Native Cross-platform Mobile Application Development Using Voind

Version of November 29, 2011

Mathieu Bruning

I would rather write programs to help me write programs than write programs.
–Dick Sites

Native Cross-platform Mobile Application Development Using Voind

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Mathieu Bruning
born in Naaldwijk, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`www.ewi.tudelft.nl`

Grenos BV
Ambachtstraat 6, 2671 CN
Naaldwijk, the Netherlands
`www.grenos.com`

Native Cross-platform Mobile Application Development Using Voind

Author: Mathieu Bruning
Student id: 1280740
Email: `m.bruning@grenos.com`

Abstract

The fragmentation of mobile devices nowadays makes it hard for (third party) mobile application developers to develop applications that are suitable for all mobile platforms on the market. A current startup, named Obymobi (Order By Mobile), introduces a new platform for ordering drinks and/or food in the hospitality business using a mobile application. The Obymobi mobile application is targeted at all mobile platforms, so that every mobile user is able to use the application. However, developing a mobile application which can be executed at all current mobile platforms implies several portability issues.

This thesis is the result of research which focuses on how to deal with the heterogeneity in mobile devices while developing mobile applications. As current cross-platform mobile application development solutions from industry do not offer flexibility as they tend to have a static character in terms of supported platforms and features, a new cross-platform development framework prototype is presented in this thesis. Using this framework named Voind, mobile applications can be defined in an abstracted model in order to use this model for the generation of multiple versions for different mobile platforms.

Preface

This thesis consists of my findings in cross-platform mobile application development during this research period.

I would like to thank Martin Pinzger for his insights and coaching during my thesis project. I would also like to thank him for his great deal of patience when the project did not seem to progress that fast due to activities for the Obymobi startup.

Finally, thanks to my colleagues at Obymobi for their insights on cross-platform mobile application development and their support during this graduation phase.

Mathieu Bruning
Delft, the Netherlands
November 29, 2011

Contents

List of Figures

Chapter 1

Introduction

The diverging smartphone market is one of the most dynamic and competitive in the consumer electronics industry. The latest trend is the upcoming of smartphones which, in addition to traditional voice communication and messaging functionality, provide web access, personal information management, multimedia and business applications [37].

A smartphone can be defined as a next-generation, multifunctional cell phone that provides voice communication and text-messaging capabilities and facilitates data processing as well as enhanced wireless connectivity [38].

However, currently there does not exist a de facto standard for mobile computing. The current vendors offer different mobile devices with different operating systems, which support different paradigms regarding the purpose and applicability of mobile computing.

Due to these differences in today's mobile platforms, little facilities to integrate with technologies of other platforms are offered. This has led to a highly cluttered market, where devices of different types are hardly able to work together in a useful way [24].

Even though there are sufficient similarities amongst most kinds of mobile devices, applications are mostly available just for a single platform. In addition, the possibilities of cross-platform data exchange are still moderate [17].

As stated in [24], relying on common features that are available on any of the platforms and standards inevitably implies both losing the flexibility to cater to specific features of the various devices as well as introducing a performance overhead, provided that the underlying hardware is indeed heterogeneous and the software performance is a crucial criterion.

1.1 Problem statement

As mentioned in [7], mobile applications must adhere to a strong portability requirement. This is due to the fact that service carriers typically demand from developers that a single application is deployed on a dozen or more platforms [7].

Portability is the number one problem when developing mobile applications for multiple platforms. This is because of several reasons, but the two most prominent ones are the diversity of mobile devices that exists in the market and the fact that performance is one of the top priorities when developing for mobile devices [13]. The major challenge of mobile application development is the heterogeneity of mobile devices and web browsers installed on the devices. The differences in the form factors and input capabilities strongly influence the usability of an application. In addition, the pre-installed browsers differ between the devices [31].

The heterogeneity of devices is causing an increased development effort for mobile applications. Maintaining multiple device dependent versions is labor intensive [31]. Therefore, mobile applications must adhere to a strong portability requirement [7].

Developers are frequently forced to develop different variations of a single application, optimized for different types of devices, operators, and languages. This further complicates the application development process, thus likely having a negative impact on the quality of the resulting software, because these variations usually involve modifications scattered across various artifacts. Accordingly, providing consistent maintenance of these variations becomes a more expensive and error-prone task, as the functional common core is normally dispersed accross such variations [7].

In industry, this problem is known as *fragmentation* and can be described as the inability to ”write once and run anywhere”, which often results in multiple versions of an application. The author of [30] defines fragmentation as the ”inability to develop an application against a reference operating context and achieve the intended behavior in all operating contexts suitable for the application”. He also defines an *operating context* (OC) for an application as the ”external environment that influences its operation”. Therefore an OC is defined by the hardware/software environment in the device, the user, and the environmental constraints introduced by various other stakeholders such as the network operator [30]. Although fragmentation can occur in the development of any kind of application, this report focuses on the fragmentation of mobile applications.

1.2 Research Questions

An overview of the research questions for the thesis is presented below. First, the main research question is presented which is followed by the sub research questions.

Main research question:

- *How to deal with mobile device heterogeneity during cross-platform mobile application development?*

Sub research questions:

- *What current solutions for cross-platform mobile application development exist?*
- *How can development effort be minimized while maintaining multiple versions of an application for different platforms?*
- *How to make use of device-specific features while developing a cross-platform application?*

The goal of this thesis project is to investigate how different mobile software platforms influence the application development process and what are the best practices in cross-platform mobile application development.

1.3 Contributions

The contributions of this thesis project are as follows:

- **Research on mobile platform** - Research has been conducted on the mobile platforms Android, Windows Phone 7 and Apple iOS to identify the similarities and differences between these platforms. In this phase, the potential issues in cross-platform development for these platform have been identified.
- **Development of XUL-based interface language** - A dynamic XUL-based language is developed to generically define mobile user-interfaces with. Using this language, mobile user-interfaces can be defined in an abstracted fashion which is used later to generate platform specific code for user-interfaces.
- **Development of Voind** - A web application has been developed which allows the user to define the artifacts of a mobile application which are used in the code generation process. The web application can also be used to maintain the supported mobile platforms with by persisting the platform specific components and mapping them to abstracted components.
- **Development of code generator** - A code generator has been developed which combines the persisted platform specific information with the abstracted artifacts of a mobile application to generate code for the mobile platform Android and Windows Phone 7.

1.4 Overview of chapters

The next chapter presents an introduction on Obymobi, a startup of a new mobile ordering platform for the hospitality business, which is used as a case study for cross-platform mobile application development. A theoretical background on fragmentation of mobile applications is presented in Chapter 3. Chapter 4 consists of an overview of three important mobile platforms, namely Android, Windows Phone and iPhone. An overview of the most promising cross-platform mobile application development framework from industry today is being presented in chapter 5. The requirements for a new cross-platform mobile application development solution is reflected in Chapter 6. Chapter 7 reflects on the design and development of the Voind application. An empirical evaluation of the developed application and the development process is presented in Chapter 8. Finally, Chapter 9 consists of the conclusions of the thesis project and presents future work related to the developed application.

Chapter 2

Obymobi Startup

This chapter introduces Obymobi, a startup consisting of a new mobile ordering platform for the hospitality business. First, an introduction of the Obymobi platform is presented. The second part of this chapter lists the requirements of the Obymobi application. How the Obymobi mobile application evolved is reflected in the last part of this chapter.

2.1 Introduction

In 2008, a startup named Grenos was founded with the idea of improving the quality time in the hospitality business by introducing self-ordering solutions. Obymobi (Order by mobile) is a new platform for ordering drinks and/or food in the hospitality business. Using a mobile application, customers are able to place orders without having to wait for a waiter. Figure 2.1 presents a visual overview of the Obymobi process. The first step in the process consists of the guest arriving at the hospitality business and wants to order something. Using a mobile phone, the guest is able to browse the menu of the hospitality business and compose an order. The composed order can then be sent to the Obymobi webserver using mobile internet. When the order is processed on the Obymobi webserver, it is being sent to a printer located at the hospitality business. The printer then prints out an overview of the placed order. Staffing personell can then prepare the order and serve it out to the guest.

In order to allow all mobile phone users to be able to use the Obymobi platform, it is necessary to investigate what the best practices are in dealing with mobile phone differentiation.

2.2 Requirements

Primary requirements:

- Obymobi should run on the most popular mobile platforms in order to cover a marketsegment as large as possible
- Obymobi development should follow the write-once, run everywhere paradigm

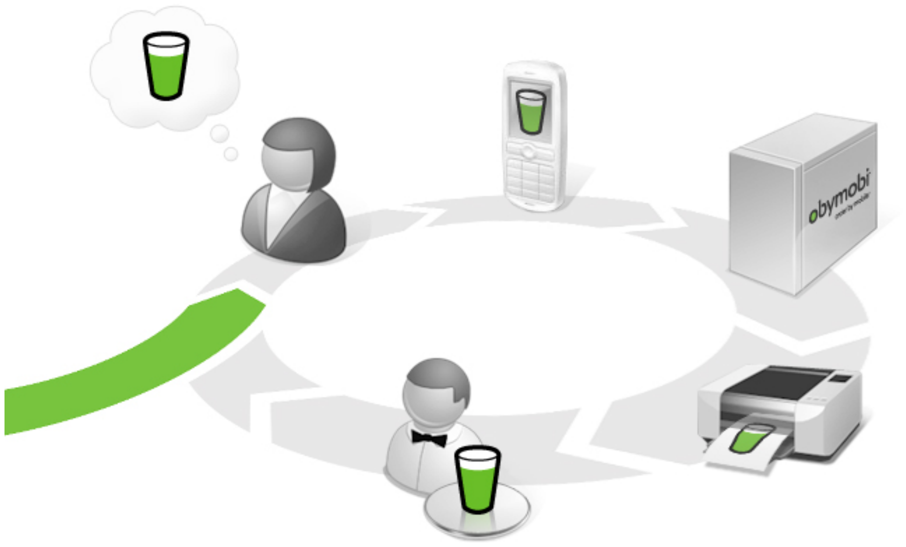

Figure 2.1: Obymobi process overview

Secondary requirements:

- Obymobi should be able to use device-specific features such as phonebook, GPS, etc.
- Obymobi should have a native look-and-feel

- **Installability** - Installing the Obymobi application should require minimal effort for end-users.
- **User input** - User input is a vital requirement for operating the application in order to interact with the end-user.
- **Connectivity** - In order to communicate with the central Obymobi backend, the application has to be able to connect to the Obymobi webservice using HTTP or SOAP.
- **Persistent storage** - The persistent storage deals with caching data retrieved from the central Obymobi back-end.
- **Security** - Communication with the central Obymobi back-end is secured using HTTPS and OpenSSL.
- **Performance** - Performance is an important requirement of the Obymobi application, because the application should allow users to place orders as quick as possible.

- **Usability** - One of the key aspects of the Obymobi application is usability. In order to have an optimal interaction between a mobile application and a user, an application should behave like its native environment. In this way, the user is operating in a familiar environment and is not distracted by an uncommon style [13].

2.3 Implementation

The architecture of the Obymobi ordering platform consists of a client-server architecture as presented in Figure 2.2. The server consists of a webserver which acts as the central backend. This server contains the database which holds all the data, the business logic for processing the data and a webservice which is used to communicate with the mobile clients.

The mobile client application consists of a network layer in order to communicate with the backend server. Some business logic is also included in the client application to handle the data retrieved from the server and to handle user input in the client application. The graphical user interface layer acts as the communication layer between the user and the application.

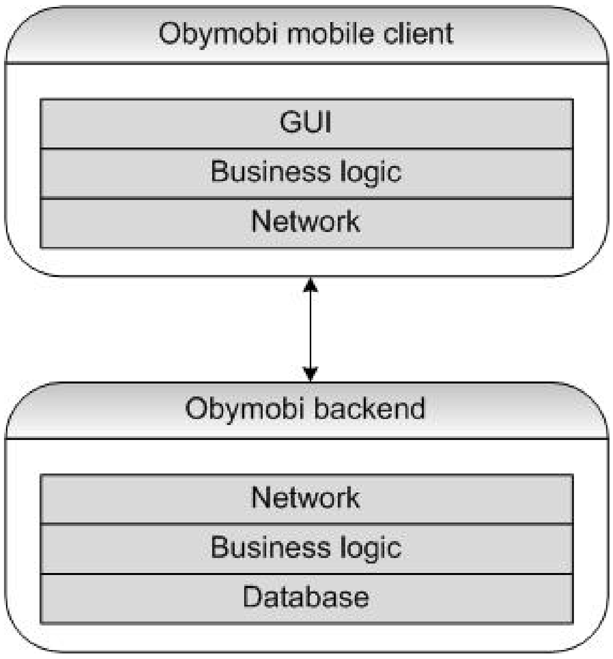

Figure 2.2: Obymobi architecture

2.3.1 Phase 1: J2ME application

The first prototype of the mobile client was implemented using the Java 2 Micro Edition (J2ME) platform. Although the J2ME platform pretends to be cross-platform, there are mobile platforms, like the Apple iOS platform, on which J2ME applications will not run.

