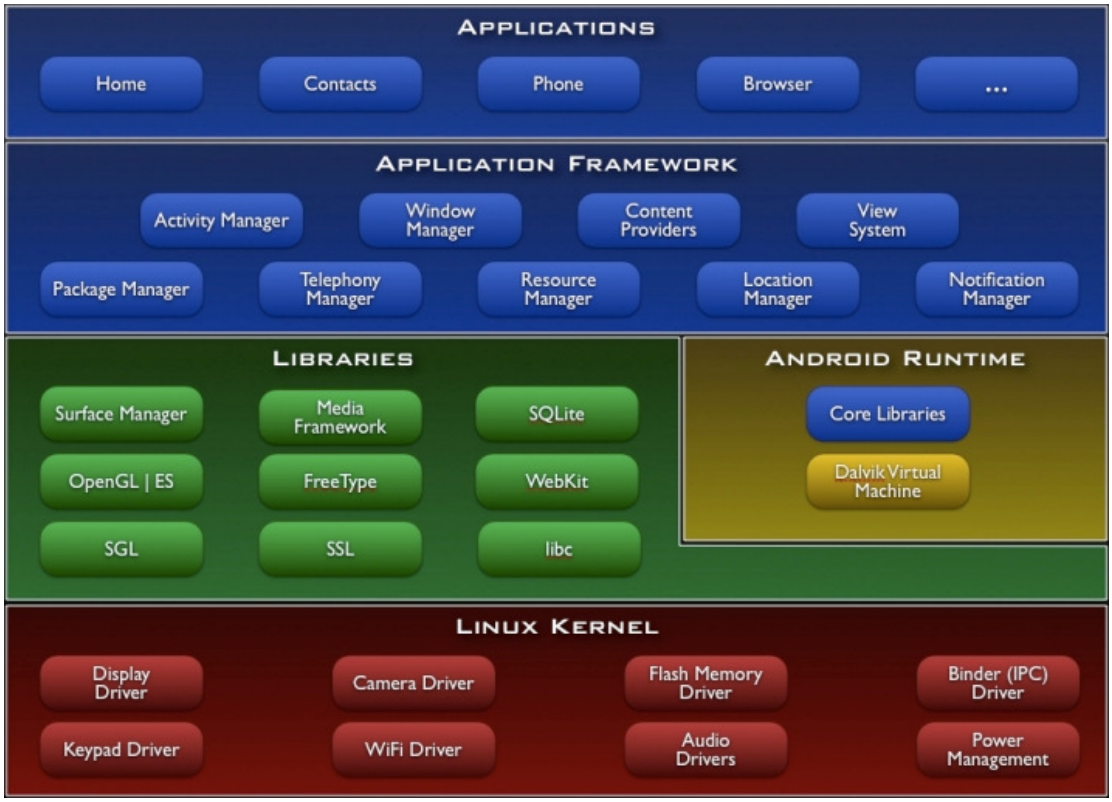

Figure 2 – The major components of the Android operating system. [10]

4.1.1 Application

Application is the top-layer of Androids operative system. Android comes with a set of core applications e.g. browser, calculator and Email. [2]
All applications are written in Java and runs in its own process. The process is activated when the application is executed, and when it is no longer needed it shuts down. [3]

For Android to be able to execute an application it must first be declared in a XML file called AndroidManifest.xml. The manifest file also keeps track of the applications permissions. [4]

4.1.2 Application framework

In this layer the applications are all written in Java. Application framework is the toolkit that all applications, both these who are included in the phone and also the homemade applications uses. Through the application framework developers get full access to all frame APIs that is used in the core applications.

4.1.3 Libraries

Libraries also called the native libraries are written in C or C++ and it is at this level where a lot of the core power of the Android platform comes from. Examples of libraries are Media Libraries; it contains the entire codex that make the support for playback and recording of many video and audio formats, also static image files. Another example is the 3D libraries; it is based on OpenGL ES 1.0 APIs and can use hardware 3D acceleration or the 3D software rasterizer. [2, 11]

4.1.4 Android runtime

The main component in the Android runtime is Dalvik Virtual Machine. Dalvik VM is built specifically for Android and to run in an embedded environment where there are many limitations, for example limited battery and memory. The applications in Android run in its own process in Dalvik VM and it can run multiple applications at the same time. The Dalvik VM runs files called dex files. It is byte code which has been converted from for example a Java file from a computer.
The main reason for converting a Java file to dex file is that it makes up a more efficient byte code so it can run well on small processors.

4.1.5 Linux kernel

Android is running on Linux kernel version 2.6. The Linux kernel provides security, memory management, process management, network stack and drivers model.

4.2 SDK – System Development Kit

Androids SDK is a development kit for applications to Android. The SDK includes tools to help with the development. Two of the most important tools are Android Emulator and Android Development Tools.

4.2.1 Android Emulator

The Android Emulator is a QEMU-based emulator that makes it possible to design, test and debug in an Android environment without having to do it on a mobile device.

