UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Oleg Petšjonkin

Migration of Native Android Applications to HTML5

Master Thesis (30 EAP)

Supervisor: Satish Srirama, PhD
Co-supervisor: Huber Flores, MSc

Author:.................................... ".....” May 2012

Supervisor:............................... ".....” May 2012

Co-supervisor:............................... ".....” May 2012

Professor:................................. ".....” May 2012

TARTU, 2012

Abstract

Mobile devices market as well as mobile application market is growing rapidly. When starting mobile application development, often the first question is which platform to choose? This consequently leads to limitations in software usage. Theoretically, this problem can be solved by cross-platform technology. There are several methods to "write once run anywhere" application; however there are several major drawbacks for each of them.

Flash for mobile devices is no longer supported for Adobe, mobile website solution requires Internet connection and have no access to native mobile capabilities.

HTML5 is a new technology that tries to solve cross platform in-compatibility, supports offline web-applications and allows using some native mobile features. Despite the fact it is still under development, the support of the browsers is really good even on mobile devices. As a result, HTML5 have a potential to become major tool for cross-platform development.

The goal of this work is to investigate possibilities for converting Android native application to HTML5 and write prototype application which will do the conversion. Definitely there are applications that are using Android specific features which are not supported by HTML5, so the first step will be defining scope of application that can be converted. Next step will be investigation and choosing tools which can help to achieve the goal. The final step of this work will be designing and developing prototype which performs conversion.

Contents

List of Figures

1

Introduction

1.1 Introduction

Nowadays, the development of mobile applications has increased dramatically, this can be observed due the amount of applications that exist in mobile application markets such as Play Store or Apple App Store (mobile market is very volatile. iOs and Android are leading, Symbian market share is constantly decreasing). This high demand for mobile software have made the smartphones become a part of everyday life. According to latest reports (1) smartphones sales worldwide in the fourth quarter of 2011 increased more than 47% from the fourth quarter of 2010. Smartphones sales depends not only on hardware capabilities, but mostly on available software.

Generally, when starting the development of a mobile application, often the first concern is what platform to choose? (e.g. Android, iOs, Windows 7, etc.). The answer to this question is mainly based on the purpose of the application (e.g. mobile game, social mobile application, etc.). However, developing a mobile application for a specific platform limits its scope, in terms of distribution, commercialization, etc. Thus, making the application perspective narrow to one single mobile vendor.

Furthermore, a considerable effort and knowledge in low level programming techniques (e.g. different programming languages, SDK, tools, etc.) is required for porting the application between platforms, and thus in general most of the application are written once targeting an specific platform.

However, with the introduction of cross-platform technologies such as HTML5 for the development of mobile software, the applications can be written once and run in multiple platforms on the top of the mobile browser. Cross-platform development tools reduce the time and the cost in the mobile development process as it is aim to provide high portability features and rich tools for managing some (e.g. PhoneGap, etc.) of the underlying hardware resources from the browser. However, it is necessary an extensive analysis in this technology as multiple issues can arise such as decreasing the mobile application performance (e.g. graphical aspects, etc.), unsupported hardware in the HTML layer (e.g. cameras) , etc. For instance, the functionality of an HTML5 application could be limited by the fixed memory assigned to the browser by the mobile operating system.

Despite the growing interest in cross-platform development tools, large amount of applications are still written native. Moreover, many of them are supported only by Android and iOs or even by one of those platforms. The question about automatic application conversion from one platform to another, or even to cross-platform solution is still unanswered.

In order to investigate the possibilities of migrating Android native applications (written in Java) to HTML5 (based on JavaScript), this thesis studies the semantic and syntactic transformation of Java code to JavaScript.

1.2 Contributions

To overcome the issues regarding the migration of native application to cross-browser platforms and to foster the development of portable mobile applications. This thesis proposes a framework that enables the conversion of Java to JavaScript. The solution is based on GWT and it uses ANTLR. It consist of two parts. First one is GWT module that wrapps GWT classes into Android structures to reduce code amount changes needed for GWT compatibility. Second part is Java to Java converter, which made code changes that can not be achived by simple class wrapping. The rest of the thesis is organized as follow.

1.3 Outline

Chapter 2: provides short overview of Android (2) platform and HTML5 technology. First section of this chapter concentrates on Android application fundamentals like

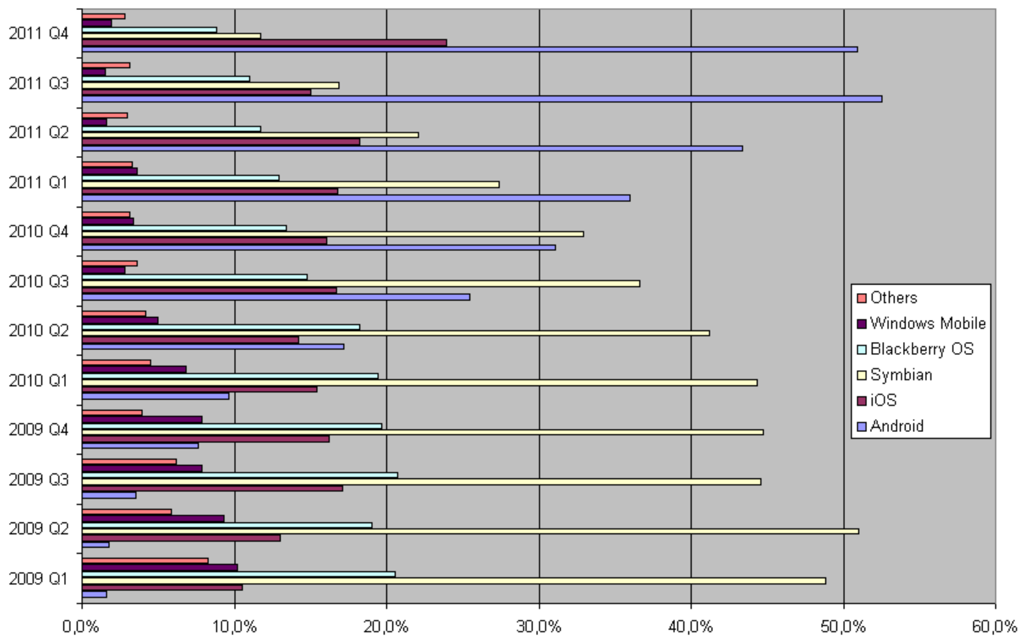


Figure 1.1: Worldwide mobile Os marketshare

basic components and application content. Section also gives examples about Android application resources customization for different device configurations. Second section briefly describes HTML5 differences from HTML4 and shows those features support by different browsers.

Chapter 3: discusses the state of art addressed by this thesis. Firstly, it compares HTML5 and native applications. Secondly, short overview to bridging frameworks is made. And lastly, chapter gives information about three tools: Java2Script (3) and GWT (4) - Java to JavaScript compilers and ANTLR (5) - parser generator.

Chapter 4: defines the problem regarding lack of tools for automatic conversion.

Chapter 5: describes the development of the framework created for transforming native android application to HTML5. It also limits the problem scope and shows automatic conversion application design and work principle. At the end of this chapter limitations are described and onversion result for one application is introduced.

2

Background Information

This chapter provides background information about Android operating system and HTML5 technology. It also provides Android application fundamentals and HTML5 new features comparing with HTML4.

2.1 Android operating system

Android is Linux based operating system for mobile devices (6). It is developed by Open Handset Alliance and consists of operating system, middleware and key applications. Android kernel is based on Linux kernel with several layers on top of that as shown on figure 2.1.

Application development is done using Java programming language. Android application consists of one or more application components. They are *Activity*, *Service*, *Content Provider* and *Broadcast Receiver*. *Activity* is typical entry point of application. Typically it represents a single screen which user can interact with. *Activity* is created by subclassing *Activity* class and implementing *onCreate()* method.