This causes that a certain share of mobile phone users cannot use the Obymobi platform, because their mobile phones do not support the J2ME platform. Despite of the fact that other mobile phones did support J2ME, device configurations made it possible that our J2ME application was not fully compatible [34].

Although it followed the write-once, run everywhere paradigm, J2ME was not suitable because it did not cover significant mobile platforms as iPhone. Deployment issues with platforms that did support J2ME. Support for device-specific features was limited when developing using J2ME.

2.3.2 Phase 2: Mobile webapplication

In order to overcome this distribution problem, a webapplication was developed which mimics the functionality of the original Obymobi application. Having such a webapplication, all internet-enabled mobile phones consisting of a web browser were able to use the Obymobi application.

Although developing such a web application sounded like a fit solution for the distribution problem, slow performance was a consequence. Due to the slow speed of current mobile internet in general and the slow processing of JavaScript on mobile devices [18, 31, 33], the application's page loading times took up to seconds. This was quite a big difference compared to the J2ME user-interface which was developed earlier and could be executed on the device itself. As mentioned in [8], providing mobile data services via a web browser is easier to develop but the user interface of these services is in general poor as compared to Java-based applications. In addition, network traffic is considerably higher when using mobile Web pages, because not only raw data but also page formatting elements have to be transmitted. Another drawback of using a web application on a mobile device is that you cannot take advantage of advanced or platform-specific browser features on one device and expect it to work on another device with a less capable browser [6].

2.3.3 Phase 3: Native application

As the previous attempts of developing a cross-platform application using J2ME and a web application did not meet the 'write once, run everywhere' and performance requirements, the focus changed to developing native applications per mobile platform. Native applications have several advantages over virtualized or web application as they tend to have optimal performance and provide an application with a native look-and-feel. Another advantage is that device-specific features are supported as well when developing native applications. Although developing native applications per platform sounds like a fit solution for the current Obymobi requirements, this does not adhere to the 'write once, run everywhere' paradigm, as it requires programming knowledge of each of the implementation platforms.

Chapter 3

Theorical Background

The following chapter presents a theoretical background on fragmentation of mobile application which based on related work.

3.1 Introduction

As mentioned in [30], fragmentation is caused by the diversity of operating contexts (see Figure 3.1). One operating context may differ from another in the following ways:

- Hardware diversity
- Software diversity
  - Platform diversity
  - Implementation diversity
- Feature variations
- User-preference diversity
- Environmental diversity

This diversity of operating contexts often implies a problem for third party mobile application developers which target at multiple mobile platforms. For example, the development process requires additional development time, it requires effort to keep application for different platforms in sync and it requires additional knowledge of multiple device platforms.

In order to overcome this problem, several approaches of cross-platform development for mobile applications are proposed. Rajapakse [30] presents a taxonomy for de-fragmenting techniques for mobile applications. By analyzing the various aspects of fragmentation in mobile applications, an overview of techniques for de-fragmenting is presented. Using this taxonomy of de-fragmentation techniques, the cross-platform development approaches presented in this report can be classified. Figure 3.2 presents a visual overview of the de-fragmentation techniques presented.

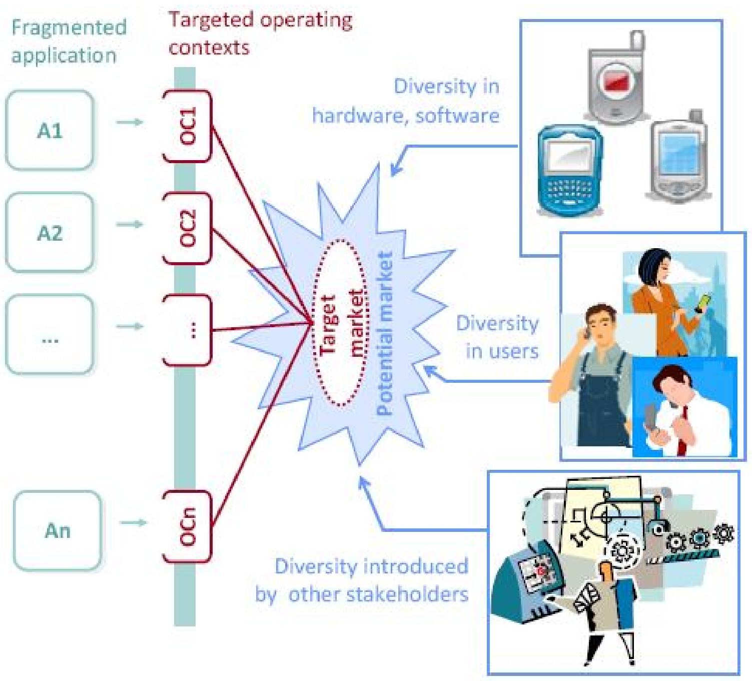
Fragmented application
Targeted operating contexts
A1
A2
...
An
OC1
OC2
...
OCn
Diversity in hardware, software
Target market
Potential market
Diversity in users
Diversity introduced by other stakeholders

Figure 3.1: Fragmentation overview. Taken from [30].

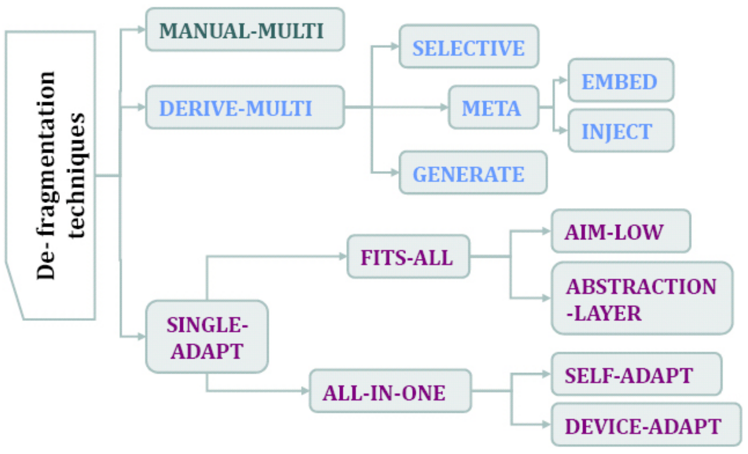

Figure 3.2: De-fragmentation techniques. Taken from [30].

3.2 Manual-multi

The Manual-multi appproach can be considered as the most primitive way of de-fragmenting mobile applications. It aims at manually developing multiple versions of an application to suit multiple operating contexts.

3.3 Derive-multi

The Derive-multi approach aims at developing multiple versions of an application from a single codebase. The Derive-multi approach can be divided into the subapproaches, namely Selective, Meta and Generate.

3.3.1 Selective

The Selective approach localizes variations into interchangable components and uses a build script to create a version for each operating context.

3.3.2 Meta

The Meta approach uses meta programming to specify how to derive operating context specific versions of an application. This approach can be divided into two subapproaches, namely: Embed and Inject.

Embed
The Embed approach uses preprocessing features to embed operating context specific variations into the source code of an application.

Inject
The Inject approach is based on writing operating context specific code which is separated from the application code. Using aspect-oriented concepts, the operating context specific code is then weaved into the common application code, in order to create a operating context specific version.

3.3.3 Generate

The Generate approach is based on automatically generating multiple versions of an application using a intelligent generator. Such a generator is then capable of knowing how to adapt a generic application to a operating context specific application.

3.4 Single-adapt

The Single-adapt approach is focused on creating a single version of an application that works on multiple operating contexts. This approach can be divided into two subapproaches, namely: Fits-all and All-in-one.

3.4.1 Fits-all

The Fits-all approach aims at creating a one-size-fits-all-application that fits all operating contexts. This approach can can be devided into two subapproaches, namely: Aim-low and Abstraction-layer.

Aim-low
The Aim-low approach focuses on creating a one-size-fits-all-application which uses only what is common to all operating contexts.

Abstraction-layer
The Abstraction-layer approach is concerned with hiding the variations of different operating contexts between an abstraction layer.

3.4.2 All-in-one

The All-in-one approach is focused on creating a single application which is able to adapt itself to operating context at runtime. The approach is divided into two subapproaches, namely: Self-adapt and Device-adapt.

Self-adapt
In the Self-adapt approach, an application programmatically discovers information about the operating context and adapts itself to that specific operating context.

Device-adapt
In the Device-adapt approach, the application is written in an abstract way, and the device decides how to adapt it to the prevailing operating context, at runtime.

3.5 Summary

The related work can be categorized into three major approaches in cross-platform mobile application development, namely: porting, virtualization and Generative Programming. The cross-platform development approaches mentioned above correspond to the de-fragmentation techniques mentioned earlier in this chapter. Therefore, a mapping of the cross-platform approaches to the de-fragmentation techniques is presented below.

In this report, (automated) porting is considered as the process of developing an application only once using a single codebase, and then automatically transferring the application into other exeuction environments. The de-fragmentation approach Derive-multi [30] has two subapproaches Selective and Meta, which correspond to the our definition of (automated) porting. This is because they focus on using build scripts or meta programming for creating device-specific versions, like the porting approaches from the literature do.

Cross-platform development using virtualization techniques corresponds with the Abstraction-layer subsubapproach. This approach is focused on hiding the variations of different execution environments between an abstraction layer in such a way that the development of an application can be realised by using abstract APIs. This corresponds with virtualization techniques in such a way that virtualization is also concerned with an abstraction layer over the underlying execution environment.

Generative Programming is concerned with the development of applications using higher-level abstractions, which are used by an code generator to derive instances of an application.

This corresponds with the de-fragmentation approach Generate, in which multiple versions of an application are automatically being generated using an intelligent generator [10, 32].

Chapter 4

Mobile Platforms

The following chapter reflects on the mobile platforms Android, iOS, and Windows Phone. Research has been conducted to make a comparison of these platforms which is used to identify the potential issues in cross-platform development for these platforms.

4.1 Introduction

In today's everyday life, ubiquitous computing has been integrated as a de facto standard. An example of such is mobile computing. Mobile computing has become popular over the last years as the use of smartphones has increased tremendously. As mentioned in the introduction of this thesis, the current smartphones offer little facilities to integrate with other technologies, which had led to a highly cluttered market. Smartphones are manufactured with different physical functionality. Therefore certain platform APIs may not be available on all smartphones. Each platform also depends on the features of the physical device to implement its APIs. Finally, each platform has its own development environment that support different sets of programming languages and APIs. The differentation in mobile platforms, also known as fragmentation [30] has caused the problem that there is no single platform or device that can do anything reliably [27]. Third party mobile application developers like Obymobi need to deal with mobile device fragmentation in order to overcome these differences in mobile devices. To get an overview of the differences in mobile devices, the several mobile platforms have to be compared.

Currently, the smart phone is dominated by five mobile platforms, namely: Android, Symbian, Apple iOS, Research in Motion (BlackBerry) and Microsoft Mobile / Phone. The following table represents an overview of the market penetration in the first quarter of 2011 of these platforms.

A comparison of three of these platforms (Android, Windows Phone and Apple iOS) will follow in the next sections of this thesis.

Platform	Sold units (x1000)	Market share (%)
Android	36,267.8	36
Symbian	27,598.5	27.4
Apple iOS	16,883.2	16.8
Research in Motion	13,004.0	12.9
Microsoft	3,658.7	3.6
Other platforms	3,357.2	3.3

Table 4.1: Market penetration per mobile platform. Source: Gartner (May 2011) [16]

4.2 Android

Android is an open-source smartphone platform which was originally created by Android Inc. and has been bought by Google in 2005. As presented in Table 4.1, the Android platform is currently market leader based on the sales of the first quarter of 2011.