There are some limitations though. The emulator does not support actual phone calls, USB connections, camera/video, hands-free devices, determining connection state, battery level, charge level, detection of SD (Secure Digital) Card or Bluetooth. [5]

4.2.2 Android Development Tools plug in

Android Development Tool plug in (ADT plug in) is a plug in for Eclipse Integrated Development Environment. It makes it easier to develop applications using Eclipse. ADT plug in allows use of the ADT inside Eclipse. For example get access to DDMS, writing XML layout files and a project wizard.

4.2.3 DDMS

The Dalvik Debug Monitor Server (DDMS) is a program that comes with Android's SDK. It allows developers to send mock coordinates, simulate incoming phone calls, send SMS, monitor threads, browse files and pull and push files to the emulator.

4.3 NDK – Native Development Kit

Android NDK is a development kit for writing parts of applications in native C and C++ code. Java is still the main program language for writing application, but it makes it possible to reuse already existing native code. It is also a good way to increase performance in some cases, but it is not equivalent to increased performance. [9]

There must be a shared library were the native methods are implemented. The shared library should be named like this: lib<name>.so.
When the shared libraries are loaded as they must be by the application, only the <name> part should be called. [8]

4.3.1 Limitations

If an API or library is not considered stable by Android, they should not be used. There is a chance that the non-stable APIs will not be supported in the next release of Android.

The supported APIs for Android 2.0 and above that are considered stable are:

- The C library (Not all of them)
- The Math library
- The C++ library (very limited amount)
- Zlib Compression library
- Android logging
- Open GL ES 1.1 and Open GL ES 2.0

The C++ libraries that are supported are cstddef, new, utility and stl_pair. It is possible to use the STL library through STLPort. [7] At the moment there is no support for Exceptions or RTTI. [8]

5 Port native code

There are three different approaches to port an existing program to Android. The first approach is to use the official NDK with limited support for standard C++ libraries. The second is to modify the NDK so it gives a full C++ support. The last approach is to not use a NDK, but instead compile the program source code along with the Android source code.

5.1 Using NDK

This way is how the Android developer team intended native code to be written, with the official NDK. The benefits are that applications are guaranteed to work on later versions of Android. It is easy to install and can even be uploaded to Android Market, making them available to everyone. The drawback of using this approach is obvious, the limited support for C++ standard libraries, lack of support for exceptions and RTTI.

5.2 Modified NDK

Using a modified NDK is not advised, but it works at least for the version of Android that it is modified for. With a modified NDK it is possible to get full C++ support with Exception and RTTI etc. The reason why it is not recommended to go ahead with this approach is there are no guarantees that the application will work in later versions of Android.
And there is a chance that applications built with a modified NDK will not load correctly or crash. A lot of tests need to be run and evaluated before it is safe to say that a modified NDK is stable.

5.3 Compile with source code

The last approach is to do it without NDK, and instead compile the C++ code along with the Android source code. This way there is no need to run the program in the Dalvik virtual machine. And that eliminate the need to wrap the program in Java and/or use Java Native Interface.
This is not without drawbacks either. The main problem is that it needs to be compiled with the Android source code. This in turn makes it difficult to distribute and install. The procedure of compiling the code along with Androids source code will have to be done again, if the Android device would ever be updated to a new version.

5.4 What will the future bring

Android is planning to extend their support for C++ in future versions of NDK. Both support for exceptions and RTTI will sooner or later be added to the NDK. Google can not give an estimated time of arrival for these updates, but meanwhile they encourage the use of Crystax's modified NDK. [9] Which got a much bigger support for C++ then the official version does, but David Turner who is an Android engineer stress that they can not give any guarantees that it will cover all cases. [Appendix A]

6 CT - Application for Crisis Training

6.1 Background

Saab has since they started always been one of the world-leading suppliers of military training programs. Today, they also look at what kind of solutions they can offer for civil training.
CT is one of the projects that Saab is now running and they can offer training solutions for accidents in the "normal" life where many authorities are involved, for example a traffic accident between a big train and a school bus. Saab has set up a web system for this and a template with the functions a prospective application could have.

6.1.1 CT Components

There are 3 main components in CT.

- CT Server
    - Hosts database back end and server specific logics
- CT Web
    - Application that runs on a web server
    - Full support for all exercise phases
    - Minimal install and roll out effort required
- Mobile application
    - Application that runs in mobile devices
    - Supports real-time evaluation from field

6.1.2 Exercise Roles

In a CT training exercise there is a lot of people involved. At the top there is a Trainee Organization Manager who sets up the whole exercise and at the bottom the Opposite actors are, and it is them who's going to use the application NILS.