Another application component is *Service*. Its purpose is to perform long-running tasks in background. *Service* does not provide user interface. Typically *Service* does not require to report result to other component and stop itself after finishing operation. One of the *Service* usage is uploading or downloading files over network.

Next component is *Content Provider*. Its task is to manage access to application data. Application data can be stored in any persistent storage which is accessed by application. Moreover, data can be shared to other applications if *Content Provider* allows it.

Last component described in this section is *Broadcast Receiver. Broadcast Receiver* listens and responds to system-wide broadcast messages. Typically *Broadcast Receiver* is used to establish communications between *Activities*, *Services* and Android system. For example, Android system can notify *Activity* or *Service* about network status or *Service* can notify *Activity* about finishing task.

In addition to code, Android application requires resources such as images, string constants, animation descriptions, menus, styles, dimensions, and layout of activity user interface. Most resources are defined with XML files. Using such approach, it makes it easy to update application various components without code modification.

It is possible to provide different set of resources for different device configurations. This includes:

* Device hardware properties such as screen resolution and physical keyboard availability; this can be used to provide better quality drawables for large screen resolution devices.

* Device software properties such as language, region and platform version; this can be used to provide translated version of string constants for different languages

* Device current state such as portrait or landscape orientation, if it is connected to dock station; this can be used to change activity layout depending on screen orientation.

For each resource included in Android project, unique identifier is generated, which can be used to access this specific resource from application code or other resource defined in XML.

2.2 HTML5

HTML5 is the fifth revision of the HTML standard. A lot of changes made in elements (7). Beside changing and enriching old one, it provides plenty of new elements. This includes new media elements like <*audio*> and <*video*>, new structural elements like <*section*> and <*article*>. Another thing is that HTML5 specifies not only markup

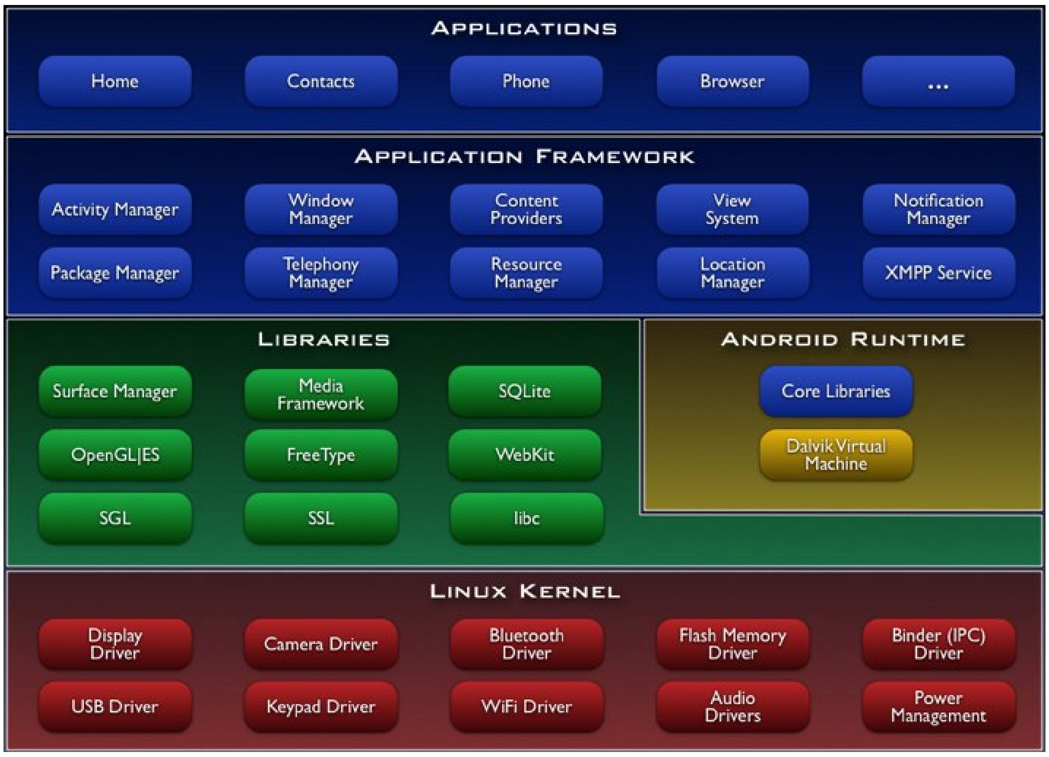


Figure 2.1: Android architecture

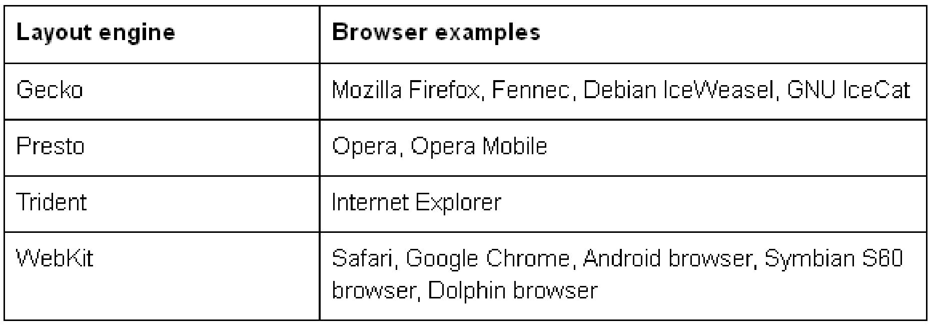
but also APIs which can be used with JavaScript. Many new APIs are introduced such as:

* Canvas element, for 2D drawing
* Offline applications for interacting with application event when network connection is not available
* Native drag and drop - no need to write complex JavaScript code to implement drag and drop functionality - it will be supported natively
* Web storage - key-value storage framework similar to Cookies, but with larger storage capacity and improved API
* Databases API - to manipulate client-side databases using SQL

Some related technologies are not included into W3C HTML5 specification, however they can be used together with HTML5. W3C publishes specification for them separately. Here is some of them:

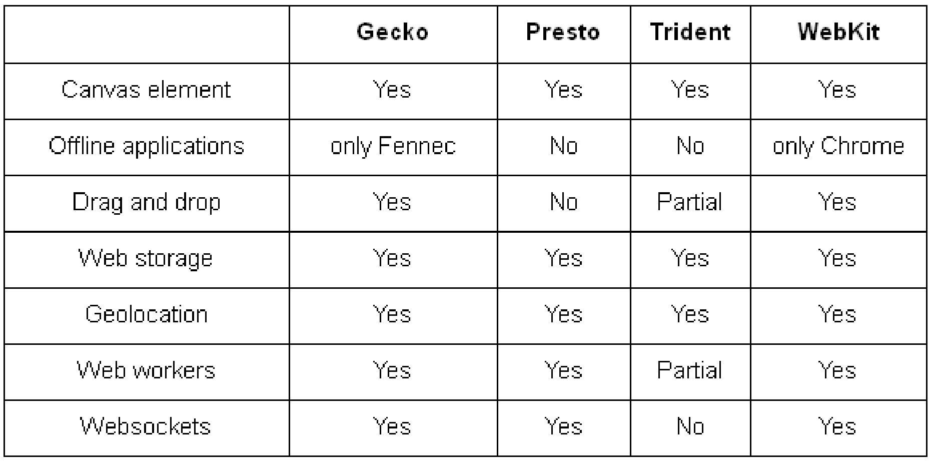
* Geolocation - provides access to geographical location information associated with client
* Web workers - give ability to run background scripts independently form UI scripts. This will help to keep web page responsible and execute long running scripts simultaneously
* WebSockets - to enable bidirectional communication between browser and webserver

Despite the fact that HTML5 specification status on May 2012 is still under development, it is partially supported by many browsers (8). It includes Android and iOs browsers which are based on WebKit (9). For convenience, layout engine names are used instead of browser names in HTML5 APIs support table 2.3. Table 2.2 shows example browser names behind each engine.