4.2.1 Foundation

The foundation of Android is based on the Linux kernel and the Java programming platform using the Software Development Kit (SDK). However, a Native Development Kit (NDK) is also available which allows developers to write applications or extensions in C or C++ [25]. User-interfaces are usually defined using the eXtensible Markup Language (XML).

4.2.2 Learning Curve

The learning curve for the Android platform can be labeled as excellent as the documentation is very extensive as well as the corresponding Android community. As Android is founded on the Java Programming platform, a lot of documentation and a large community is already available.

4.2.3 Development

Android development is most likely done using the open-source IDE Eclipse as a plug-in is available which integrates the Android tools and emulator within Eclipse. However, command-line tools and other Java based integrated development environments like Netbeans can also be used.

4.2.4 Supported Platforms

Multiple platforms are supported as Android runs on several platforms using Java libraries controlling the different devices.

4.2.5 Installation

Applications developed for the Android platform can be distributed via the Android Market. Distribution of applications from outside the market is also allowed and can be done using the distribution of Android Package (APK) files.

4.2.6 Development Tool Cost

Tools for developing for Android are completely free as the SDK is free to download [21] and the most likely to use IDE Eclipse is also free for download. In order to distribute an Android application via the Android Market a fee of $25 is required.

4.2.7 Functionality

Accessing phone features or data on Android is possible by granting permissions to an Android application. A basic Android application does not have permissions to access any phone feature or data, so to gain access to the phone you will need to set those permissions.

An overview of the capabilities for Android can be found in Table 4.2.

Capability	Android
Graphical interface	2D, 3D graphics
Functionality	No restrictions
Phone Data Access	Full
Runtime Speed	Best

Table 4.2: Overview of Android capabilities

4.2.8 Performance

The runtime speed of the Android platform is determined by the Dalvik virtual machine based on the available memory and processor speed of the device to run the application on. By converting an application into the Dalvik Executable (DEX) format, an application is constrainted for the capabilities of a device. Although applications are optimized for different devices using this technique, they have to be interpreted into CPU instructions before execution instead of direct execution of compiled CPU instructions.

4.2.9 Developer Community and Support

The developer community and support for the Android platform can be rated as extensive as the Android platform is currently the largest smartphone platform in industry today. As the foundation of Android is based on Java, the extensive developer community and support from the Java platform is also applicable for Android.

An overview of the breadth for Android can be found in Table 4.3.

Feature	Android
Developer Community and Support	Extensive
Market Penetration	Market leader
Distribution and Licensing	Free via files or $25 for Android Market

Table 4.3: Overview of Android breadth

4.2.10 User-Interface

The Android platform offers support for various screen sizes and densities, so that the range of devices on which the Android platform can be executed is supported. By dividing the range of screen sizes into four generalized sizes (small, normal, large, xlarge) and four generalized densities (low, medium, high and extra high), designing a user-interface for multiple screens is simplified. The generalized sizes have the following minimum screen resolutions associated with it:

- small screens are at least 426x320
- normal screens are at least 470x320
- large screens are at least 640x480
- extra large screens are at least 960x720

The following screen resolutions are defined in the Android platform by default:

	Low density	Medium density	High density	Extra high density
Small screen	240x320		480x640	
Normal screen	240x400	320x480	480x800	640x960
	240x432		480x854	
			600x1024	
Large screen	480x800	480x800		
	480x854	480x854		
		600x1024		
Extra Large screen	1024x600	1280x800	1536x1152	2048x1536
		1024x768	1920x1152	2560x1536
		1280x768	1920x1200	2560x1600

Table 4.4: Android screen resolutions

Declaring a User-Interface in Android

A user-interface can be defined in Android using a XML layout file [25], which consists of nested XML elements in the same way like HTML pages are created. These XML elements represent View classes and their subclasses used for widgets and layouts. When an Android application is compiled, the layout XML files are being compiled in View resources which are loaded at runtime to apply a certain layout. Below is an example of an Android XML layout file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Listing 4.1: Example of an Android layout XML file.

Sizing of Views in Android can be done using constant relative values or using absolute values. The latter is however not recommended as specifying a user-interface in absolute values will cause an application not to display properly on different devices. Relative sizing is done using the following constant values:

- wrap_content, which causes the view to size according to its contents
- fill_parent, which causes the view to become as big as itś parent view would allow

4.2.11 Webservice Connectivity

In terms of webservice connectivity, Android does not provide native libaries for accessing web services with. There are third party libraries like kSOAP [23] of WSDL2Java [35], which allow you communicate with SOAP web services, however, these tools are not under active development anymore.

4.2.12 Persistent storage

Persistent storage is supported by Android in various ways [25], which are described below:

- Shared Preferences, which can be used to store private primitive data in key-value pairs
- Internal Storage, which can be used to store data on the device memory
- External Storage, which can be used to store data on the shared external storage
- SQLite Databases, which can be used to store structured data in a private database
- Network Connection, which can be used to store data on the web using a network server

4.3 Apple iOS

After Apple was rumouring on working on a mobile phone for a long time, the iPhone was introduced in 2007. The iPhone consists of a touchscreen smartphone which runs on iOS. The iPhone OS can be described as a scaled back version of the desktop Mac OS X.

Apple iOS currently holds the third place in the smartphone market with a 16.8% market share sales in Q1 2011 [16].

4.3.1 Foundation

The foundation of the iPhone OS is based on Objective-C, which can be used to develop native iPhone applications with. Objective-C is a superset of the C programming language, with additional features to support object-oriented programming. The language serves as the core of Mac OS X. Since it is based on ANSI C, Objective-C code can be integrated with C as well as C++ code [5].

4.3.2 Learning Curve

Developing for Apple iOS means you have to be familiar with Objective-C, a programming language which is used almost alone by Apple. Although Objective-C does ressemble the C++ language, it has a particular syntax which developers often find to have a steep learning curve.

4.3.3 Development

iOS applications are developed using XCode, an integrated development environment for Mac OS X. As XCode consists of an extensive suite of tools for Mac OS X application development, debuggers and an emulator are integrated.

4.3.4 Supported Platforms

In terms of cross-platform application development, applications developed using XCode are not able to run on other platforms as they only run on Apple iOS or Mac OS X

4.3.5 Installation

As mentioned in [22], the only way to sell an application to an iPhone user (at least one who hasn't jailbreaked the phone) is through the Apple AppStore on iTunes. Here Apple has control and won't let you sell anything which might compete with programs Apple has or plans to release. However, on 'jailbreaked' iPhones, applications can be installed using third party installer applications.

4.3.6 Development Tool Cost

The tools for developing iPhone applications are free, although obtaining a license for developing iPhone applications costs $99 to join the iPhone Developer Standard Program or

$299 to join the iPhone Developer Enterprise Program [20]. This allows a developer to develop and distribute an application for the iPhone.

4.3.7 Functionality

The functionality available for iPhone application developers is restricted as the source code of the iPhone OS cannot be accessed and the platform's functionality is only partially available through APIs.

Scanning for WiFi network or retrieving information about neighbouring cell towers is not allowed by the current iPhone SDK. Network interface selection is also not allowed. The state of the battery or the level of available RAM can also not be queried. Location sensing is supported using Global Positioning System (GPS) or cell tower/WiFi triangulation. Persistant storage is possible using file I/O and an integrated SQLite database. Secure HTTP connection using HTTPS and SSL is also supported by the iPhone platform.

Although it is possible to access data stored on the iPhone using APIs, it is rather limited to the contacts and media stored on the device.

An overview of the capabilities for iPhone OS can be found in Table 4.5.

Capability	iOS
Graphical interface	2D, 3D graphics
Functionality	Partial through API
Phone Data Access	Partial through API
Runtime Speed	Best

Table 4.5: Overview of iOS capabilities

4.3.8 Performance

The runtime speed or performance of program execution can be described as best as the Objective-C code can be compiled into direct ARM CPU instructions for the iPhone. These instructions can be executed on the underlying hardware directly which implies optimal performance.

4.3.9 Developer Community and Support

At first, developing applications for the iPhone was restricted in a way that all developers were required to sign a non-disclosure agreement which stated that it was not allowed to share information between iPhone application developers. Fortunately, this is no longer the case as Apple dropped the non-disclosure agreement for released iPhone software in 2008. After the non-disclosure agreement was dropped in 2008, application developers were able to share information about their experiences in iPhone development and the developer community started to grow. With the iPhone as one of the most popular mobile platforms nowadays, the developer community is growing fast and has become quite extensive already.

An overview of the breadth for iPhone OS can be found in Table 4.6.

Property	iOS
Developer Community and Support	Extensive
Market Penetration	Average
Distribution and Licensing	Requires Apple-issued certificate

Table 4.6: Overview of iOS breadth

4.3.10 User-Interface

Declaring a User-Interface in Apple iOS

User-interfaces for the Apple iOS can be specified using an application called Interface Builder, which is part of the Xcode. A shortened example of an iPhone layout file is presented below:

```
<?xml version=”1.0” encoding=”UTF–8”?>
<archive type=”com.apple.InterfaceBuilder3.CocoaTouch.XIB” version=”7.03”>
    <data>
...
        <object class=”NSMutableArray” key=”IBDocument.RootObjects” id=”1000”>
...
            <object class=”IBUIWindow” id=”380026005”>
                <reference key=”NSNextResponder”/>
                <int key=”NSvFlags”>1316</int>
...
                    <object class=”IBUIButton” id=”909449754”>
                        <reference key=”NSNextResponder” ref=”380026005”/>
                        <int key=”NSvFlags”>1316</int>
                        <string key=”NSFrame”>{{15, 74}, {285, 37}}</string>
                        <reference key=”NSSuperview” ref=”380026005”/>
                        <bool key=”IBUIOpaque”>NO</bool>
                        <bool key=”IBUIClearsContextBeforeDrawing”>NO</bool>
                        <int key=”IBUIContentHorizontalAlignment”>0</int>
                        <int key=”IBUIContentVerticalAlignment”>0</int>
                        <object class=”NSFont” key=”IBUIFont”>
                            <string key=”NSName”>Helvetica–Bold</string>
                            <double key=”NSSize”>1.500000e+01</double>
                            <int key=”NSfFlags”>16</int>
                        </object>
                        <int key=”IBUIButtonType”>1</int>
                        <string key=”IBUINormalTitle”>Zoek</string>
                        <object class="NSColor" key="
                            IBUIHighlightedTitleColor">
                            <int key="NSColorSpace">3</int>
                            <bytes key="NSWhite">MQA</bytes>
                        </object>
                        <object class="NSColor" key="IBUINormalTitleColor
                            ">
                            <int key="NSColorSpace">1</int>
                            <bytes key="NSRGB">
                                MC4xOTYwNzg0MyAwLjMwOTgwMzkzIDAuNTIxNTY4NjYAA
                                </bytes>
                        </object>
                        <object class="NSColor" key="
                            IBUINormalTitleShadowColor">
                            <int key="NSColorSpace">3</int>
                            <bytes key="NSWhite">MC41AA</bytes>
                        </object>
                    </object>
                </object>
...
        </object>
...
    </data>
</archive>
```

Listing 4.2: Shortened example of an iPhone layout XIB file.

The screen-resolution of the first three generation iPhones is 320x480 (HVGA), at 163 pixels-per-inch (PPI). The iPhone 4 has a screen resolution 640x960 at 326 PPI.

Relative sizing and relative positioning of controls is not possible on the iOS platform as all sizing and positioning is done in pixel values. It is not possible to specify values using percentages, which makes it hard to develop applications to different screen-resolutions.

4.3.11 Webservice Connectivity

Just like Android, Apple iOS does not offer native support for communicating with SOAP web services. In order to communicate with web service on the Apple iOS platform, we need to use third party libraries like WSDL2ObjC [36]. However, for this third party library counts that it is still under active development.

4.3.12 Persistent storage

Persistent storage is taken care of in iOS using the Core Data framework, which consists of a collection of tools to store, access and share data with [20, 5]. The Core Data framework provides the following options:

- SQLite, which can be used to store structured data in a private database
- Data sharing amongst apps
- Access of contacts and photos

- Calender access
- XML files
- HTML5 Client-Side Storage in Safari

4.4 Windows Phone

Microsoft Windows Phone is the successor to Windows Mobile, a mobile operating system which was introduced in 1996 and was targeted at devices with limited hardware resources available, like smartphones or embedded systems. Windows Phone is targeted at consumer devices only. With a share of 3.6% market sales in Q1 2011 [16], Windows Phone has a low market penetration.