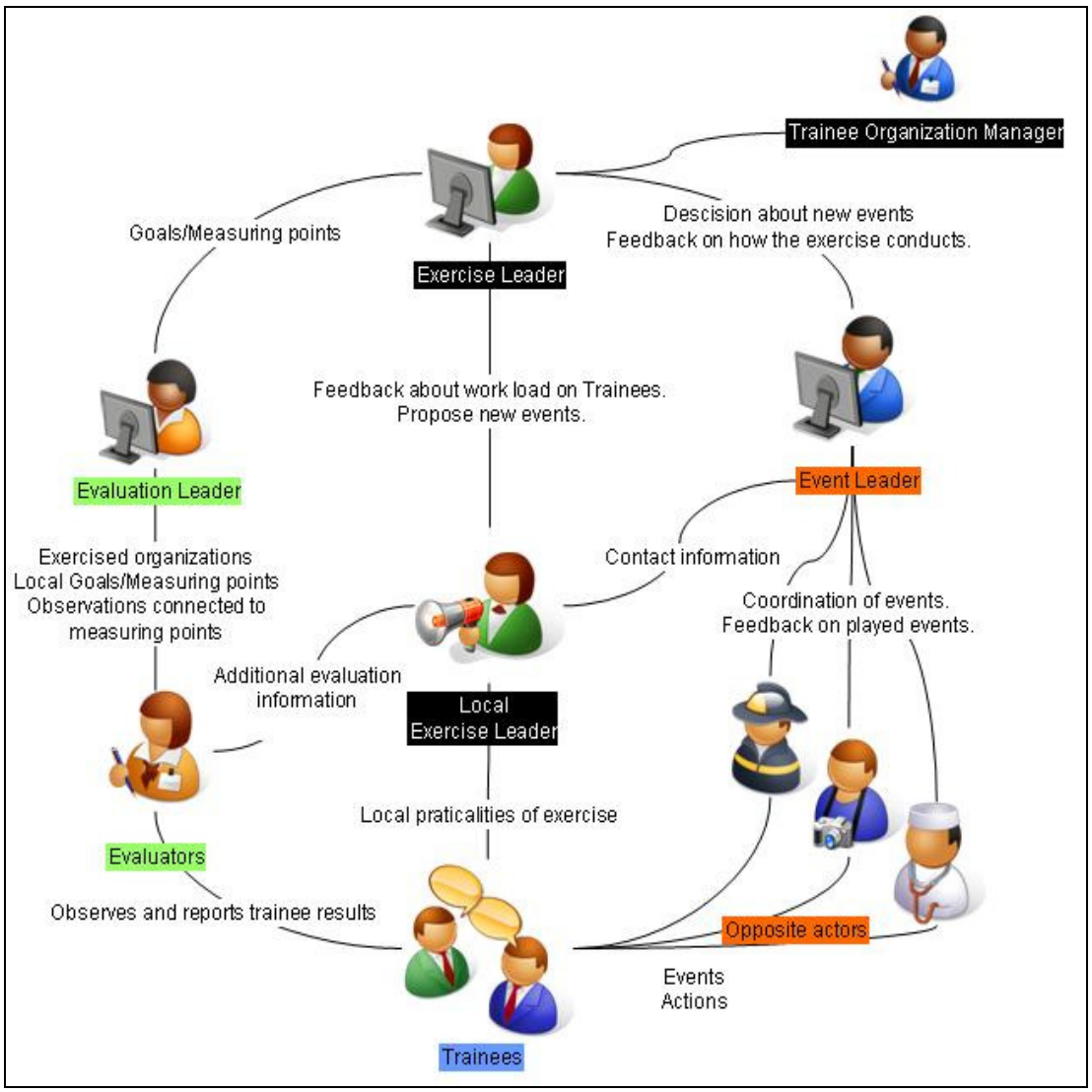

Figure 3 – Show the relations between the various persons in the exercise.

7 NILS – Network Injects Logistic Service

One step to offer CT and a complete solution for the customer is to have an Android application for potential actors.
NILS is based on the CT web interface and has all the functions that the web system has. The main purpose with NILS is to be a simple Android application that you can put in the hands of anyone that is going to be a part of the training.

7.1 Software

NILS is built in Eclipse Classic 3.5.2 which is an open development platform from Eclipse Foundation. One of the main things about NILS is that it is build with pretty simple UI (User Interface) so everyone involved can understand and use it. DroidDraw which is a UI editor for the Android Cell Phone Platform is used for that.

7.1.1 How to get started with Eclipse

1. Download Eclipse Classic
2. Download and extract Androids latest SDK
3. Install the ADT plugin
    a. Start Eclipse.
    b. Press “Help” > “Install New Software…”.
    c. Press “Add” and type in a suitable name in the name field and in the location filed add the following address: “https://dl-ssl.google.com/android/eclipse/” alternatively “http://dl-ssl.google.com/android/eclipse/”.
    d. Mark Developer Tools, press next and then follow the instructions and accept the license agreement.
    e. Press “Window” > “Preferences”
    f. Select Android and browse to where you have extracted the Android SDK and press “OK”.
    g. Press “Window” > “Android SDK and AVD Manager”.
    h. Press “Available Packages” and mark desired components.(All if you unsure which you need)
    i. Press “Install Selected” and follow the instructions.
4. Press “File” > “New” > “Project…”.
5. Select “Android” > “Android Project” and then follow the instructions.