Layout engine	**Browser examples**
Gecko	Mozilla Firefox, Fennec, Debian IceWeasel, GNU IceCat
Presto	Opera, Opera Mobile
Trident	Internet Explorer
WebKit	Safari, Google Chrome, Android browser, Symbian S60 browser, Dolphin browser

Figure 2.2: Layout engines



	Gecko	**Presto**	**Trident**	**WebKit**
Canvas element	Yes	Yes	Yes	Yes
Offline applications	only Fennec	No	No	only Chrome
Drag and drop	Yes	No	Partial	Yes
Web storage	Yes	Yes	Yes	Yes
Geolocation	Yes	Yes	Yes	Yes
Web workers	Yes	Yes	Partial	Yes
Websockets	Yes	Yes	No	Yes

Figure 2.3: HTML5 APIs support

2.3 Summary

HTML5 provides a lot new features comparing with its predecessor and is still under development, however its support by browsers is on the good level. Despite the amount of new features, HTML5 is limited by browser capabilities and can not use all the features provided by Android platform.

3

State of the Art

The state of art in this thesis describes advantages and disadvantages of HTML5 applications over native applications on mobile platforms. It gives short overview of native bridging frameworks, describing features they expose to JavaScript. It also shows possible ways to converting Java code into JavaScript. The last section of this chapter concentrates on ANTLR - tool for parser generation.

3.1 HTML5 and native applications

Mobile devices market as well as mobile application market is growing rapidly. When starting mobile application development, often the first question is which platform to choose? There are several methods to "write once run anywhere" application; however there are several major drawbacks for each of them. Flash for mobile devices is no longer supported for Adobe, mobile website solution requires Internet connection and have no access to native mobile capabilities. HTML5 is a technology that solves cross platform in-compatibility, supports offline web-applications and allows using some native mobile features.

Native application have some advantages over HTML5 application (10), (11). One of them is that native application can do more. Native application support multi touch, hardware sensors, can provide access to the device operating system. Native android application can communicate with each other using intents. This cant be achieved by using HTML5.

From the other hand, web standards are evolving. With HTML5 we already can run application offline, access local storage, playback multimedia files, create socket connection (12). There is also possibility to create hybrid application - native application with embedded web view which provides user interface. This approach gains benefits from both native and HTML5 applications, however it adds complexity to your application.

Another native application benefits are speed - HTML5 applications have runtime layer, they can use such hardware acceleration as GPU, they can use multithreading, they get native look and feel just by using native toolkit. Also HTML5 applications are limited in local storage and lack of source code protection.

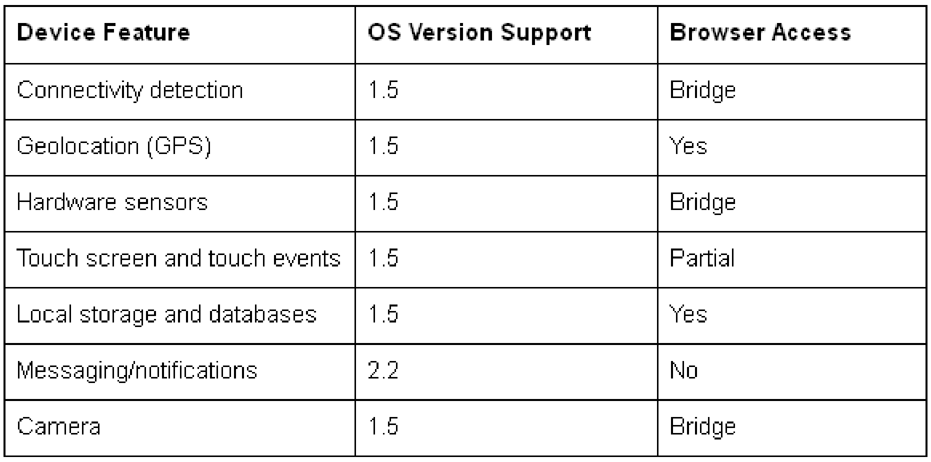
3.2 Bridging frameworks

Despite the fact that HTML5 supports many useful features for mobile application, the wide range of mobile device features is still uncovered. To enrich pure HTML5 functionality bridging frameworks can be used (13). They give ability to access native functionalities by exposing them to JavaScript. Bridging frameworks encapsulate all supported native functionality and provide JavaScript APIs for accessing them, so developers can concentrate only on writting code using HTML, CSS and JavaScript.

3.2.1 PhoneGap

PhoneGap (14) is open source mobile development framework developed by Nitobi Software. Framework uses HTML, CSS and JavaScript. Framework creates native WebView - controller and all application logic is executed inside this controller. Great thing about this framework is plugin support, so basically any phone function support can be added. PhoneGap supports all major mobile platforms including iOs, Android, Blackberry, Windows Phone.

Figure 3.1 shows Android device features, from which Android version they are supported and if they can be accessed by device WebView. From the table it is seen that bridging frameworks gain benefits from HTML5 features like geolocation and local storage, which are supported by layout engine. Some features, mostly hardware related, still require exposing by bridging framework.



Device Feature	OS Version Support	Browser Access
Connectivity detection	1.5	Bridge
Geolocation (GPS)	1.5	Yes
Hardware sensors	1.5	Bridge
Touch screen and touch events	1.5	Partial
Local storage and databases	1.5	Yes
Messaging/notifications	2.2	No
Camera	1.5	Bridge

Figure 3.1: Android Os deatures and browser support

3.2.2 Titanium Appcelerator

Titanium Appcelerator (15) was developed by Appcelerator Inc. and released in 2008. Initially platform provides possibility to create cross-platform desktop applications for PC, Mac and Linux. Later Android and iOs support were added.

For mobile platforms Titanium application are written using HTML, CSS and JavaScript. Using Titanium API developers can get access to native phone capabilities like geolocation and accelerometer. Moreover, Titanium API supports access to native UI constructions, so application gets native look and feel on every platform.

The output package consist mostly of native code. In addition it contains customized WebKit layout engine.

3.3 Java to JavaScript compilers

This section gives short overview of existing Java to JavaScript compilers.

3.3.1 Java2Script

Java2Script is open source tool which translates Java code to JavaScript. Java2Script allow you to build application using SWT as UI framework, and then compiles it to JavaScript (16). It is well integrated with Eclipse JDT and allows to reuse SWT (17) development tools such as SWT Designer.

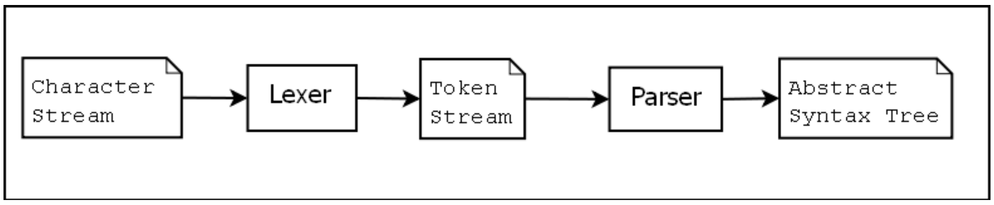


Figure 3.2: ANTLR parser workflow

3.3.2 GWT

Another ready to use software for conversion Java source to JavaScript is GWT - the open source set of tools by Google. GWT can be used as framework for developing mobile and tablet application. One of the useful GWT features is that Java to JavaScript compiler performs code optimization. It includes not only removing dead code but also in-lining methods, to avoid unproductive calls. It also supports several HTML5 features such as Database API, Storage API, Geolocation API (18). Another usefull thing is that GWT can obfuscate JavaScript code.

From the code point of view, GWT supports almost all core Java language syntax and semantics. Several core classes have functionality which is too expensive to support entirely. Lightened version of those classes provided by framework. For example GWT provides own browser-safe timer class which is simplified for running in single-threaded environment. For building user interface GWT uses own widgets library. It includes simple widgets, for example: *Button*, *CheckBox* and *TextArea* as well as compex ones, for example: *TabPanel*, *PopupPanel*, *RichTextArea*.