4.4.1 Foundation

Developing applications for the Windows Phone platform can be done using the .NET language C#. In the latter case, the .NET Compact Framework is needed in order to run the applications [38].

4.4.2 Learning curve

As mentioned in [22], the technical documentation of the Windows Phone platform is excellent and there is an easy path for developers who have worked on a PC to move their application to Windows Phone.

4.4.3 Development

In order to develop applications for the Windows Phone platform, Microsoft's integrated development environment (IDE) Visual Studio .NET would be the first choice to use. Visual Studio .NET can be used to develop different kinds of smart-device applications, including Pocket PC, Smartphone and Windows CE applications. Visual Studio .NET consists of a suite of development tools, which also includes debuggers and emulators for the various types of projects.

4.4.4 Supported Platforms

In terms of cross-platform development, Windows Mobile was primarily limited to Windows Mobile and Windows CE. Nowadays its also possible to develop Windows Mobile applications for Android devices using Mono [26].

4.4.5 Installation

Installing a Windows Phone application on a device can be done using Over The Air (OTA) deployment, XAP files or the Windows Phone Market Place.

4.4.6 Development Tool Cost

Most tools for developing applications for Windows Phone applications are free, however, these are so-called 'Express' editions which have some limitations. The 'Standard' or 'Professional' editions of Visual Studio are not free. To submit an application to the Windows Phone Market Place, a fee of $19,99 per new application submission is charged for applications which are downloadable for free in the Market Place. Paid applications can be submitted unlimited at no charge. However, in order to promote the Windows Phone Market Place, Microsoft announced that the first 100 submissions are free each year for non-paid applications in the Market Place.

An overview of application development for the Windows Phone platform can be found in Table 4.9.

4.4.7 Functionality

The functionality offered by the Windows Phone platform is rather extensive, it offers facilities to scan for cellular, Bluetooth, and WiFi networks, establish connections on a specific network interface, enable and disable interfaces, determine the current location using GPS, and to run applications in the background [27]. Secure HTTP connections are supported using HTTPS and SSL. Because the Windows Phone operating system contains a virtual memory system, memory is not a concern for applications. Windows provides applications with standard file I/O facilities and an integral database engine SQL Server Compact Edition. Although the Windows Phone platform seems extensive in terms of functionality, the audio access is rather limited. In contrast to other platforms, changes to the battery status can be notified to applications. In this way, an application can react when the current battery status has changed.

Access to the data on a Windows Phone device is not limited, all data stored on the device can be accessed programmatically.

4.4.8 Performance

As Windows Phone is based on Microsoft .NET Compact Framework, the source code is compiled in an intermediate language, which has to be interpreted when it is executed. Therefore, it is much slower than executing the CPU-instructions directly.

An overview of the capabilities of the Windows Phone platform can be found in Table 4.7.

4.4.9 Developer Community and Support

In terms of developer community and support, Microsoft can be recognized as the best. It has the biggest community for application developers and offers the best conferences and

Feature	Windows Phone
Graphical interface	2D, 3D graphics
Functionality	Limited audio access
Phone Data Access	Full
Runtime Speed	Average

Table 4.7: Overview of Windows Phone capabilities

development tools [22].

An overview of the breadth of the Windows Phone platform can be found in Table 4.8.

Feature / Platforms	Windows Phone
Developer Community and Support	Extensive
Market Penetration	Low
Distribution and Licensing	Free via files or $19,99 per app submission to Market Place

Table 4.8: Overview of Windows Phone breadth

4.4.10 User-interface

User-interfaces on the Windows Phone platform are defined using the Extensive Application Markup Language (XAML), which is an XML based language.

An example of a Windows Phone XAML file is presented below:

```
<phone:PhoneApplicationPage
    x:Class="Obymobi.WindowsPhone7.SearchLocation"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=
        Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.
        Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006
        "
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot contains the root grid where all other page content is
        placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent" VerticalAlignment=
        "Top">
      <StackPanel Orientation="Vertical" x:Name="bTop" Width="auto"
          Height="768" >
        <TextBlock x:Name="text" Text="Zoek op naam of plaats" Width="
            auto"
        Height="38" ></TextBlock> <Button x:Name="button" Content="Zoeken
            " Width="auto" Height="76"
        Style="{StaticResource Button}" ></Button> </StackPanel>
    </Grid>

</phone:PhoneApplicationPage>
```

Listing 4.3: Example of a Windows Phone layout XAML file.

The following screen resolutions are initially supported by Windows Phone:

- 480x800 (WVGA)
- 320x480 (HVGA)

Positioning of controls is done by specifying margins in relation to the parent control.

The size of a layout element in Windows Phone is specified using one of the following values:

- *double*, which represents the size in pixels
- *qualifiedDouble*, which represents the size in pixels, inches, centimeter or points
- *Auto*, which enables autosize behavior

4.4.11 Web service Connectivity

Windows Phone applications can communicate with web services in two ways, namely via generated classes or via standard HTTP requests. Windows Phone offers integrated support voor web services, which enables developers to generate proxy classes for a web service in order to consume a web service. The other way is making use standard HTTP requests, although this is more complex as the response of the HTTP request has to be parsed.

4.4.12 Persistent storage

Windows Phone offers the following support for persistent storage in mobile appplications:

- SQL Server Compact Edition, for data storage
- IsolatedStorageSettings.ApplicationSettings for storing user preferences

4.5 Comparison

In order to identify the similarities and differences between the platforms Android, Apple iOS and Windows Phone, a comparison of these platforms is presented in this section. By comparing the characteristics of each of these platforms, the potential challenges in cross-platform development for these platforms are being identified.

Each of the compared platforms has a different foundation, where Android is based on Java, Windows Phone is based on C# and Apple iOS is based on Objective-C. The supported CPU-instruction set is the same for Android and Apple iOS, as they both support ARM CPU-instructions. The current version of Windows Phone only support x86 CPU-instructions, although Microsoft announced that Windows Phone 8 will support ARM instructions as well. A difference in application development on the platforms Android, Windows Phone and iPhone consists of the learning curve which is difficult for the iOS platform. This is related to the fact that Objective-C uses a syntax that is not like any other syntax and that youĺl have to do your own memory management. All of the compared platforms offer proper development environments which include debuggers and emulators. Development for Android is mostly done using Eclipse, which can be used to develop Android applications using the Android Development Tools (ADT) plugin. Developing applications for Windows Phone can be done using Microsoft Visual Studio, which is one of the best IDEs available in industry containing an extensive debugger and an emulator for Windows Phone applications. XCode is used to develop iOS applications with, which is bundled with an iPhone emulator and has a debugger integrated. In terms of cross-platform development, all of the compared platforms can only be used to develop applications for that certain platform with. Deploying applications for Android is done using the Android Market or distribution via APK files. Apple iOS applications are deployed using the AppStore. Windows Phone applications can be deployed using Over-The-Air (OTA) deployment, XAP file deployment or the Windows Phone Market Place. Opposed to Apple iOS, Android and Windows Phone offer the possibility to deploy applications using installation files, where iOS applications can only be deployed using the AppStore. Development tools are free of costs when developing for Android, as Eclipse and the ADT plugin are both free of costs. Developing applications for Windows Phone can be done at no charge using the free Express edition of Visual Studio 2010. More extended versions of Visual Studio are not free. XCode, the IDE to develop iOS applications is free of charge.

A comparison of the application development characteristics for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.9.

In terms of the graphical interface, all platforms support 2D and 3D graphics. When developing applications for Android or Windows Phone, you have full support for the functionalities of the device. However, to use these functionalities, permissions have to be granted in the application to allow the application to use those functionalities. For Apple iOs, the supported functionality is limited and can be accessed through APIs. The phone data access is unrestricted for Android and Windows Phone, which means that all data on a device can be accessed programmatically. This is different for Apple iOS, which offers limited access to the data on the device. The runtime speed for Apple iOS can be labeled

Feature / Platforms	Android	Windows Phone	iOS
CPU instruction set	ARM	x86	ARM
Foundation	Java	C#	Objective-C
Learning curve	Excellent	Excellent	Difficult
Debuggers available	Good	Excellent	Good
Emulator available	Free emulator	Bundled with the IDE	Bundled with the IDE
IDE available	Eclipse	Visual Studio 2010	XCode
Cross-platform	Android	Windows Phone	only iPhone, iPod Touch and iPad
Deployment	APK files or Android Market	OTA deployment, XAP files, Market Place	AppStore or jailbreak
Development tool cost	Free	Free	Free

Table 4.9: Mobile platform application development comparison table

as best, as the Objective-C code is compiled into CPU-instructions which can be executed on the CPU directly. As Android uses the Dalvik virtual machine, an application first has to be interpreted into CPU-instructions before it can be executed. Therefore, the runtime speed of Android is labeled as average. Windows Phone uses the Microsoft .NET Compact Framework, in which applications are converted to an intermediate language, before it can be executed on the underlying hardware. Therefore, the runtime speed of Windows Phone is also labeled as average.

A comparison of the capabilities for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.10.

Feature / Platforms	Android	Windows Phone	iOS
Graphical interface	2D, 3D graphics	2D, 3D graphics	2D, 3D graphics
Functionality	No restrictions	No restrictions	Partial through API
Phone Data Access	Full	Full	Partial through API
Runtime Speed	Average	Average	Best

Table 4.10: Mobile platform capabilities comparison table

The developer community can be labeled as extensive for all of the compared platforms. The comparison of the market penetration of the compared mobile platforms is convincing, Android is the big market leader holding a share of almost half of the market sales [16]. Apple iOS currently holds the third place in market sales with a percentage of 16.8%. Market penetration for Windows Phone can be labeled as low with a market sales percentage of 3.6%. Distribution and licensing for Apple iOS requires an Apple-issued certificate which costs $99 for a Standard license and $299 for a Enterprise license. To distribute an application using the Android Market, a free of $25 is required. Distributing Windows Phone applications using the Market Place is free of costs for paid applications. For non-paid applications, the first 100 submissions to the Market Place are free of costs each year.

A comparison of the breadth for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.11.

Feature / Platforms	Android	Windows Phone	iOS
Developer Community and Support	Extensive	Extensive	Extensive
Market Penetration	Market leader	Low	Average
Distribution and Licensing	Unknown	Unknown	Requires Apple-issued certificate

Table 4.11: Mobile platform breadth comparison table

Declaring user-interfaces for Android and Windows Phone are both done using XML-based languages. Android uses XML layout files where Windows Phone uses XAML. Declaring user-interfaces for Apple iOS is done using XIB and NIB files, which can be created using the Interface Builder application. Although the Android XML layout files, the Windows Phone XAML files and the iOS XIB files are all based on XML, the iOS XIB files are very extensive and more complex structured than the others.

The following code examples present the declaration of a user-interface button for each of the compared platforms. Listing 4.4 present the declaration of a button in Android, which can be declared using a single XML element.

```
<Button android:id="@+id/button" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Hello, I am a
    Button" />
```

Listing 4.4: Declaration of a Button in an Android layout XML file.

Declaring a button in iOS is far more complex as presented in listing 4.5. The properties of a button a specified using individual XML element, opposed to the attributes using on the Android framework.

```
<object class="IBUIButton" id="909449754">
    <reference key="NSNextResponder" ref="380026005"/>
    <int key="NSvFlags">1316</int>
    <string key="NSFrame">{{15, 74}, {285, 37}}</string>
    <reference key="NSSuperview" ref="380026005"/>
    <bool key="IBUIOpaque">NO</bool>
    <bool key="IBUIClearsContextBeforeDrawing">NO</bool>
    <int key="IBUIContentHorizontalAlignment">0</int>
    <int key="IBUIContentVerticalAlignment">0</int>
    <object class="NSFont" key="IBUIFont">
        <string key="NSName">Helvetica-Bold</string>
        <double key="NSSize">1.500000e+01</double>
        <int key="NSfFlags">16</int>
    </object>
    <int key="IBUIButtonType">1</int>
```

```
        <string key="IBUINormalTitle">Zoek</string>
        <object class="NSColor" key="IBUIHighlightedTitleColor">
            <int key="NSColorSpace">3</int>
            <bytes key="NSWhite">MQA</bytes>
        </object>
        <object class="NSColor" key="IBUINormalTitleColor">
            <int key="NSColorSpace">1</int>
            <bytes key="NSRGB">
                MC4xOTYwNzg0MyAwLjMwOTgwMzkzIDAuNTIxNTY4NjYAA</bytes>
        </object>
        <object class="NSColor" key="IBUINormalTitleShadowColor">
            <int key="NSColorSpace">3</int>
            <bytes key="NSWhite">MC41AA</bytes>
        </object>
    </object>
</archive>
```

Listing 4.5: Declaration of a Button in an iPhone layout XIB file.