Software	By	Version
Eclipse Classic	Eclipse Foundation	3.5.2
DroidDraw	DroidDraw	r1b14
Visual Studio	Microsoft	2008
Java Runtime	Sun Microsystems	v6 u20

Figure 4 – Table of software that were used during developing of NILS.

7.2 Activities

Developing an Android application is a bit different than developing Java programs, where all the calls are made in the main class. In Android and of course also in NILS it is built with a lot of activities instead. In NILS there is an "activity" for every page that is shown and also an XML file to every page where the layouts and views are described.

7.2.1 Activity

Developing in Android is pretty much about activities. Every new page that is shown in NILS is a new activity. The activity class takes care of creating new windows and connects the new window to the layout that is declared in the XML file with the function setContentView (View).
Windows can be showed as full-screen windows, floating windows or embedded inside another activity.

```
public class Login extends Activity {
```

Figure 5 – Example of how to create the Login activity.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.login);
```

Figure 6 – Example of initializing an activity (onCreate(Bundle)). Also the call to the layout file, in this case login.xml.

7.2.2 Activity Lifecycle

Activities in applications are managed as an “activity stack”. “When a new activity is started, it is placed on the top of the stack and becomes the running activity”. [13] There are seven methods in the activity lifecycle. To understand the complete chain here is an explanation on what all of them do:

onCreate
This method is called in the beginning when the activity starts. Here are all normal static set up created, like creating views and lists. onCreate is always followed by onStart.

onStart
This method is called just before the activity becomes visible on the screen. onStart is followed by onResume if the activity can become the foreground activity, otherwise, the activity it is followed by onStop.

onResume
Is called after onStart and is the first method after the activity has become visible. Now the application can receive input from the user from touch inputs and keyboard. onResume is also used if another activity temporary takes the foreground place and then exit, then onResume become the foreground activity again. onResume is always followed by onPause.

onPause
Is called at the same point the system is resuming another activity and giving that activity the foreground. The activity which now is onPause is no longer visible for the user, and waits for either onResume to come back as foreground activity or onStop to become invisible to the user.

onStop
When an activity is no longer visible because another activity has put it in the background, this method is called. If it follows by onRestart the activity goes back to onStart or if it follows by onDestroy, the activity will be “killed”.

onDestroy
To destroy an activity Android uses the method “finish ( )”. An activity can also be destroyed because the system needs the resources it uses for another activity.

onRestart
This method is called after onStop and the activity

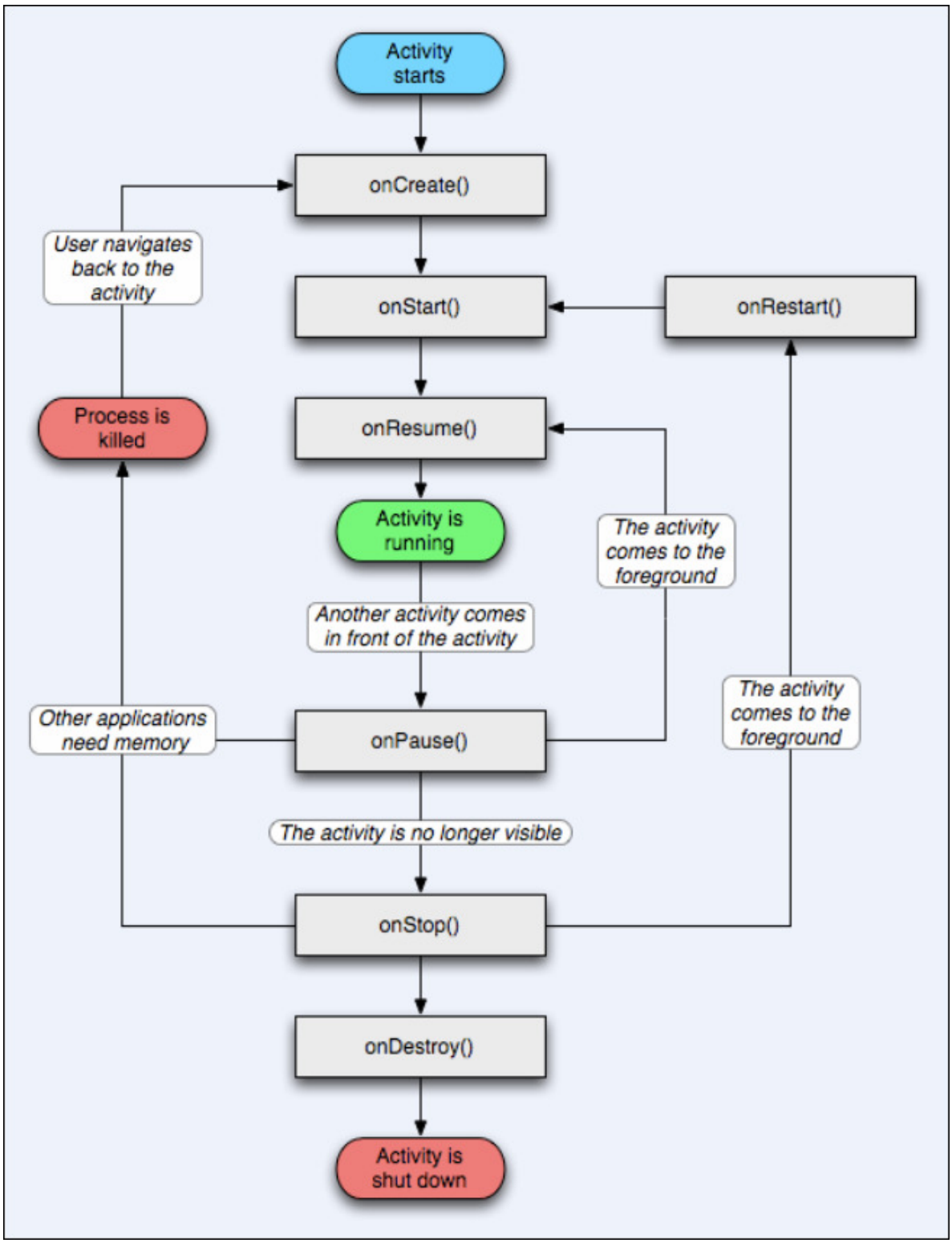

Figure 7 – Activity Lifecycle [14]

7.2.3 AndroidManifest

This file is like the nave of the application. NILS and all other applications must have an AndroidManifest.xml file where all the essential information is present.

First thing that is declared is the Java package name for the application. The package name is a unique identifier for the application [4]. In NILS it is "com.saab.NILS".
In NILS there is also a "uses-permissions" that allow NILS access to internet. In other application there is other permissions declaration, for example to get permissions to Google's embedded API's.
The rest of AndroidManifest.xml in NILS contains all the declarations of the activities.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.saab.NILS" android:versionCode="1"
android:versionName="1.0">
          <uses-permission android:name="android.permission.INTERNET" />
          <application android:icon="@drawable/icon"
android:label="@string/app_name">
                    <activity android:name=".Login"
android:label="@string/app_name"
                    android:theme="@android:style/Theme.NoTitleBar">

                                        <intent-filter>
                                        <action
android:name="android.intent.action.MAIN" />
                                        <category
android:name="android.intent.category.LAUNCHER" />
                                </intent-filter>
                    </activity>
                    <activity android:name="Current"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="Completed"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="All"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="Home"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="CreateNew"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="Handle"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="TelephoneList"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
                    <activity android:name="ListCompleted"
          android:theme="@android:style/Theme.NoTitleBar"></activity>
```