3.3.3 ANTLR

ANTLR stands for ANother Tool for Language Recognition (19). It is tool developed by Terence Parr for parsers generation. ANTLR grammars are based on Extended Backus-Naur Form (EBNF). Example and explanation of EBNF can be found in Appendix 2. Based on the grammar file ANTLR generates lexer - which converts stream of characters to a stream of tokens and parser - which processes a stream of tokens and generates abstract syntax tree.

3.4 Summary

There are several tools which can help with conversion to HTML5. Few Java to JavaScript compilers are among them. To enchance converted HTML5 application with native capabilities, bridging frameworks can be used.

4

Problem Statement

As described in the previous chapter, the migration of native mobile to cross-browser applications, its feasible by adapting the current tools and specialized frameworks (e.g. GWT, Titanium Appcelerator, etc.) for such task. However, several post-conversion drawbacks that affect the execution of the application on the top the mobile browser, must be considered. This chapter highlight those issues and presents our ideas to address them.

4.1 Problem statement

HTML5 is arising as a prominent technology that has the potential to become primary tool for cross-platform mobile application development. Several tools for cross-platform development like PhoneGap already benefit from its features. However, applications which were already developed for a concrete platform, they have to suffer from several limitations such as distribution model, platform incompatibility, etc. Moreover, such kind of application are difficult to port, reuse and maintain.

On the other hand, pure HTML5 applications can hardly challenge native applications as explained before, however they can be used together with bridging frameworks for extending the capabilities (e.g. Geolocation, etc) of a browser beyond its normal features. Despite fragmentation across browsers, their HTML5 support is on the good level.

Converting native application to HTML5 could be a good start to migrate to cross-platform solution and later application can be enriched using bridging framework. Another thing is that not all applications need scope of smartphone features, so browser capabilities could be enough for them. It is meaningless to convert 3D games, applications that highly rely on multithreading or frequently use device specific capabilities like camera or sensors. However simple 2D games, information providing software - can be converted to HTML5 without loss of its functionality.

The primary goal of this thesis is to investigate possibilities for converting Android native application to HTML5. After researching existing solutions, which can help in conversion, next step will be to design and develop application prototype which enables native Android applications conversion to HTML5. Definitely there are specific features which are not supported by HTML5, so not all applications can be converted. To limit the problem scope and define tools which can be reused, preliminary research need to be performed.

4.2 Summary

Mobile device market is growing rapidly and due its fragmentation cross-platform solutions have a great potential. There are tools for cross-platform development, however migration from native platform still have to be done by hand.

5

Android to HTML5 Converter

This chapter describes prototype, which was developed in this thesis scope. It gives detailed overview of all major parts, describes reasons behind each of those part design.

5.1 Overview

In the scope of this thesis prototype for conversion Android application to HTML5 was designed and developed. Because major programming language used for Android application development is Java and major technology used to provide interaction to webpages is JavaScript, the first thing which required to conversion is Java to JavaScript translator.

There is not much done in Java to JavaScript conversion. Single solution which supports some HTML5 APIs and potentially supports more in nearest future is GWT. It was decided to use GWT as translator. GWT uses its own library for UI components to generate HTML5 elements, however Java to Java conversion is easier task.

Another benefit of using GWT is that intermediate conversion result - GWT compatible source is Java code and it allows to make modification just before translation to JavaScript.

Beside Java source code, Android project usually have resources defined in XML files. Those resources can be accessed using resource IDs, that are generated in projects *R* class. It is widely used technique in Android application development to access those resources from Java code. To preserve such resource access possibility it was decided to convert XML resource files to Java code.

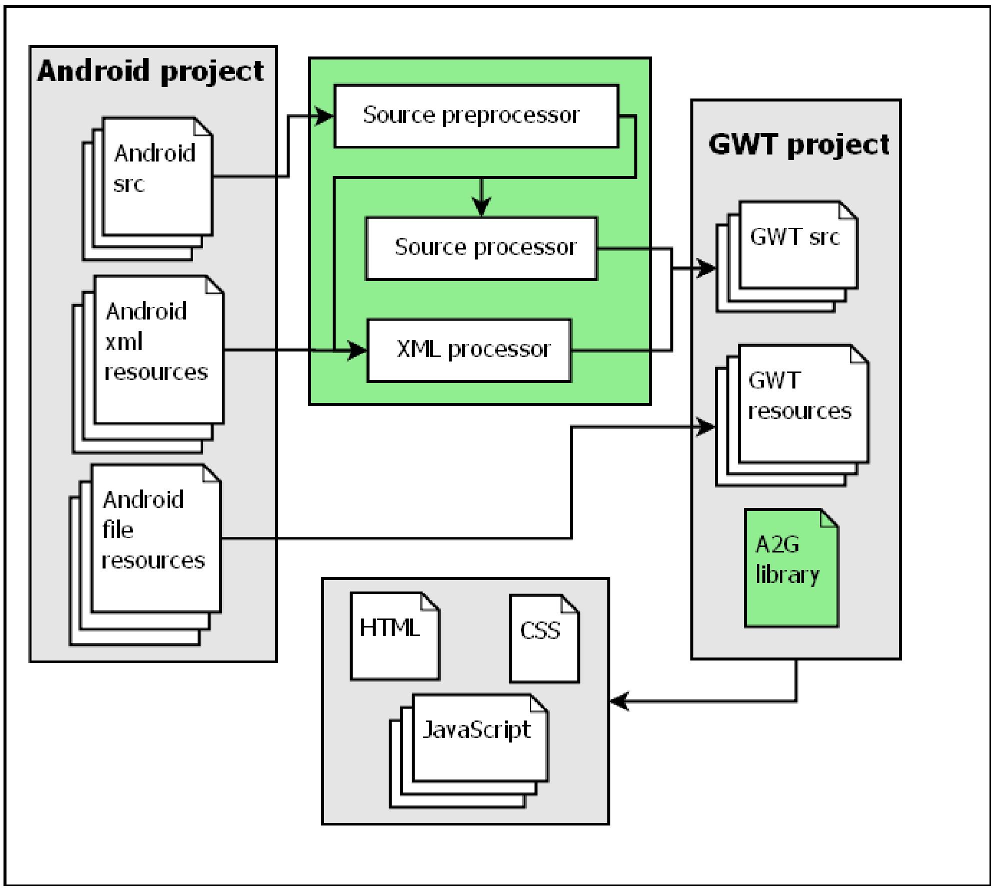


Figure 5.1: Conversion basic scheme

Using described above principles GWT module called *a2g* and conversion tool called *A2GConverter* were developed. *A2GConverter* uses principle is to make as few changes in Java code as possible - compatibility with GWT achieved by *a2g* module which wraps GWT components into Android-like structures.

5.2 Converter working principle

Basically converter consist of two independent parts:

* *a2g* GWT module - jar library, which needs to be attached to output GWT project.

* *A2GConverter* - java program, which makes necessary modifications in Android source and resource files.

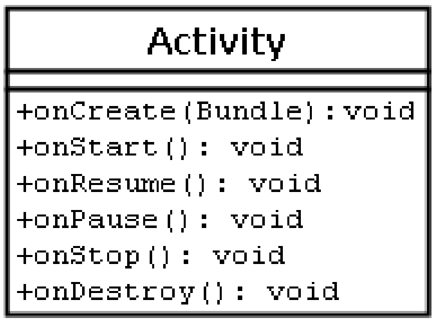


Figure 5.2: Android *Activity*

The conversion requires empty GWT project with included *a2g* module in project dependencies. Path to this project, as well as path to input Android project are two required input parameters for converter. After performing conversion using *A2GConverter*, it is possible to make changes in GWT output project before compiling it to JavaScript.