Listing 4.6 presents the declaration of a user-interface button on the Windows Phone. This declaration also consists of a single XML element in which attributes are used to specify its properties.

```
 <Button x:Name="button" Content="Zoeken" Width="auto" Height="76"
  Style="{StaticResource Button}" ></Button> </StackPanel> </Grid>
```

Listing 4.6: Declaration of a Button in a Windows Phone layout XAML file.

In terms of support screen resolutions, Android is designed to support the most screen resolutions as the Android platform is supported on a wide range of devices with various screen resolutions. Windows Phone does also support multiple screen resolutions, although it supports fewer resolutions than Android. Therefore, it's support for multiple screen resolutions is labeled as average. As the iOS platform only supports two screen resolutions for iPhone development, it has been labeled as least.

A comparison of the user-interface support for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.12.

Property / Platforms	Android	Windows Phone	iOS
UI declaration	XML layout files	XAML files	XIB and NIB files
Screen resolutions	Most	Average	Least

Table 4.12: Mobile platform user-interface comparison table

Windows Phone is the only platform that has native support for integrating web services into an application. By generating proxy classes based on a web service description (WSDL) file, communication with web services can easily be integrated into Windows Phone applications. Android and Apple iOS do not offer native support for web services, and communicating with web services needs to be implemented using standard HTTP requests.

Property / Platforms	Android	Windows Phone	iOS
UI declaration	Not native	Native	Not native

Table 4.13: Mobile platform web service support comparison table

A comparison of the web service support for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.13.

All of the compared platforms offer support for persistent storage using a lightweight database engine. Android and iOS have SQLite integrated, opposed to Windows Phone which has SQL Server Compact Edition integrated.

A comparison of the persistent storage support for the platforms Android, Apple iOS and Windows Phone is presented in Table 4.14.

Property / Platforms	Android	Windows Phone	iOS
Data storage	SQLite	SQL Server Compact Edition	SQLite

Table 4.14: Mobile platform persistent storage support comparison table

Chapter 5

Cross-platform Mobile Application Development

The following chapter reflects on related work in cross-platform mobile application development.

5.1 Introduction

In order to develop mobile applications using the write once, run anywhere paradigm, several solutions exist in industry today to address the issue of mobile platform heterogeneity. All of these solutions provide their own way of transforming a generic model of an application into platform specific versions of an application. In order to compare these existing tools, the following criteria have been determined to compare them:

- **Platform support** - Which platforms are supported when transforming a generic model of an application into platform specific versions
- **Device-specific feature support** - Which device-specific features are supported per mobile platform
- **Work-flow to define the generic model** - How is the generic model of an application defined per cross-platform development tool
- **Work-flow to transform the generic model to platform specific versions** - How is the generic model of an application transformed into platform specific versions
- **Output of the generation process** - What is the output of the transformation process

The criterium of platform support is based on the Obymobi requirement to run an application on the most popular mobile platform in order to cover a market segment as large as possible. Comparing the existing cross-platform development tools on support for device-specific features is based on the Obymobi requirement that device-specific features should be supported to make use of mobile phone capabilities such as phonebook access and camera. The work-flow to define the generic model of an application is compared for each of the existing tools to compare which of the work-flows offers the most ease-of-use. The criterium of the work-flow to transform the generic model into platform specific versions deals with what needs to be done to transform a generic model of an application into platform specific versions and how easily that can be done. Comparing the output of the generation process looks at the result of the transformation process and how that can be used.

In the following sections of this chapter, three cross-platforms development solutions from industry are being compared based on the criteria mentioned above. These solutions are PhoneGap [2], Titanium Mobile [6] and Rhodes [6]. Experiments have been conducted with each of these tools by using these tools to create a simple version of the Obymobi application containing four user-interface forms and communication with the Obymobi web service. Based on the process and output of these experiments, PhoneGap, Titanium Mobile and Rhodes are being compared according to the criteria mentioned above.

5.2 PhoneGap

PhoneGap [2] is an open source development tool for cross-platform mobile development. Using PhoneGap, native applications can be developed with the use of standard web languages (HTML, CSS and JavaScript). As PhoneGap is still in pre-release phase, it is currently free of cost and will stay that way for open-source projects. Commercial application developed using PhoneGap will be charged when it has been publicly launched.

Using PhoneGap, an application can be written once using HTML and JavaScript and can then be deployed to the platforms Android, iOS, Windows Phone 7, BlackBerry, Symbian and Palm webOS. An overview of the supported platforms for PhoneGap is presented in Table 5.1.

Platform	PhoneGap
iPhone	+
Android	+
Windows Mobile	-
Windows Phone 7	-
BlackBerry	+
Symbian	+
Palm webOS	+

- Platform not supported + Platform supported

Table 5.1: Supported platforms per cross-platform development framework

Device-specific features can be accessed through JavaScript libraries which are not available to webapplications by default. An overview of the device-specific features supported by PhoneGap is presented in Table 5.2.

Using PhoneGap, native applications are created which act as a wrapper application for a standard web browser control on the device. The created application handles the

Feature	PhoneGap
GeoLocation	+
PIM Contacts	+
Camera	+
Date/Time Picker	+
Native Menu / Tab Bar	+
Audio / Video Capture	-
Bluetooth	-
Push / SMS	-
Orientation	-
Native Maps	+
Vibration	+
Accelerometer	+
Sound Playback	+
Click To Call	+
Photo Gallery	-
Screenshot	-
Shake	-
Proximity Events	-

- Feature not supported   + Feature supported

Table 5.2: Support for device specific features

communication with the web application in such a way that the user has the experience that it uses a native application, but in fact it is using a web application which is executed on the device. Such a web application is defined using HTML and JavaScript and is presented like a normal web application would do. Therefore, PhoneGap applications are hybrid applications and not 100% native as they merely exist of a shell around the standard web browser control. This is in contrast with native applications, in which native user-interface controls of the platform are used for user interaction.

The following code example presents an example of a PhoneGap layout implementation of the Obymobi screen to search for a location. It consists of HTML, CSS and JavaScript to define a user-interface form with which can be using by PhoneGap for the cross-platform generation of mobile applications.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="width=320; user-scalable=no" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <title>Obymobi</title>
    <link rel="stylesheet" href="master.css" type="text/css" media="
        screen" title="no title" charset="utf-8">
    <script type="text/javascript" charset="utf-8" src="phonegap.0.9.4.js
        "></script>
    <script type="text/javascript" charset="utf-8" src="main.js"></script
        >

  </head>
  <body onload="init();" id="stage" class="theme">
    <h1>Zoek een locatie</h1>
    <h2>Zoek op naam of plaats</h2>
    <input id="tbInput" type="text" class="input large"></input>
    <br />
    <a href="#" class="btn large" onclick="searchLocation();">Zoek</a>
  </body>
</html>
```

Listing 5.1: Example of a PhoneGap HTML layout file.

In order to develop an application using PhoneGap, the application has to be written in HTML, CSS and JavaScript. By uploading the application to the PhoneGap Build Service, the HTML and JavaScript is converted and binary builds of the application are returned to deploy via app-store or other distribution.

The output of the PhoneGap Build Service consists of a hybrid application in which the user interface is defined in HTML files. These HTML files are loaded at runtime to be displayed in a webview.

5.3 Titanium Mobile

Titanium Mobile [6] is a commercially supported, open source platform for developing cross-platform applications using web technologies. It is introduced in 2008 by Appcelerator Inc.

Titanium Mobile currently offers support for generating native applications for the platforms Android and Apple iOS (including iPad). Support for generating applications for the BlackBerry platform was announced in 2010, however this is not released publicly yet. An overview of the supported platforms are presented in Table 5.3.

Platform	Titanium Mobile
iPhone	+
Android	+
Windows Mobile	-
Windows Phone 7	-
BlackBerry	-
Symbian	-
Palm webOS	-

- Platform not supported + Platform supported

Table 5.3: Supported platforms per cross-platform development framework

In order to access native platform functionality, Titanium Mobile provides a platform independent API which can be used for native UI components as well as native device functionality. An overview of the supported features by Titanium Mobile is presented in Table 5.4.

Feature	Titanium Mobile
GeoLocation	+
PIM Contacts	+
Camera	+
Date/Time Picker	-
Native Menu / Tab Bar	-
Audio / Video Capture	+
Bluetooth	-
Push / SMS	+
Orientation	+
Native Maps	-
Vibration	+
Accelerometer	+
Sound Playback	+
Click To Call	-
Photo Gallery	+
Screenshot	+
Shake	+
Proximity Events	+

- Feature not supported + Feature supported

Table 5.4: Support for device specific features

Using a platform independent API called the Titanium SDK [19], Titanium Mobile

offers an abstraction layer which can be used to define an application in a generic fashion. Defining an application can be done using the build-in editor by writing JavaScript code. In this way, an user interface and corresponding functionality can be defined by using JavaScript corresponding with the Titanium SDK.
The following code example presents an example of a Titanium Mobile layout implementation of the Obymobi screen to search for a location.

```
\\
// this sets the background color of the master UIView (when there are no
    windows/tab groups on it)
Titanium.UI.setBackgroundColor('#000');

// create tab group
var tabGroup = Titanium.UI.createTabGroup();

//
// create base UI tab and root window
//
var win1 = Titanium.UI.createWindow({
    title:'Tab 1',
    backgroundColor:'#fff',
    layout: 'vertical'
});
var tab1 = Titanium.UI.createTab({
    icon:'KS_nav_views.png',
    title:'Tab 1',
    window:win1
});

var top = 0;

var label1 = Titanium.UI.createLabel({
  color:'#999',
  text:'Zoek op naam of plaats',
  font:{fontSize:20,fontFamily:'Helvetica Neue'},
  textAlign:'center',
  width:'auto',
});

top = top + button1.height;

var button1 = Titanium.UI.createButton({
  color:'#999',
  title: 'Zoek',
  font:{fontSize:20,fontFamily:'Helvetica Neue'},
  textAlign:'center',
  width:'100%',
})

button1.addEventListener('click', function()
{
    searchLocations();
});

win1.add(label1);
win1.add(button1);

//
//   add tabs
//
tabGroup.addTab(tab1);

// open tab group
tabGroup.open();
```

Listing 5.2: Example of an Titanium Mobile JavaScript layout file.

By submitting the source files written with Titanium Mobile to a web-based, cross-compilation tool, binaries are created which can be used to deploy on mobile devices.

As Titanium Mobile cross-compiles the generic application defined using JavaScript into platform-specific versions, the output are hybrid applications.

5.4 Rhodes

Rhodes [4] is a commercially supported, open source cross-platform mobile application development framework which allows developers to create cross-platform mobile applications using web technologies [6]. It has been released in 2008 by a company called Rhomobile Inc.

Using Rhodes, applications can be created for Android, Apple iOS, BlackBerry, Windows Mobile and Symbian. An overview of the support platforms is presented in Table 5.5.

Platform	Rhodes
iPhone	+
Android	+
Windows Mobile	+
Windows Phone 7	+
BlackBerry	+
Symbian	+
Palm webOS	-

- Platform not supported + Platform supported

Table 5.5: Supported platforms per cross-platform development framework

Rhodes does offer support for device specific features, although not all features are supported by all platforms. When a platform does not offer support for a certain feature, it simply ignores it. An overview of the device specific features supported by Rhodes is presented in 5.6.

Capability	Rhodes
GeoLocation	+
PIM Contacts	+
Camera	+
Date/Time Picker	+
Native Menu / Tab Bar	+
Audio / Video Capture	+
Bluetooth	+
Push / SMS	+
Orientation	+
Native Maps	+
Vibration	+
Accelerometer	-
Sound Playback	+
Click To Call	-
Photo Gallery	-
Screenshot	+
Shake	-
Proximity Events	-

- Feature not supported   + Feature supported

Table 5.6: Support for device specific features

Creating cross-platform applications using Rhodes can be done using web technologies. In order to create cross-platform applications using Rhodes, user-interfaces can be created using HTML and CSS. The application logic can be added to the user-interface using embedded Ruby (ERB) files. The flow of an application can also be defined using Ruby code according to the Model-View-Controller (MVC) design pattern. According to [6], Rhodes makes it easy to create applications that present a series of screens that include standard UI widgets, including common phone UIs. However, it is not suitable for fast-action games and other such consumer applications which demands for rich interactive graphic interfaces or platform-specific native UI controls.
The following code example presents an example of a Rhodes layout implementation of the Obymobi screen to search for a location.