Figure 8 - Information from NILS's AndroidManifest.xml

7.3 User Interface

There are a lot of different choices in Android when it comes to layouts, views, widgets and other styles. Layout can also be built on each other, so for example there can be a Linear Layout inside a Table Layout. The layouts that are used in NILS are Tab Layout, Linear Layout and Table Layout.

7.3.1 Tab Layout

Tab Layout is a very useful layout because it makes a better page overview in applications instead of showing one page at time.
In mobile application where the screens are small in relation to all information that needs to be shown, it is very useful. Also because no one want to go to deep into the application and have to push the back button 10 times to get back to the main page.

7.3.2 Linear Layout

Linear Layout is a layout that displays views in a linear direction, either horizontally or vertically. NILS's login page uses this layout, also every view in NILS's Tab Layouts uses Linear Layout.

7.3.3 Table Layout

Table Layout is similar to Linear Layout but like the name says it is easier when a table looking view is needed. NILS's home page uses Table Layout for example, in that case just because buttons automatically stretches to full size of the screen.

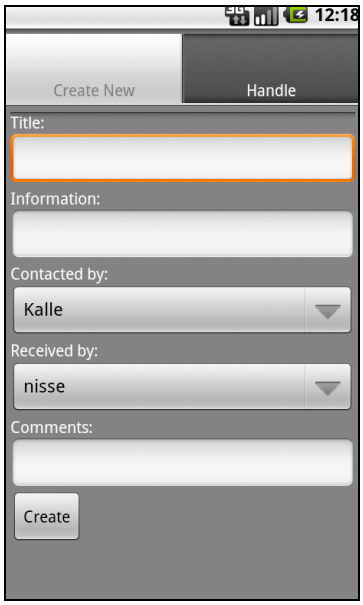

Figure 9 – Linear Layout inside a Tab Layout

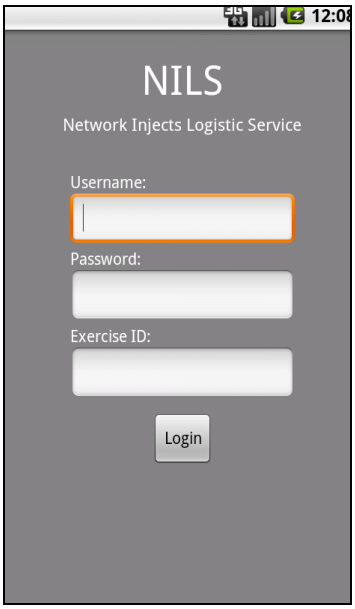

Figure 10 – Linear Layout

Figure 11 – Table Layout

7.4 XML

All layout structures for the activities in NILS are declared in XML files. Every activity in NILS has its own XML file where the layouts, like positions of text views, buttons, background color and text size are declared.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/loginLinearLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical"
xmlns:android="http://schemas.android.com/apk/res/android"
android:background="@drawable/background">
<TextView
android:id="@+id/loginNILS"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginTop="20px"
android:layout_gravity="center_horizontal"
android:text="@string/loginNILS"
android:textSize="35sp"
android:textColor="#FFFFFF">
</TextView>
</LinearLayout>
```

Figure 12 – A piece of code from an XML file. This XML consists of a text view inside a Linear Layout.

NILS also has a "strings.xml" file where all text are declared and linked to each XML file. It makes it easy for Saab to add support for new languages when they are going to implement the product in different countries.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="appName">-=NILS=-</string>

    <string name="loginNILSShort">NILS</string>
    <string name="loginNILS">Network Injects Logistic Service</string>
    <string name="loginUsername">Username:</string>
    <string name="loginPassword">Password:</string>
    <string name="loginExerciseId">Exercise ID:</string>
    <string name="loginLoginButton">Login</string>

    <string name="homeMyInjectsButton">My Injects</string>
    <string name="homeMyOutjectsButton">My Outjects</string>
    <string name="homeHelpButton">Help</string>
    <string name="homeAboutButton">About</string>
    <string name="homeTime">Time:</string>
    <string name="homeTrainingPeriod">Training period:</string>
    <string name="homeStartedInjects">Started injects:</string>
    <string name="homeCompletedInjects">Completed injects:</string>
    <string name="homeReportedOutjects">Reported outjects:</string>
```

Figure 13 – A piece of code from strings.xml

8 Web service

An ASP.NET web service was written in C# in Visual Studio so that NILS can communicate with the SQL database that stores all the information about the exercises. The web service is put as the top layer of CT. It does not query the database directly, but instead it communicates with the layers below and they in turn query the database.
The communication between the client and the server is made with HTTP requests.

8.1 JavaScript Object Notation