Figure 5.1 illustrates converter basic component and work principle. In next sections each converter component will be considered in details. *A2GConvertor* constructs abstract syntax trees (AST) based on Java code, makes changes in them and writes new Java files to GWT project. For construction AST used *JavaParser* and *JavaLexer* generated by ANTLR. Java grammar used for constructing parser and lexer is modified Terence Parr Java 1.5 grammar (20).

5.3 GWT module

The main purpose of *a2g* GWT module is to reduce code modification amount. It acts like intermediate layer between GWT core components and Android source. It emulates Android application behaviour using GWT components or fills missing functionality by stub methods.

For example, one of the Android application entry points is *Activity*. *Activity* is created by subclassing *Activity* class and extending *onCreate*. There is several other lifecycle important methods in *Activity* class which developer may override see figure 5.2.

GWT module entry point must implement interface *EntryPoint*, which have only one method *onModuleLoad*(see Figure 5.3)

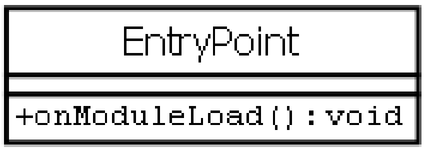
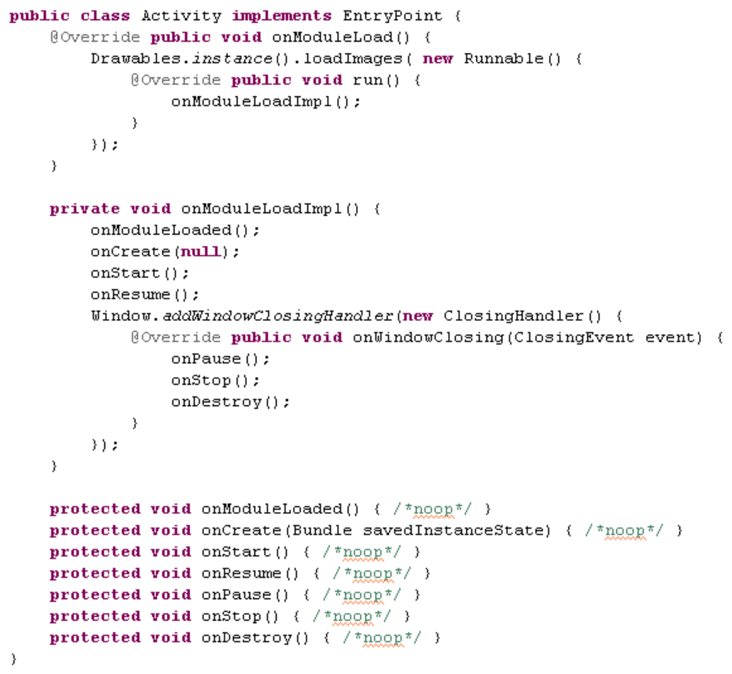


Figure 5.3: GWT *EntryPoint*



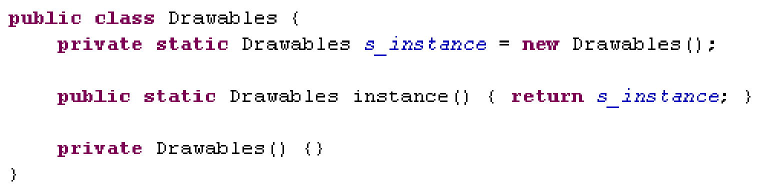
```
public class Activity implements EntryPoint {
    @Override public void onModuleLoad() {
        Drawables.instance().loadImages( new Runnable() {
            @Override public void run() {
                onModuleLoadImpl();
            }
        });
    }

    private void onModuleLoadImpl() {
        onModuleLoaded();
        onCreate(null);
        onStart();
        onResume();
        Window.addWindowClosingHandler(new ClosingHandler() {
            @Override public void onWindowClosing(ClosingEvent event) {
                onPause();
                onStop();
                onDestroy();
            }
        });
    }

    protected void onModuleLoaded() { /*noop*/ }
    protected void onCreate(Bundle savedInstanceState) { /*noop*/ }
    protected void onStart() { /*noop*/ }
    protected void onResume() { /*noop*/ }
    protected void onPause() { /*noop*/ }
    protected void onStop() { /*noop*/ }
    protected void onDestroy() { /*noop*/ }
}
```

Figure 5.4: *a2g* library *Activity* class

To avoid heavy changes in Android project Activity during conversion to GWT *a2g* library have *Activity* class, which implements *EntryPoint* and defines methods shown in Figure 5.2 . The *a2g Activity* class preserves methods call sequence and tries to emulate Android *Activity* behaviour. After images loaded (this part will be described later), *Activity* calls *onCreate*, *onStart* and *onResume* methods in the same sequence Android *Activity* does.The code example see in Figure 5.4.

Another functionality provided by a2g module is static access to resources. It is achieved by using singletons Strings, Drawables and Layouts. Those classes use standard singleton pattern - declaring private constructor, to prevent initialization from other classes and initializing static member of this class type (See Figure 5.5)

The resources providers do not have access to resources, so they should be set



```
public class Drawables {
    private static Drawables s_instance = new Drawables();

    public static Drawables instance() { return s_instance; }

    private Drawables() {}
}
```

Figure 5.5: Singleton pattern

externally on application start.

There is one limitation with HTML and correspondingly with GWT concerning images. Image should be loaded before it can be used on canvas. The solution is to add invisible Image to document - to initiate loading and using LoadHandler wait till image is loaded. As shown in Figure 5.4, actual module loading is done after images are loaded.

5.4 Converter

Converter consist of three parts

* Source preprocessor - its responsibility is to create class dependencies map - class full name with its package and classes it derived from. Also it finds application entry point - required for GWT module descriptor.
* XML processor - based on XML resources like strings and layouts generates corresponding Java code. This code is written to *CoreInitilizer* class, which is responsible for setting those resources mapping to corresponding *a2g* library classes
* Source processor - makes code changes.

5.4.1 ANTLR grammar

Both source preprocessor and processor work is based on Abstract Syntax Tree. As mentioned earlier, ANTLR and slightly modified Terence Parr Java grammar(20) used for this purpose. The original grammar constructs flat AST based on source file i.e. every tree node have root node as a parent. Grammar modification was aimed to make some Java constructions accessible by one root node, to simplify processor and preprocessor work in finding them. For example code "package com.example;" produces

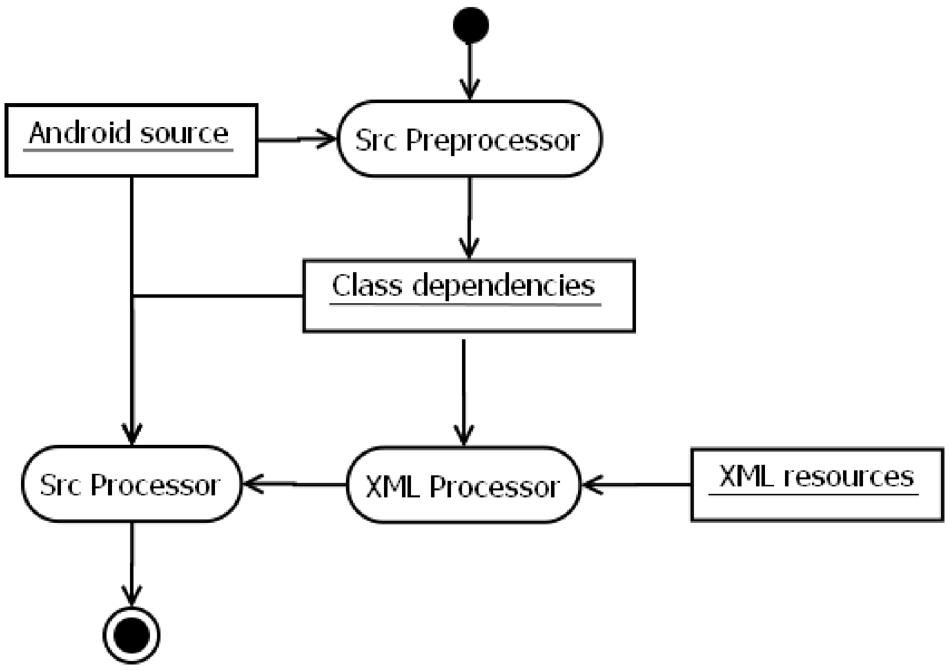


Figure 5.6: *A2GConvertor* object flow diagram

tree shown on Figure 5.7 using unmodified grammar. Change grammar produces tree show on figure 5.8 for same part of code.