```
<div data-role="page">

  <div data-role="header" data-position="inline">
    <h1>Zoek een locatie</h1>
    <h2>Zoek op plaats of naam</h2>
  </div>

  <div data-role="content">

```

```
    <form method="POST" action="<%= url_for :action => :do_submit %>">
        <% if get_msg() %>
          <p style="color: red"><%= get_msg() %></p>
        <% end %>

        <div data-role="fieldcontain">
          <label for="tbInput" class="fieldLabel">Zoek op plaats of naam<
              /label>
          <input type="text" name="tbInput" <%= placeholder("Input") %>
              />
        </div>

        <input type="submit" class="standardButton" value="Zoek"/>
    </form>
  </div>
</div>
```

Listing 5.3: Example of a Rhodes HTML layout file.

In order to create applications using Rhodes, an initial skeleton of the application has to be created which generates the fixed directory structure of a Rhodes application and the corresponding support files. This process also generate the main UI file. The rest of the development is done by modifying and adding new files to the generated directory structure.

Although Rhodes applications are installed and run as native applications, they are in fact hybrid applications in which the HTML and CSS is rendered in a native browser UI control. Therefore, applications created using Rhodes can be classified as hybrid applications.

5.5 Evaluation

5.5.1 Platform support

As presented in 5.7, PhoneGap and Rhodes support the widest range of platforms as they both support allmost major mobile platforms from industry today. Titanium Mobile currently offers support for Android and Apple iOS, with BlackBerry to be soon released.

A comparison of the supported platforms per cross-platform development tool is presented in Table 5.7.

5.5.2 Device-specific feature support

When it comes down to support for device-specific features, Titanium Mobile offers the widest support wrapped into the APIs of the Titanium SDK. PhoneGap comes second in support for device-specific features, following by Rhodes which offers the least support. An overview of the device-specific features supported by each platform is presented in Table 5.8.

A comparison of the supported device-specific features per cross-platform development tool is presented in Table 5.8.

Platform	PhoneGap	Titanium Mobile	Rhodes
iPhone	+	+	+
Android	+	+	+
Windows Mobile	-	-	+
Windows Phone 7	-	-	+
BlackBerry	+	-	+
Symbian	+	-	+
Palm webOS	+	-	-

- Platform not supported   + Platform supported

Table 5.7: Supported platforms per cross-platform development framework

Feature	PhoneGap	Titanium Mobile	Rhodes
GeoLocation	+	+	+
PIM Contacts	+	+	+
Camera	+	+	+
Date/Time Picker	+	-	+
Native Menu / Tab Bar	+	-	+
Audio / Video Capture	-	+	+
Bluetooth	-	-	+
Push / SMS	-	+	+
Orientation	-	+	+
Native Maps	+	-	+
Vibration	+	+	+
Accelerometer	+	+	-
Sound Playback	+	+	+
Click To Call	+	-	-
Photo Gallery	-	+	-
Screenshot	-	+	+
Shake	-	+	-
Proximity Events	-	+	-

- Feature not supported   + Feature supported

Table 5.8: Support for device specific features

5.5.3 Work-flow to define the generic model

In order to create applications using PhoneGap, knowledge of HTML and JavaScript is required. The same holds for Rhodes, with the addition that knowledge of Ruby is needed in order to implement the logic into an application. Titanium Mobile does not adhere to the web technology paradigm, but uses JavaScript only to define user interfaces with. Implementation of logic into a Titanium Mobile application has be done after the cross-compilation process in the generated native code. This implies that programming knowledge of each mobile platform is required.

A comparison of the work-flows to define the generic model of an application per cross-platform development tool is presented in Table 5.9.

Framework	PhoneGap	Titanium Mobile	Rhodes
User-interface	HTML	JavaScript	HTML
Web services	JavaScript	JavaScript	Ruby
Persistent storage	JavaScript	JavaScript	Modeling

Table 5.9: Work-flow to define the generic model per cross-platform development framework

5.5.4 Work-flow to transform the generic model to platform specific versions

Creating applications for each of the supported mobile platforms on PhoneGap is done by uploading the web application files to the PhoneGap Build Service. The result of the build process are binary files which can be deployed directly on mobile devices. Titanium Mobile allows you to deploy an application directly on your device or open the development project in your IDE and run it yourself. Using the latter, you are able to modify a generated project to your own needs and compile it yourself from that point on. Rhodes offers an integrated build process which results in binary files which can be deployed on directly on the mobile devices.

A comparison of the work-flows to transform the generic model of an application to platform specific versions per cross-platform development tool is presented in Table 5.10.

Framework	PhoneGap	Appcelerator	Rhodes
Work-flow	PhoneGap Build Service	Compilation and Development	Integrated Build Process

Table 5.10: Work-flow to transform the generic model to platform specific versions per cross-platform development framework

5.5.5 Output of the generation process

The output of the cross-compilation process of PhoneGap and Rhodes are hybrid applications, in which the user-interface defined in HTML is rendered into a standard webview component of the underlying device. This is opposed to Titanium Mobile, in which native code is being generated for each of the supported platform. The user-interface generated by Titanium Mobile is therefore completely native using native UI components.

A comparison of the output of the generation process per cross-platform development tool is presented in Table 5.11.

5.5.6 Summary

Although the cross-platform development solutions described in this chapter offer support for various platforms and device-specific features, they all are statically bound to certain

Framework	PhoneGap	Titanium Mobile	Rhodes
Output	Hybrid	Hybrid	Hybrid

Table 5.11: Output type per cross-platform development framework

platforms and/or features as the support for such features are determined by the underlying APIs of the used framework. This means that none of the reflected solutions can be extended by the end-user itself, with the exception of Titanium Mobile in which the user is able the modify the generated code afterwards. Although this tends to be a flexible solution, programming knowledge of the generated code per platform is required. Next to that, Titanium Mobile offers the least support in terms of mobile platforms. In general, none of these existing cross-platform mobile application development solutions from industry offer a wide range of mobile platforms in combination with the flexibility of extending generated code afterwards.

Chapter 6

Requirements

The following chapter reflects on the requirements for cross-platform mobile application development, which are also based on the results of comparing of existing cross-platform development tools from industry. The first section of this chapter focuses on the requirements for the Obymobi application. Requirements for cross-platform mobile application development are reflected in the second section.

6.1 Obymobi application requirements

As described in Chapter 2, the current focus of Grenos is to develop a native application as earlier versions of the Obymobi application did not meet the requirements in terms of performance or deployment capabilities. Developing native applications has several advantages over virtualized or webapplication solutions, like optimal performance, extensible code per platform, use of device-specific features and a native look-and-feel for an application. A mobile application typically consists of three layers, namely:

- **User-Interface** - The user-interface is an essential part of a mobile application in order to interact with the end-user.
- **Webservice Connectivity** - In order to connect a mobile application to other services, web services provide a way of communicating with back-end services.
- **Persistent Storage** - The persistent storage layer of a mobile application provides a way of storing user data for an application. In this way, data like preferences or common used data can be accessed without having to connect to a webservice first.

6.1.1 Cross-platform mobile application development requirements

According to the Generate approach described in [30], cross-platform platform development can be achieved by automatically generating multiple versions of an application using a intelligent generator. There currently already exist several tools and frameworks in the industry today for developing mobile applications in a cross-platform fashion. These existing solutions are being reflected in Chapter 5. However, these tools and frameworks offer cross-platform development in a static fashion as they only support a limited set of platforms and/or features. In order to overcome this limitation issue, extensibility is a key aspect of an intelligent generator. By having a generator which can be extended in terms of supported platforms and features, the static character of existing solutions can be overcome. In order to do this, the supported platforms and features should be incorporated in a dynamic fashion into the generator, so they can be extended by the end-user.

In order to automatically generate multiple versions of an application using a intelligent generator, an application has to be defined in a generic way so that the generator can transform this generic model into platform specific versions. Based on the typical layers of a mobile application mentioned earlier in this chapter, the user-interface, the web service connectivity and the persistent storage capabilities should be defined in a way.

Although a typical mobile application consists of an user-interface, web service connectivity and persistent storage, another essential part is the business logic which defines the behavior of such an application. As defining business logic in a generic fashion is quite a hard task to do, there must be some way to allow a user to inject custom user code, which has to be incorporated into the code generation process. In this way, the user is able to define custom code per platform to implement the desired business logic to define the behavior of the application.

6.1.2 Overview

The following list consists of an overview of the requirements for cross-platform development of a mobile application:

- **Definition of an user-interface in a generic way** - User-interfaces have to be defined in a generic model to use in the generation process to create platform specific versions with
- **Definition of web service connectivity in a generic way** - Web services have to be defined in a generic model to use in the generation process to create platform specific versions with
- **Definition of persistent storage in a generic way** - Persistent storage has to be defined in a generic model to use in the generation process to create platform specific versions with
- **Definition of platforms and features** - Mobile platforms and corresponding features have to be defined to specify what is supported by which platform when generating platform specific versions of an application
- **Injection / preservation of custom user code** - Injecting custom user code should be supported to enable platform specific functionality which cannot be encapsulated into an abstract model
- **Code generation of native code per platform** - Transforming the generic model of an application into platform-specific versions by generating native code per output platform

Chapter 7

Development and Design

In order to overcome the static character of existing cross-platform mobile application development frameworks from industry today, a more dynamic solution for cross-platform mobile application development is presented in this chapter. Using the Generate approach as described in [30], the design and development of an cross-platform code generator is being reflected in this chapter.

The outline of this chapter is as follows: the first section of this chapter introduces Voind, the web application consisting of the code generator. The following section reflects on how to define an application in a generic fashion, so it can be used in the code generation process. Persisting the abstractions of the platform specific properties is reflected in the third section of this chapter. The section after that consists of how to persist a generic application using Voind. Section 7.5 focusses on how to generate applications for different operating contexts. The section after that contains an overview of the workflow of the Voind application for the code generation process.

7.1 Introducing Voind

As stated in [11], Generative Programming can be applied using an intelligent generator in order to generate applications for different operating contexts. The design and development of the application which incorporates this generator named Voind is reflected in the following sections.

The development methodology applied during the development of Voind is Feature Driven Development (FDD) [28], an agile development methodology originally developed by Jeff de Luca. Feature Driven Development states that the first step in the development process is to create an overall model of the system and its context.

The following feature model presents the overall model of the Voind application. The root feature of the model represents the Voind application. The features below the root feature are the main features which consists of:

- **Front-end** - Enables the end-user to interact with Voind so that generic models can be defined and can be transformed into platform specific versions

- **Persistent storage** - Persistence of all data needed to transform the generic model into platform specific versions
- **Projects** - Definition of projects for cross-platform development
- **Platforms** - The supported platforms to generate applications for
- **Models** - The generic models to use in the generation process
- **Mappings** - Data associated with mappings to transform the generic models into platform specific versions
- **Code generator** - The generator which transforms the generic models into platform specific versions

The Voind web application is implemented using a n-tier architecture, as presented in Figure 7.2. The core layer holds the base functionality such as configuration classes, constant values and enumerations. The data access layer handles the persistent storage with the SQL Server 2008 database [1]. It is generated by an Object-Relational Mapper so that the data in the database can be treated as objects. The business logic layer contains the actual logic to convert a generic application into platform specific versions. It consists of the following functionality:

- **Parser** - The parser which is generated by the parser generator based on the XUL-based grammar to parse generic user-interface definitions with
- **Models** - The models which are used by the parser to convert the user-interface definition into an object representation
- **Template engine** - The template engine is used to parse platform specific template files with and convert them into an object representation
- **Code generator** - The code generator uses the platform specific templates and combines them with platform specific output to generate code for multiple platforms with

7.2 Defining a Generic Application

In order to design and develop a generator that is able to create multiple versions of an application for specific operating contexts we first need to find a way to define an application in a generic fashion [12, 14]. As mentioned in Chapter 6 a typical mobile application consists of three layers, namely a user-interface, webservice connectivity and persistent storage. These typical layers of an application need to be abstracted in order to make them useable by a code generation process. Existing cross-platform mobile application development solutions, as reflected in Chapter 5, also define applications in a generic way using web technologies like HTML and JavaScript. However, as these ways of defining an application have a static character as they are bound to the limitations of the underlying APIs of such a framework, a more dynamic way of defining an application is needed.

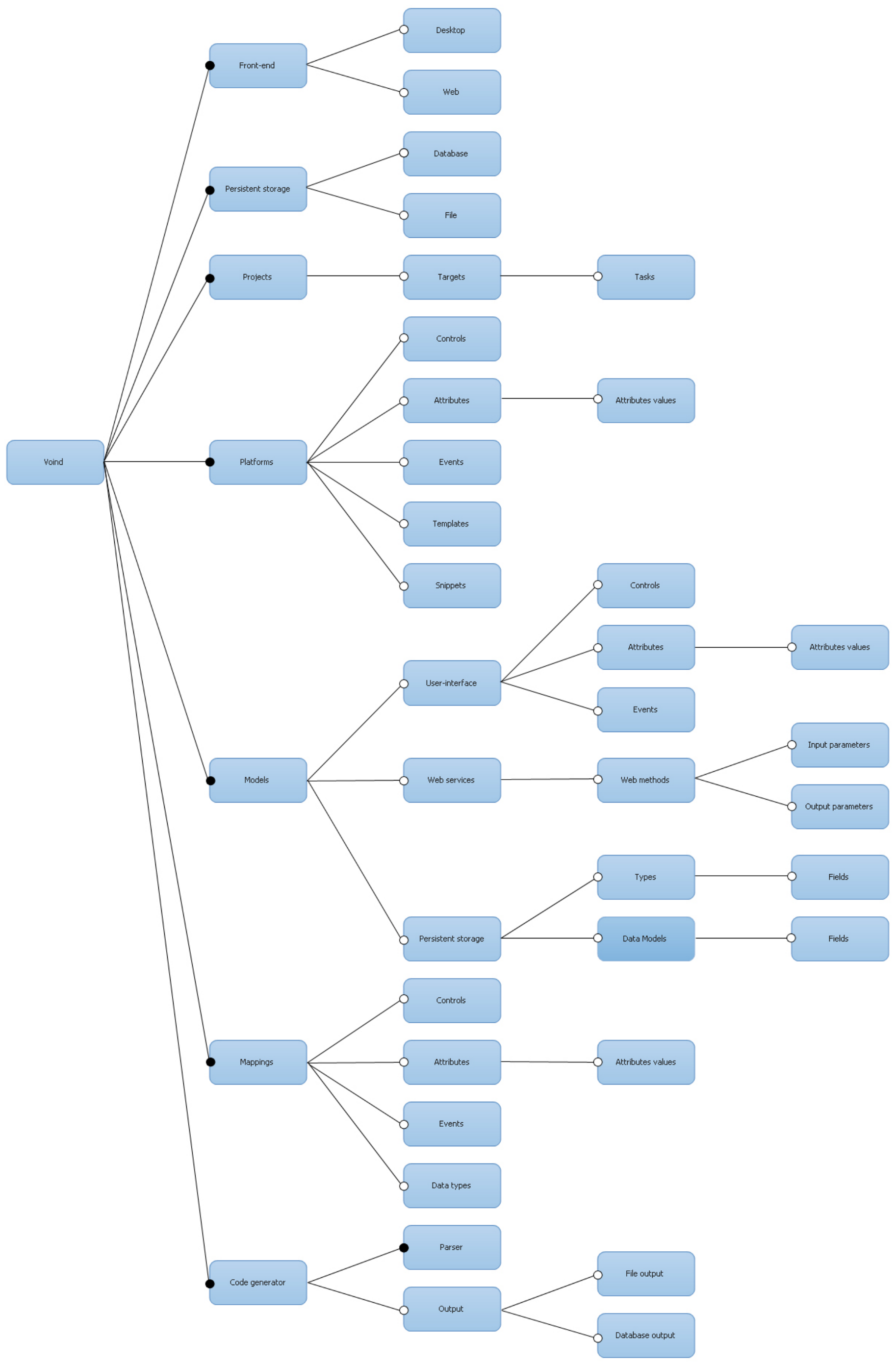

Figure 7.1: Overall feature model of the Voind application.

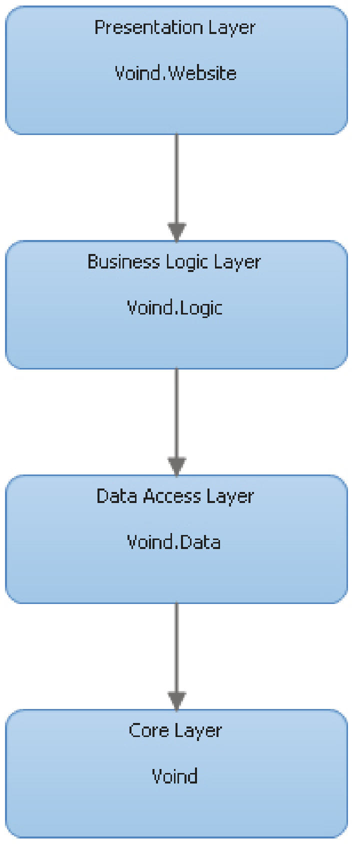

Figure 7.2: N-tier architecture of the Voind application.

7.2.1 User-Interface

Defining a user-interface can be done using all sorts of technologies and is merely depending on the underlying platform. For example, Android uses XML resource files, Windows Phone 7 uses XAML files, Apple iOS uses XIB files and so on. In order to define a user-interface using a generic model, we need to abstract the ways of defining an interface of the several platforms into a abstracted model which serves as a Domain-Specific Language (DSL) for defining UIs.

An example of such is the XML User Interface Language (XUL), developed by Mozilla [3] which can be used to define user-interfaces in XML format. Although XUL provides a low-level way of defining interfaces, it is however restricted to a limited set of components. When defining a UI for a mobile application, this set of components might not be able to abstract all of the components in the targeted mobile platforms. Therefore, an XUL based interface language is needed which can be extended with new abstractions when needed.

In order to create a parser which is able to parse the XUL based interface language syntax the parser generator GOLD Parser [29] is used. By specifying a source grammar

written in the GOLD Meta-Language, Look Ahead Left-to-right Right-deriviation (LALR) and Deterministic Finite Automaton (DFA) parse tables are produced, which are used by the GOLD parser engine to parse a source file.
Below is an overview of the grammar used to parse the XUL based interface language:

```
"Start Symbol" = <Window>

{Symbol Chars}      = {Printable} - {Letter} - [<>'"=] - {Whitespace}
{Attribute Chars}  = {Printable} - ["]

Comment Start  = '<!--'
Comment End    = '-->'

 AttributeValue =   '"' {Attribute Chars}* '"'

Name      = {Letter}{Alphanumeric}*

<Window>       ::= '<window>' <Controls> '</window>'

<Controls>     ::= <Control> <Controls>
                |

<Control>      ::= <Start Tag> <Controls> <End Tag>
                | <Unary Tag>

<Start Tag> ::= '<' Name <Attributes> '>'
<End Tag>   ::= '</' Name '>'

<Unary Tag> ::= '<' Name <Attributes> '/>'

<Attributes> ::=  <Attribute> <Attributes>
                |

<Attribute> ::= Name '=' AttributeValue
```

Listing 7.1: Grammar for XUL based interface language.

The grammar presented above represents the BNF grammar used by the parser generator to generate the parse tables with for the XUL based user-interface language. The Window symbol is the root symbol which represents a user-interface form. The forms of a user-interface consists of controls, which are specified in the grammar using the Control symbol. These controls correspond to the generic controls defined in the Voind web application. Each control can have attributes to specify values with using the Attribute symbol. These attributes and attribute values correspond to the generic attributes and attribute values defined in the Voind application.
The following code example consists of a higher-level representation of a form using the XUL based user-interface language. The form consists of a vertical box containing three controls, namely a label, a textbox and a button.