JavaScript Object Notation, or JSON, is a data-interchange format that reminds of XML but it is more lightweight. It is based on JavaScript, but is languages independent so it is a good way to send data between different platforms. JSON can be built in two ways, as name-value pairs or as a list of values. [16]

8.2 Server

The web service got 10 public web methods,

- getAbout
- getAll
- getCompleted
- getCreateNew
- getCurrent
- getExerciseRoles
- getHandle
- getStatus
- login
- updateDatabase

Each method got a corresponding class that is used to form the data into objects with the preferred layout. The web service then automatically translates the object into the format that the client has requested it in.

```
        public class Completed
        {
            public string ID { get; set; }
            public string Event = "Completed";
            public string Title { get; set; }
            public string Status { get; set; }
            public string Reference { get; set; }
            public string Instruction { get; set; }
            public string From { get; set; }
            public string Starts { get; set; }
            public string Ends { get; set; }
            public string Started { get; set; }
            public string Ended { get; set; }
            public string Result { get; set; }
        }
```

Figure 14 - This class is used to form the information about a completed inject

In this case all information is at the same level. To add a sublevel the same procedure as creating the "Completed" object has to be done to the sublevel, but instead for example a string, a class will be created to form the sublevel data into an object.

```
        public class Injects
        {
            public string Numerator { get; set; }
            public string Denominator { get; set; }
        }
```

Figure 15 - The Injects class represent the sublevel for the class in figure 16

```
public class Status
        {
            public string ID { get; set; }
            public string Event = "Status";
            public string TrainingPeriod { get; set; }
            public Injects StartedInjects = new Injects();
            public string ReportedOutjects { get; set; }
        }
```

Figure 16 – This class is used to for the information with a sublayer StartedInjects

The JSONObject that will be sent from the server will look like figure 17.

```
{
"ID":"12",
"Event":"Status",
"StartedInjects":     {
                      "Numerator":"3",
                      "Denominator":"4"
                      }
"ReportedOutjects":"2"
}
```

Figure 17 – Note that the indent and text-wrapping is just added here to make it more lucidly

8.2.1 Requests

The methods that start with “get” handles information requests from the Android client. For example the getCurrent method will return a JSON object with all injects that are not finished or cancelled. To get the information an exercise id and a user id must be sent along with the request. The web service gets the information that is requested from the database, and then forms it to an object as mentioned before. The object is then sent to the client as a JSON object.

8.2.2 Updates

All updates and outjects go through the updateDatabase web method in the web service. It works much like the requests, but instead of just sending exercise id, user id and in some cases a request message, all the information that should be updated is sent along with the message. As in the requests, the content type is JSON. In the updates and creation of new outjects, sublayers are more widely used then in the requests. Parsing JSON is very simple at server side. The information is deserialize into an object that then can be used to extract information from.

```
Completed completed = new Completed();
completed = js.Deserialize<Completed>(update);
```

Figure 18 – Deserializing update into the object completed

Where “update” is the JSON object that should be parsed and “completed” is the same object as in figure 14. The object has to have the same layout as the JSON object, that is with the same layers and correct variable names.