The grammar changes affect only AST construction, not source parsing. Changed grammar parts used approach shown above and includes following constructions:

* Package keyword
* Import declarations
* Extends and implements keywords
* Class declaration
* Class body
* Constructors
* Type modifiers
* Method declarations

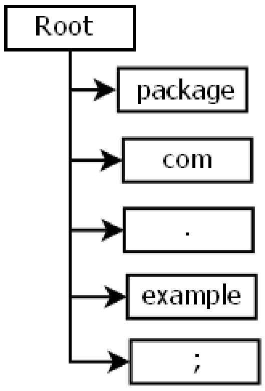


Figure 5.7: Flat tree

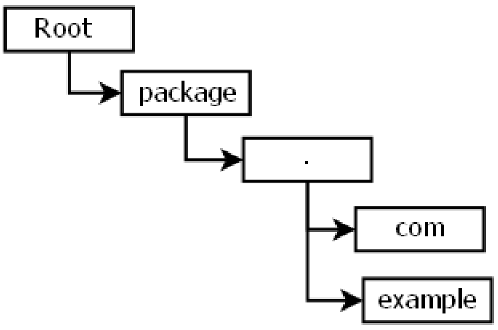
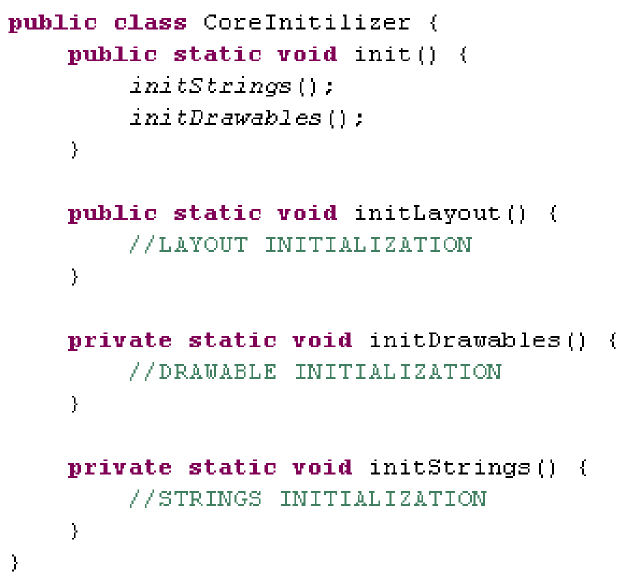


Figure 5.8: Normal tree



```
public class CoreInitilizer {
    public static void init() {
        initStrings();
        initDrawables();
    }

    public static void initLayout() {
        //LAYOUT INITIALIZATION
    }

    private static void initDrawables() {
        //DRAWABLE INITIALIZATION
    }

    private static void initStrings() {
        //STRINGS INITIALIZATION
    }
}
```

Figure 5.9: *CoreInitilizer* template

5.4.2 Source preprocessor

The single task of this component is to generate class hierarchy. This information is later used by both XML and source processors. Preprocessor uses package and class declaration nodes to obtain full class name. Base class and interfaces info is constructed using import, implements and extends nodes.

5.4.3 XML processor

To preserve resource access possibility by its identificator defined in *R* file, Android resource XML files are converted to Java code. In other words it fills *CoreInitilizer* class corresponding sections.

Android layout files may use custom views from the project source. To let XML processor know about those custom views, class hierarchy, made by source preprocessor is used.

5.4.4 Source processor

Despite the fact that major work is done in a2g library, there are still cases that require code changes. First of all *a2g* library resource wrappers Stings, Drawables and Layouts

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <com.example.android.lunarlander.LunarView
      android:id="@+id/lunar"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"/>

</FrameLayout>
```

Figure 5.10: Custom view example

```
@Override public void onModuleLoad() {
    CoreInitilizer.init();
    super.onModuleLoad();
}

@Override public void onModuleLoaded() {
    CoreInitilizer.initLayout();
}
```

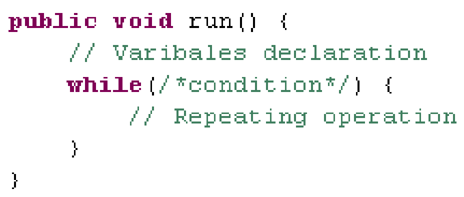
Figure 5.11: *CoreInitilizer* usage

need to be initialized with resources on application start. To get this done, processor finds entry point and adds code lines shown in figure 5.11

This invokes *Drawables* and *Strings* initialization(see also figure 5.11 and figure 5.4) and after all images are loaded - Layouts initialization.

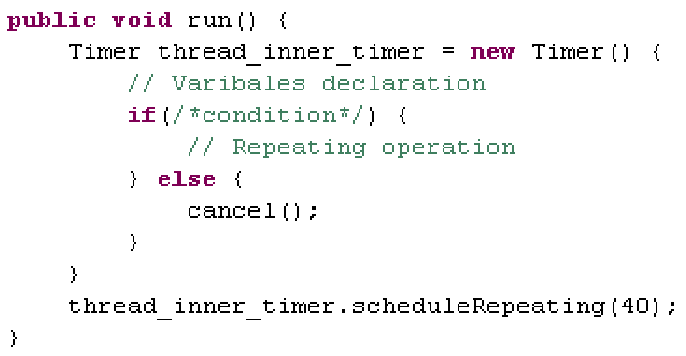
Second thing is custom view constructor. Android View can have AttributeSet as parameter. This is collection of attributes, found in XML layout and associated with this custom view element. Current implementation of XML processor does not support attributes processing, so this parameter is omitted from constructor.

Third thing is related to multithreading - in browser all JavaScript runs in single thread. Multithreading can be simulated in browser by using asynchronous timers. Same thing apply to GWT - it does not support threads, however it has possibility emulate it using Timer. From the converter point of view, cycle inside Thread run method should be replaced with scheduled *Timer* runs. Current implementation of Source processor supports only thread which run method structure show on figure 5.12. It is converted to code shown on figure 5.13.



```
public void run() {
    // Varibales declaration
    while(/*condition*/) {
        // Repeating operation
    }
}
```

Figure 5.12: *run* method before conversion