8.3 Client

The client communicates with the server side with Apaches HttpClient. The HttpClient uses the HTTP methods get, put, post and delete. It is in these methods the URL and the message to send is added. Adding a message to a post method is done with an entity. Messages have to be added in key-value-pairs, this can be done in many different ways. In this case JSON is used to transmit the messages. The corresponding web methods on the server side, can directly receive the top layer values from the JSON object into its parameters. The HTTP methods also need headers so that the receiving side knows what kind of information and in what format the client is sending it.

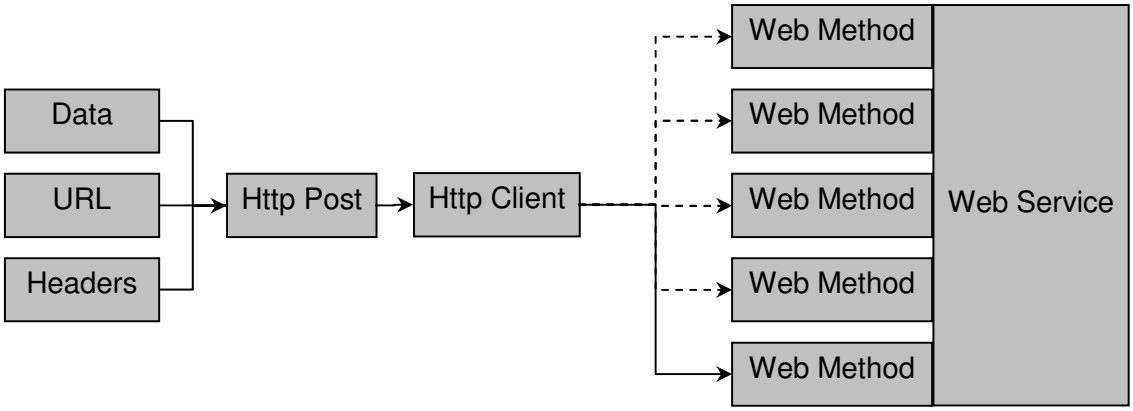

Figure 19 – How the client communicates with the web service

8.3.1 Lists

To get a better overview of injects and at the same time limit the size of data retrieved from the server, NILS first list the most important information about injects. The list only contains titles and time for when injects should be played out. When clicking on an inject in the list, more detailed information about the inject appears. This way only information that is relevant at the time is retrieved from the server.

8.3.2 Parsing JSON

On the client side parsing is not as easy as on server side. There is no quick way of just deserializing the JSON into an object. Instead every layer in the JSON must be extracted and then the values can be extracted with help of the name parameter. So for example to get to a value in the third layer we first need to get through the first and second layer. So for example to get to a value in the third layer, first the second layer needed to be extracted.

```
JSONObject firstLayer = new JSONObject(jsonString);
JSONObject secondLayer = json.getJSONObject("d");
JSONObject thirdLayer = secondLayer.getJSONObject("StartedInjects");
String demoninator = thirdLayer.getString("Denominator");
```

Figure 20 – How to parse a JSON object

The first row is where the result from the HttpClient is reintegrated to a JSON object. The second and third row is where the layers are extracted and the last row is where a value is extracted.

8.3.3 Container

To store the parsed data until it is time to use it, a container class is used. The container got a set and a get function for every value that it can store. This makes it easy to map data to the right text fields. The container class layout supports an implementation of a SQLite database which would increase the ability to store and cache data.

```
{
"firstName": "Nils",
"lastName": "NILSson",
"address":                          {
           "street":"S. Storgatan 37",
           "city":"Helsingborg",
           "postalCode":"25111",
           "country":"Sweden"
                      }
}
```

Figure 21 – Example of a JSON object

9 Conclusions

9.1 Result

In the analysis we examined the possibility to port an existing product to Android. We came to the conclusion that if we wanted to port a native coded program, like a product written in C++, we would need to rewrite the MMI(Multi Media Interface) in Java. The best way of porting a product is to use Android's NDK. But since the current version of the NDK lack some vital libraries, for example exceptions, at least for porting the products that Saab wants, we decided to not go ahead with the port. Instead we decided to write an application in Java from scratch. That resulted in the application NILS which is based on CT web interface. The development of NILS and the analysis of Android have shown that Android got what it takes to run applications that suits Saab.

9.2 Discussion

To develop on Android a fairly good knowledge of Java programming is recommended since applications are written in Java. We also recommend having a clear structure of the application before starting programming.

One thing to really focus on is to understand the Activity Lifecycle. With all its methods you must use them for example to make sure you not loose information because the system closes an activity unexpected because it needs resources that your activity uses to another activity.

One problem that arose when writing the server-client communication was that it was not always easy to know on which side errors occurred. Running debug mode on both Eclipse and Visual Studio made it possible to catch most of the errors and on which side they occurred.

DroidDraw was another problem we had to deal with. To make the UI for NILS we used version r1d14 which was the newest version. Unfortunately it did not support drawing UI's for the newer Android phones with higher resolutions than the older ones. That meant we had to modify a lot in the XML code and could not rely on what we produced in DroidDraw to be 100% accurate.

Another problem that needed much time to think about was how we were going to make an application for a mobile phone from an existent web based system. But it wasn't the functions that were the problem, no, it was to make a simple and lucid interface with all the information that the web system had.

Plug-and-play is not always as easy as it should be. Several times the computer could not find drivers for the phone, even though it worked fine 2 minutes earlier. It was more prominent when we had used one phone and switched to another.

One thing that is very tiresome when developing on the emulator is that it takes time to start up the emulator. It can take up to several minutes sometimes. This only occurs the first time the emulator is started or if for some enigmatical reason the IDE would lose the connection to the emulator and it needed to be manually restarted.

We first tried to develop in the same workbench that we located on an external server. That did not work out as well as we had hoped, every time we saved the project it was a big delay. Instead we developed locally and did backups on the server a couple of times per week.

The time plan we setup in the beginning of this project has been almost perfect. Besides that we started the developing one week before we had planned to and ended it one week earlier.

9.3 Future work

There are many improvements that can be made to the application to make it more user-friendly and reliable.

9.3.1 SQLite

A SQLite database would make it possible to cache data and limit the amount of data needed to be retrieved from the server. The class Container would be a good place to implement the database. Instead of saving the information in variables when using the “set-functions”, an insert could be made to the database. It would be good to set a flag to true in the database if the data has been sent to the server successfully. This way if the phone got no signal the flag would not be set to true. And when there is a signal, the information that does not have the flag set to true would be sent.
With a database the application could work in offline-mode for a period of times if necessary which is good if there is a bad reception in the area where the exercise is taking place.

9.3.2 Service

A service is a background process that does not require a UI. The application would benefit from being more service oriented. A service would work great with the database. If all the communication with the server would be in a service the delays when changing between activities would disappear. All information could be retrieved from the local database, and the service would keep the local database up to date with information. The service could be set to update the most vital data frequently and less vital data could be update more seldom. For example injects are more vital then users in the system and therefore should be updated more frequently.

9.3.3 Notification

Add a notification when there is a new inject, just like when getting a SMS, a signal and a symbol in the notification bar that indicates that there are a new inject. Another thing that would be good for the user to have, is a reminder when the time comes for an inject to play out.

10 References

1. Android developer's homepage. http://developer.android.com/index.html (March 2010)
2. Android developer's homepage. http://developer.android.com/guide/basics/what-is-android.html (March 2010)
3. Android developer's homepage. http://developer.android.com/guide/topics/fundamentals.html (March 2010)
4. Android developer's homepage. http://developer.android.com/guide/topics/manifest/manifest-intro.html (March 2010)
5. Android developer's homepage. http://developer.android.com/guide/developing/tools/emulator.html (March 2010)
6. Android developer's homepage. http://developer.android.com/guide/developing/eclipse-adt.html (March 2010)
7. STLPorts homepage. http://www.stlport.org/index.html (March 2010)
8. NDK documentation. STABLE-APIS.TXT(March 2010)
9. Crystax homepage. http://www.crystax.net/android/ndk-r3.php (March 2010)
10. http://developer.android.com/images/system-architecture.jpg (May 2010)
11. Android developer's homepage. http://developer.android.com/videos/index.html#v=QBGfUs9mQYY (May 2010)
12. Android homepage. http://www.android.com/about (May 2010)
13. Android developer's homepage. http://developer.android.com/reference/android/app/Activity.html (May 2010)
14. Android developer's homepage. http://developer.android.com/images/activity_lifecycle.png (May 2010)
15. Android Application Development, 1st Edition. http://androidapps.org.ua/i_sect11_d1e703.html (May 2010)
16. JSON homepage. http://www.json.org/ (May 2010)
17. http://www.swedroid.se/wp-content/uploads/2010/05/android-market-v19-2010.png (April 2010)

11 Dictionary

QEMU	Open source emulator. (www.qemu.org)
SDK	Software Development Kit
NDK	Native Development Kit
API	Application Programming Interface
STL	Standard Template Library
STLPort	Multiplatform open source version of STL
RTTI	Run-Time Type Information
ABI	Application Binary Interface
JNI	Java Native Interface
CT	Crisis Training
UI	User Interface
NILS	Network Injects Logistic Service
cstddef	C++ standard library
Inject	Instruction from the exercise leader
Outject	Observation from the opposite actors
Crystax	Alias for an Android developer [9]

12 Appendix A

From: David Turner
To: Andreas Carlsson

Hello Andreas,

Yes, I still plan to support that but nothing's ready yet, and I cannot discuss any ETA for this right now.
In the mean time, I encourage you to try crystax's modified NDK (see http://www.crystax.net/android/ndk-r3.php) which shall support them.

That's probably the closest thing to the real thing you can get right now, though I can't make promises it's going to cover all cases.

Regards