```
public void run() {
    Timer thread_inner_timer = new Timer() {
        // Varibales declaration
        if(/*condition*/) {
            // Repeating operation
        } else {
            cancel();
        }
    }
    thread_inner_timer.scheduleRepeating(40);
}
```

Figure 5.13: *run* method after conversion

5.4.5 Limitations

Because current software is only prototype, limitations of approach and used tools will be described and discussed in this section. First of all GWT supports very limited amount of HTML5 features. They are: web storage canvas element <*audio*> and <*video*> elements There also several HTML5 features which support is added by external libraries such as: geolocation databases
Secondly, browser capabilities put several restrictions due the fact that native to pure HTML5 conversion is used. This includes such features as:

* Connectivity detection

* Different sensors usage

* Sending and receiving system notifications

* Camera usage

* Loading different resources for different device configurations

Third serious limitation is multithreading. Applications that use connection with server to send and receive data asynchronously, process it locally and then display to user, often have at least three more threads beside application main thread. Those are Sender - for sending request to server, Receiver - for receiving data from server and Worker- for processing the data. Emulating all those threads using *Timer* is not a right solution - there is special API in HTML5 called WebSockets to handle such cases.

Another limitation is 3D graphics. Despite the fact that there is 3D graphics support for canvas element called WebGL and moreover, that there is external library for GWT that support it, such class of application not worth to convert. They highly rely on GPU and hardware acceleration, and additional runtime layer - browser will significantly decrease performance.

5.4.6 Case studies and evaluation analysis

As a basis for conversion it was decided to choose open source application that after conversion will use at least one of HTML5 features. LunarLander game was chosen as a candidate. It is simple game available under Android SDK samples. The game is classical LunarLander implementation for Android and its objective is to land on the moon. It demonstrates following Android application features:

* Loading and drawing resources on canvas
* Listening key inputs
* Animating using separate thread
* Choosing different resources for different screen orientation

It fits perfectly for conversion, and source code was taken from Android 2.1 samples. The HTML5 feature converted application should use is canvas element. The different resource loading feature is not supported by *A2GConvertor*, so it simply loads default resources.

Due the fact that only prototype was developed, *A2GConvertor* convertible possibilities are limited to described above features. The conversion was successful, and converted application shows expected result. As described earlier, Threads were replaced by Timers during conversion, and animation Timer was scheduled to show 25

MOS	Performance	
5	Excellent	No visible delays in the movement and animations
4	Good	Minimal visible delays in movement and animations
3	Fair	Movement and animations are rough and not smooth
2	Poor	Movement and animations are discontinuous
1	Bad	Game is unplayable

Figure 5.14: Mean option score

Browser	20 FPS	25 FPS	30 FPS
Internet Explorer 8	3	4	5
Firefox 12	3	5	5
Google Chrome 18	3	4	5

Figure 5.15: Visual analisys of the game performance

frames per seconds (i.e. calls *Timer run()* method once per 40 milliseconds). It also makes canvas size constant 320 x 400.

The application was tested with following configurations:

* AMD Turion 64 x 2 (1.6 GHz), Linux Debian, IceWeasel browser

* Core 2 duo (2.66 GHz), Firefox 12, Internet Explorer 9 and Google Chrome 18.0

To analyse the game visual performance in different browsers Mean Option Score was uses 5.14. The game performance was measured using different frames per seconds settings in animation timer. The higher frames per seconds is, the smoother are animations but it can decrease user interaction with application.

The analisys was performed wiht three browsers on Core 2 duo platform. The results are shown in figure 5.15. There are visual performance decrease for low FPS, however increasing FPS improves situation and does not have influence on user interaction.

5.5 Summary

Coversion performs as few code changes as possible. This is achieved by wrapping GWT components in Adroid structures. There are basically three things that require heavy code changes:

* XML resources conversion to Java code
* Handling and loading images
* *Thread* conversion to *Timer*

6

Conclusions and Future Research Directions

Mobile domain rapid evolution and fragmentation across devices have generated need for cross-platform development tools and technologies. Google and Apple - two leading mobile operating system providers are competing each other for the first place. It takes a lot of effort for developer to migrate application from Android to iOs and vice versa.

Vise solution could be migrating to cross-platform base and to gain support of smaller platforms such as Blackberry OS, Windows Mobile as a free benefit.

Nowadays, cross-platform development tools are mostly using HTML and JavaScript as a primary development technology, so converting native application to HTML5 could be a good place to start with cross-platform development.

As mentioned earlier not every application can be converted and moreover, there is no point in converting several classes of applications. Also automatic conversion should not always show good result from performance point of view. Converting native application to intermediate format and tuning it if needed can be a useful feature.

The current thesis compares HTML5 and native applications, defines application scope which can be converted to HTML5. It introduces several cross-platform development tools, such as PhoneGap and Titanium Appcelerator. It also gives suggestion and shows possibility for intermediate format such as GWT. One of the future research directions could be integration this tool with Eclipse. Another is to investigate and improve *Thread* emulation using *Timer*.

7

Resümee

Natiivsete Android rakenduste migratsioon HTML5-ks

Tänapäeval on märgata mobiilseadmete ning mobiilrakenduste turu kiiret kasvu. Mobiiltarkvara arenduse protsessi alustamisel tihti kerkib esimesena platvormi valiku küsimus, mis tavaliselt põhineb kasutajate nõudluse statistikal. See omakorda tekitab piiranguid rakenduste kasutamisel. Teoreetiliselt antud probleemi lahenduseks sobiks tehnoloogia, mis võimaldab kirjutada mitmel platvormil funktsioneerivaid rakendusi. Selleks esineb mitmeid meetodeid, kuid igaühel on tõsiseid puudujääke.

Mobiilseadmetele mõeldud Flash ei arendata enam Adobe poolt, puhas web'i rakendus vajab ühendust Internetiga ning seal ei ole võimalik kasutada natiivseid seadme omadusi.

HTML5 on uus tehnoloogia, mis toetab mitmeplatvormilist lähenemist, vimaldab töötada võrguväliselt ning kasutada seadme mitmeid natiivseid omadusi.

Selle magistritöö eesmärgiks on uurida võimalust konverteerida Android natiivset tarkvara HTML5-ks ning arendada rakenduse prototüüpi, mis teeks antud operatsiooni võimalikuks.

Kindlasti esineb teatud rakenduste hulk, mis kasutavad Android platvormi spetsiifilisi omadusi, mis ei allu konverteerimiseks, kuna HTML5 neid ei toeta. Sellest lähtuvalt eimesena tuleks teha kindlaks, millised programmid sobivad konverteerimiseks. Järgmine samm oleks uurida vahendeid mida kasutada eesmärgi saavutamiseks. Viimasena on konverteri prototüüpi disain ja arendus.

Bibliography

[1] Gartner Inc, Gartner Smart Phone Marketshare 2011 Q4, http://www.gartner.com/it/page.jsp?id=1924314. 1, 39

[2] Google Inc, Android, http://www.android.com/. 2

[3] Java2Script, Java2Script project page, http://j2s.sourceforge.net. 3

[4] Google Inc, GWT project page, http://code.google.com/intl/en/webtoolkit. 3

[5] ANTLR, ANTLR project homepage, http://www.antlr.org/. 3

[6] S. B. Damon Oehlman, Pro Android Web Apps, Apress, 2011. 5

[7] W3C, Html5 differences from html4, http://dev.w3.org/html5/html4-differences. 6

[8] J. J. Elayidom, A quick look at browser engines (trident, gecko, webkit, presto), http://getjins.wordpress.com/2008/09/10/a-quick-look-at-browser-engines-trident-gecko-webkit-presto/. 8

[9] WebKit, WebKit project homepage, http://trac.webkit.org/wiki/Applications 8

[10] B. L. Andre Charland, Mobile application development: Web vs. native, Communication of The ACM 54 (2011) 49–53. 11

[11] S. Mansfield-Devine, Divide and conquer: the threats posed by hybrid apps and html5, Network Sequrity 3 (2010) 4–6. 11

[12] Google Inc, HTML5Rocks information resource, http://www.html5rocks.com/. 12

[13] D. Oehlman, Mobile web app native bridging frameworks, www.distractable.net/coding/mobile-webapp-native-bridging-framework. 12

[14] Adobe Systems Inc, PhoneGap, http://phonegap.com/. 12

[15] Appcelerator Inc, Titanium appcelerator, http://www.appcelerator.com/platform. 13

[16] Z. Renjian, Inside Java2Script, http://inside.java2script.com/. 13

[17] The Eclipse Foundation, SWT project page, http://www.eclipse.org/swt/. 13

[18] J. Labanca, Gwt + html5: A web developers dream!, in: Google I/O 2011, 2011, http://www.google.com/events/io/2011/sessions/gwt-html5-a-web-developers-dream.html. 14

[19] R. M. Volkmann, ANTLR 3, Java News Brief. 14

[20] T. Parr, Java 1.5 grammar for ANTLR v3, http://antlr.org/grammar/1152141644268/Java.g. 21, 23

[21] Gartner Inc, Gartner Smart Phone Marketshare 2009 Q1, http://www.gartner.com/it/page.jsp?id=985912. 39

[22] Gartner Inc, Gartner Smart Phone Marketshare 2009 Q2, http://www.gartner.com/it/page.jsp?id=1126812. 39

[23] Gartner Inc, Gartner Smart Phone Marketshare 2009 Q3, http://www.gartner.com/it/page.jsp?id=1224645. 39

[24] Gartner Inc, Gartner Smart Phone Marketshare 2009 Q4, http://www.gartner.com/it/page.jsp?id=1306513. 39

[25] Gartner Inc, Gartner Smart Phone Marketshare 2010 Q1, http://www.gartner.com/it/page.jsp?id=1372013. 39

[26] Gartner Inc, Gartner Smart Phone Marketshare 2010 Q2, http://www.gartner.com/it/page.jsp?id=1421013. 39

[27] Gartner Inc, Gartner Smart Phone Marketshare 2010 Q3, http://www.gartner.com/it/page.jsp?id=1466313. 39

[28] Gartner Inc, Gartner Smart Phone Marketshare 2010 Q4, http://www.gartner.com/it/page.jsp?id=1543014. 39

[29] Gartner Inc, Gartner Smart Phone Marketshare 2011 Q1, http://www.gartner.com/it/page.jsp?id=1689814. 39

[30] Gartner Inc, Gartner Smart Phone Marketshare 2011 Q2, http://www.gartner.com/it/page.jsp?id=1764714. 39

[31] Gartner Inc, Gartner Smart Phone Marketshare 2011 Q3, http://www.gartner.com/it/page.jsp?id=1848514. 39

8

Appendices

8.1 Appenidx 1. Worldwide mobile Os marketshare

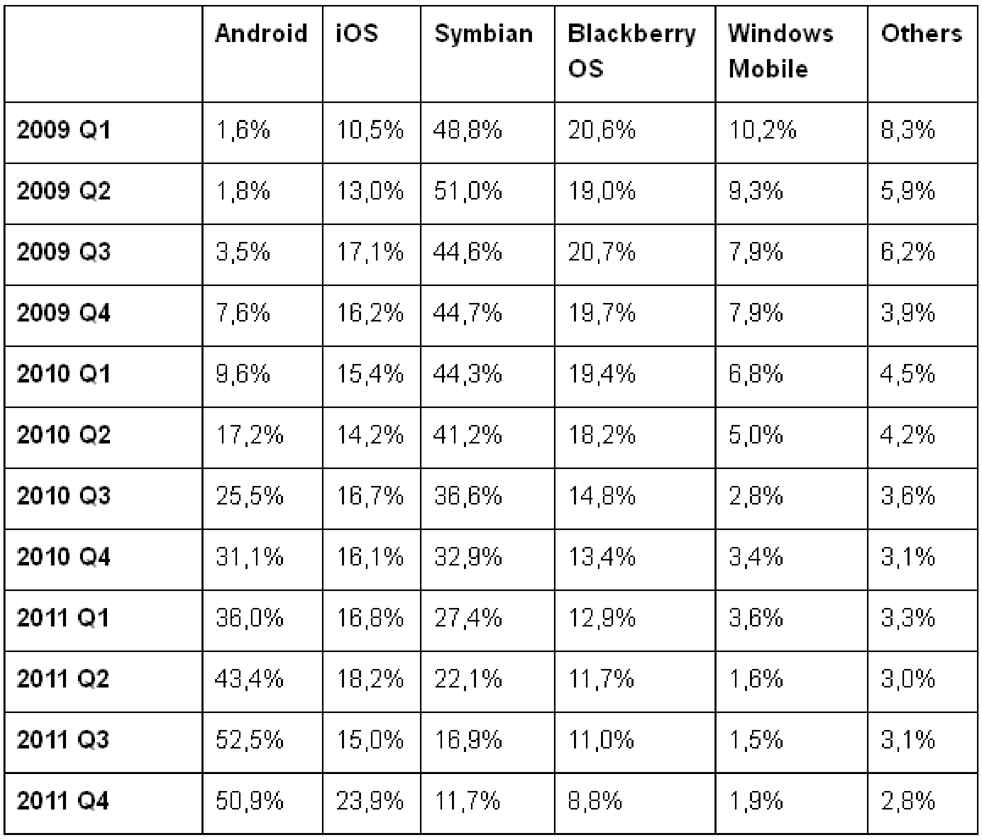
Table is created based on Gartner Inc quarter reports(21), (22), (23), (24), (25), (26), (27), (28), (29), (30), (31), (1).

8.2 Appendix 2. Extended Backus-Naur Form

Extended Backus-Naur Form (EBNF) is a family of metasyntax notations. It represents formal way to describe formal languages including computer programming languages. EBNF consists of : terminal symbols - literal characters from production rules of formal grammar that can not be changed using the grammar rules. In other words they are language elementary symbols defined by the grammar non-terminal production rules - strings composed of terminal symbols and non-terminal rules

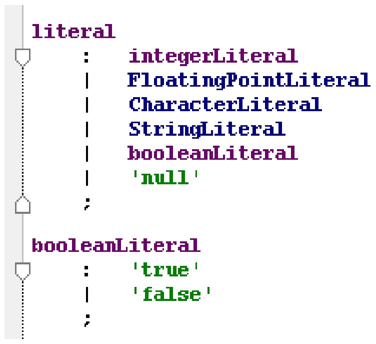
The following simple grammar rules are from Terence Parr Java grammar: Rules to the right of the colon is non-terminal rules. They get replaced by their definition (left of the colon) until only terminal symbols remains.

Another good example of EBNF is taken from same grammar that represents recursion:



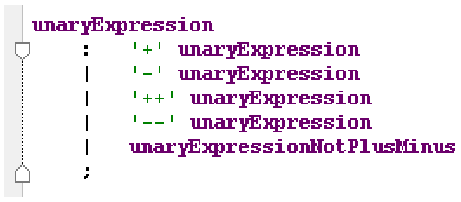
	Android	iOS	Symbian	Blackberry OS	Windows Mobile	Others
2009 Q1	1,6%	10,5%	48,8%	20,6%	10,2%	8,3%
2009 Q2	1,8%	13,0%	51,0%	19,0%	9,3%	5,9%
2009 Q3	3,5%	17,1%	44,6%	20,7%	7,9%	6,2%
2009 Q4	7,6%	16,2%	44,7%	19,7%	7,9%	3,9%
2010 Q1	9,6%	15,4%	44,3%	19,4%	6,8%	4,5%
2010 Q2	17,2%	14,2%	41,2%	18,2%	5,0%	4,2%
2010 Q3	25,5%	16,7%	36,6%	14,8%	2,8%	3,6%
2010 Q4	31,1%	16,1%	32,9%	13,4%	3,4%	3,1%
2011 Q1	36,0%	16,8%	27,4%	12,9%	3,6%	3,3%
2011 Q2	43,4%	18,2%	22,1%	11,7%	1,6%	3,0%
2011 Q3	52,5%	15,0%	16,9%	11,0%	1,5%	3,1%
2011 Q4	50,9%	23,9%	11,7%	8,8%	1,9%	2,8%

Figure 8.1: Mobile Os marketshare



```
literal
    :   integerLiteral
    |   FloatingPointLiteral
    |   CharacterLiteral
    |   StringLiteral
    |   booleanLiteral
    |   'null'
    ;

booleanLiteral
    :   'true'
    |   'false'
    ;
```

Figure 8.2: EBNF rules



```
unaryExpression
    :   '+' unaryExpression
    |   '-' unaryExpression
    |   '++' unaryExpression
    |   '--' unaryExpression
    |   unaryExpressionNotPlusMinus
    ;
```

Figure 8.3: EBNF recursion