```
<window>
  <box orient="vertical" id="bTop" width="fill" height="100%">
    <label id="lblSearchLocation" text="Zoek op naam of plaats" width="
        fill"
     height="5%"></label>
    <textbox id="tbSearchLocation" width="fill" height="10%" ></textbox>
    <button id="btnSearchLocation" text="Zoeken" width="fill" height="10%
        " style="Button" onClick="getLocations()"></button>
  </box>
</window>
```

Listing 7.2: Example of a higher-level representation of a simple window.

7.2.2 Webservice Connectivity

In order to define webservices into an abstracted model, the web methods of a webservice can be abstracted as well as their output types and the corresponding input parameters. Using this abstracted model of a webservice, proxy classes for communicating with a web service can be generated for different operating contexts.

7.2.3 Persistent Storage

Persistent storage can be abstracted in a way that data models can be defined generically in the same way that Object-Relational (OR) Mappers do. By defining the models and the corresponding fields in an abstracted model, Data Access Objects (DAO) can be generated for different operating contexts.

7.3 Persisting Dynamic Abstractions

As discussed in Chapter 5, current cross-platform development solutions from industry today offer various ways of using abstracted representations of an application and use those representations to generate platform specific applications with. However, these solutions are all limited to the abstractions implemented in such a framework and cannot be extended without releasing a new version. In order to overcome this limitation issue, the end-user should be able to modify the abstractions without having to wait for a release of the used cross-platform development framework. This can be achieved by incorporating the abstractions in a dynamic way into the framework, so that these abstractions can be adjusted by the end-user.

Figure 7.3 presents a fragment of the Voind datamodel which is used to dynamically persist the user-interface abstractions and platform specific information. On the lefthand side of the model are the entities which are used to store the platform specific information. The *Control* entity is used to store platform specific controls in. This entity has relations to the *Attribute* (via *ControlAttribute*) and *Event* (via *ControlEvent*) entities which present the corresponding attributes and events of the control. On the righthand side of the model are the entities which are used to store the abstractions in. The *GenericControl* entity is used to store abstracted controls in (i.e. button). This entity has relations to the *GenericAttribute* (via *GenericControlGenericAttribute*) and *GenericEvent* (via *GenericControlGenericEvent*) entities which present the corresponding attributes and events of the generic control. The entities in the middle of the model represent the mapping entities. These entities are used to store the links between abstracted components and platform specific components in order to make the translation from a generic component to a platform specific component.

Table 7.1 presents an overview of the controls currently supported by the Voind prototype.

7.4 Persisting a Generic Application

7.4.1 User-Interface

Persisting the higher-level user-interface XML representation is done by persisting the abstracted user-interface to the database. The process of persisting a user-interface happens when a form is saved in the Voind web application and the parser creates the Abstract Syntax Tree (AST) and then saves this to the database according to the datamodel presented in Figure 7.4.

The *Form* entity represents the generically defined user-interfaces, which are parsed from the XUL-based user-interface language. The *FormGenericControl* entity holds the links between the user-interface forms and the generic controls which specifies which control belongs to which form. The *FormGenericControlGenericControlGenericAttribute* and *FormGenericControlGenericControlGenericEvent* entities represent the links between the controls on each user-interface form and the attributes and events. The *UserCode* entity is used to persist custom user code per user-interface form with, which is used in the code generation process and enables end-users to write custom code per output platform.

User-interface forms can be extended with platform-specific custom user code which allows developers to write snippets of code specific for a single platform. In this way, device-specific features can be implemented for certain platforms and can be left away for other platforms.

7.4.2 Webservice Connectivity

Webservice connectivity is an essential component of the Obymobi application as end-users need to send their composed orders to the central Obymobi webserver. In general, communicating with webservices is a key element in mobile applications in order to connect with other services and or back-end processing. As this is a key element, connecting to webservices should be incorporated in the higher-level specification of a mobile application. This higher-level specification of a webservice is then used in the code generation process to generate proxy classes for accessing a webservice with.

In order to do this, a mechanism for deriving a generic, higher-level specification of a webservice was needed. The Web Services Description Language (WSDL) is a language based on XML to specify the model of a web service with. The WSDL of a web service contains information like the operations and corresponding in- and output types, but also

-

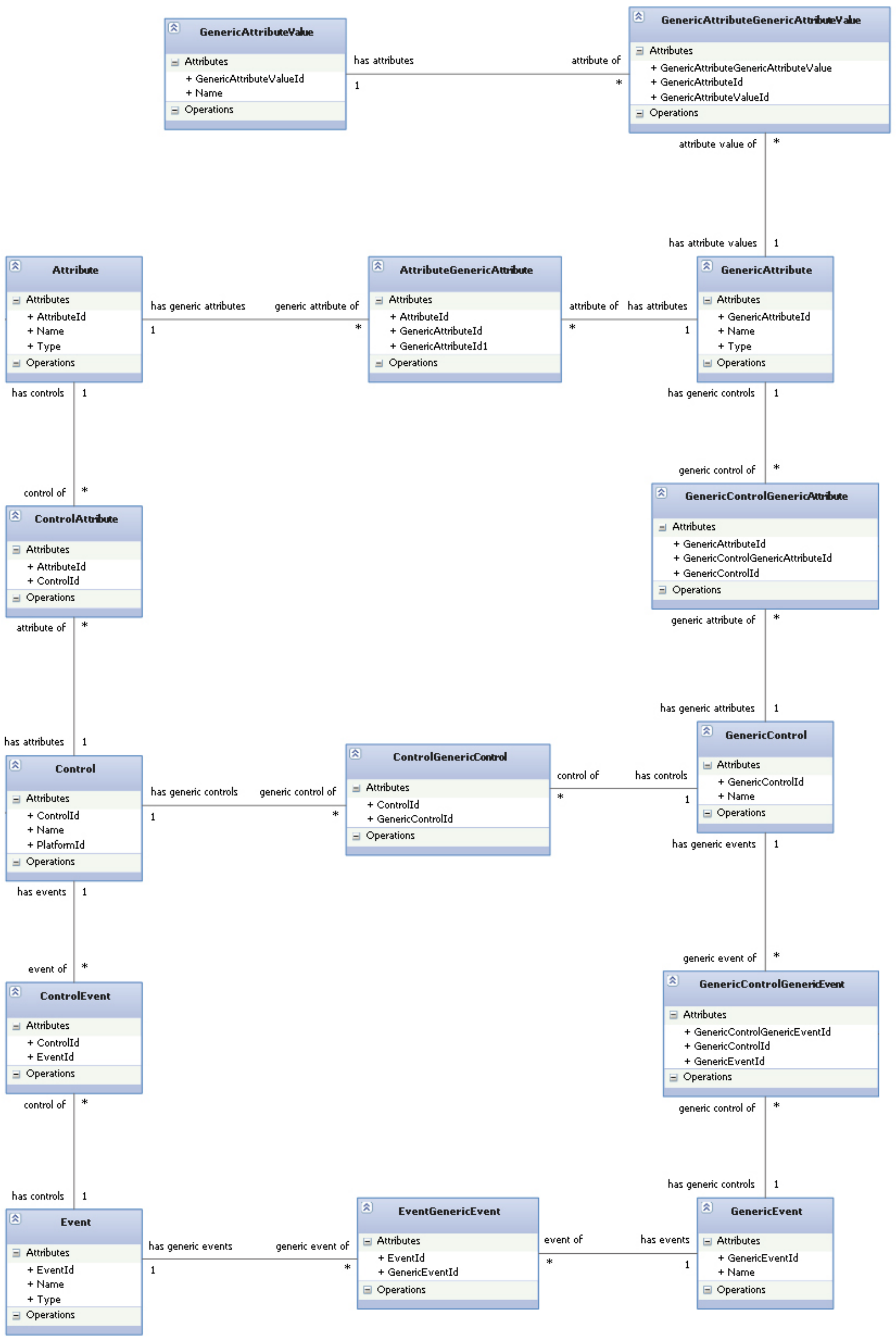

Figure 7.3: Fragment of the Voind datamodel for the mappings.

XUL control	Android	Windows Phone 7	iPhone
button	Button	Button	UIButton
checkbox	CheckBox	CheckBox	UISwitch
radiobutton	RadioButton	RadioButton	UIPickerView
textbox	EditText	TextBox	UITextField
vbox	LinearLayout	StackPanel	UITableView
hbox	LinearLayout	StackPanel	UITableView

Table 7.1: XUL control mappings

user-defined datatypes. The WSDL of a web service can be stated as a higher-level representation of a web service.

As the WSDL reflects the higher-level specification of a web service, this specification can be used in the code generation process, when generating the proxy classes for communicating with a web service. However, as we want to persist the higher-level specification into our own datamodel, a conversion of the information in the WSDL to our higher-level specification of a web service is needed.

To achieve this, a WSDL parser had to be developed which is able to parse a WSDL file and convert that to the datamodel presented in Figure 7.5.

The *Webservice* entity is used to persist web services, which has relations to *WebMethod* and *WebserviceType* entities. The web methods of a web service represent the operations which can be called at the web service. Each web method has a return type and can have input parameters for each of the web methods. The input parameter are persisted using the *InputParameter* entity. The return types of the web methods can be user-defined types. Therefore, the *WebserviceType* entity is used which represents user-defined web service types. The *WebserviceField* entity presents the fields of a user-defined web service type. With the meta information known for the web methods and their corresponding types, proxy classes can be generated in the code generation process for communicating with the web services.

Using the *System.Web.Services.Description* namespace from Microsoft .NET, the XML representation of a WSDL file can be examined and the required information can be retrieved. The next step is to synchronize the extracted information and persist it into the Voind datamodel. The following information is extracted from the WSDL file:

- the web service operations with their corresponding in- and output parameters
- the user-defined datatypes with their corresponding properties

Having this higher-level specification of a web service persisted into the Voind datamodel, the code generator can use this information to generate the webservice proxy classes with. This is being reflected in the Code Generation Process section.

7.4.3 Persistent storage

As communicating with web services can be an essential part in mobile applications, saving user-data like preferences or downloaded information can also be of great value. In order to

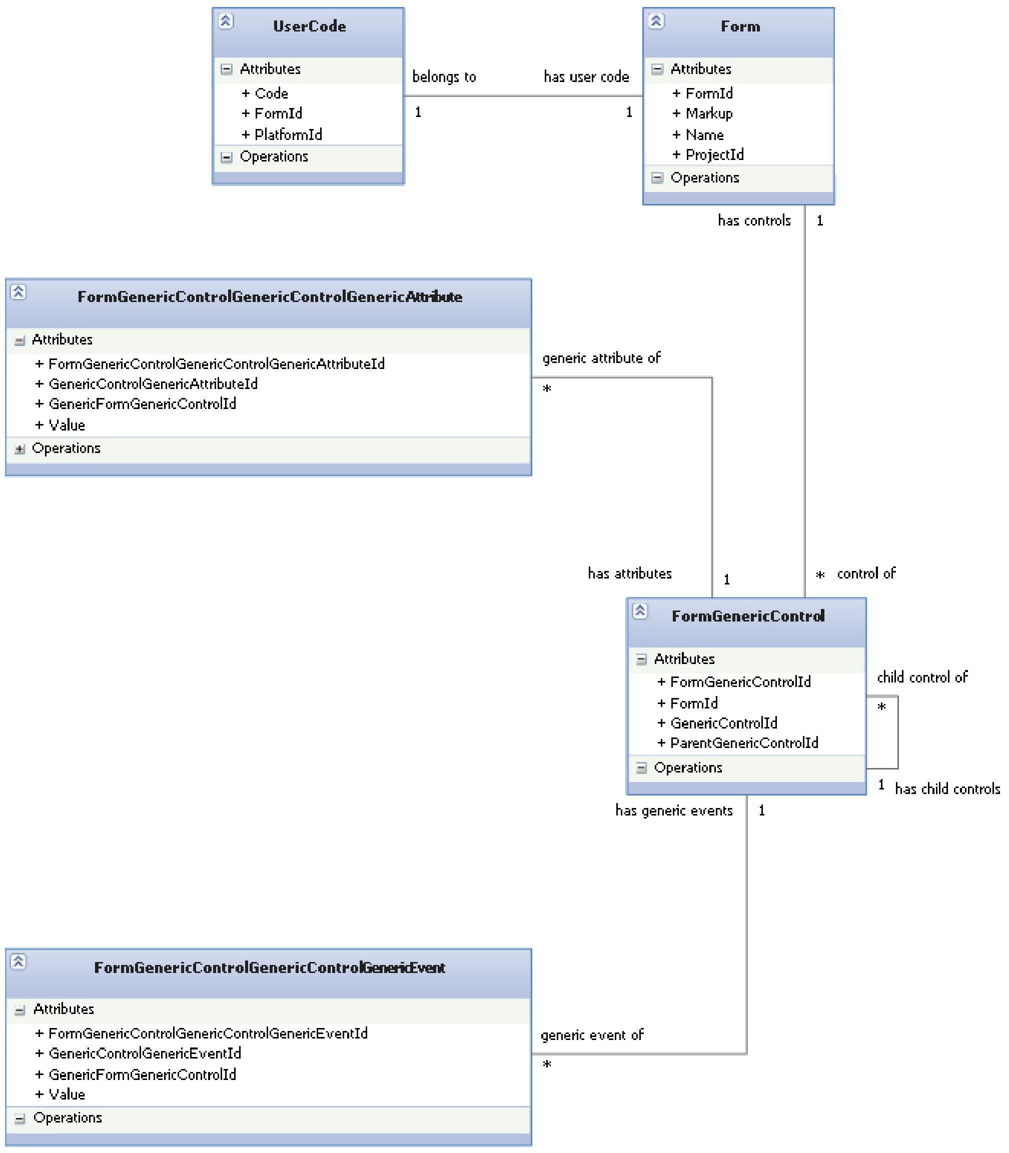

Figure 7.4: Fragment of the Voind datamodel for the persisting user-interfaces.

do this, a persistent storage mechanism is needed which has the ability to store offline data on the device.

A higher-level specification for persistent storage can be defined by a data model, which represents how the information is structured in an application. By abstracting the models and their corresponding fields, Data Access Objects (DAO) can be generated for an platform-specific application which can handle the persistent storage on a device.

Specifying the higher-level representation for persistent storage can be done using Voind

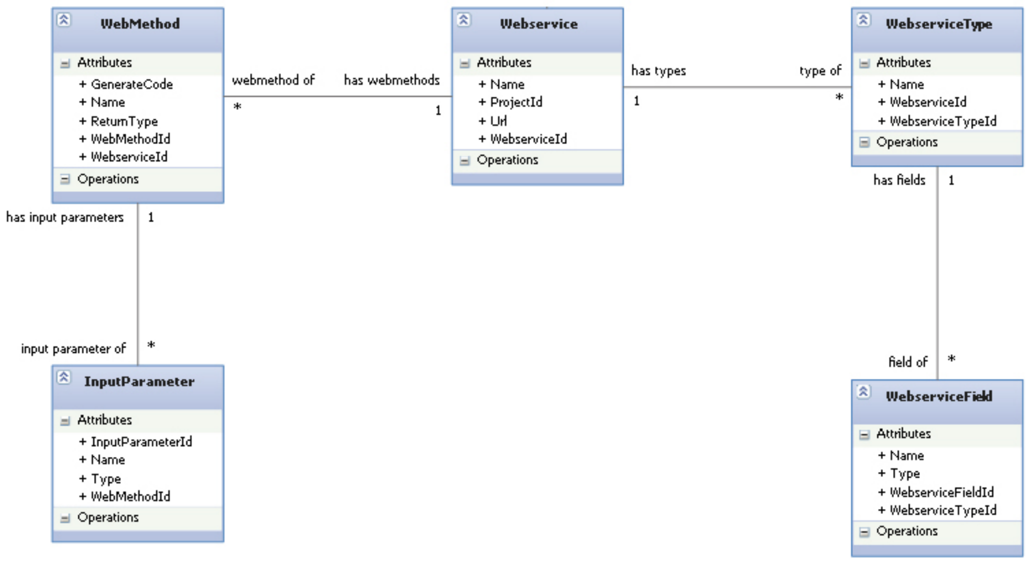

Figure 7.5: Fragment of the Voind datamodel for the webservices.

by specifying models and their corresponding fields. The data model presented in Figure 7.6 displays the structure on how these models are persisted into the Voind database.

The *Model* entity represents a data entity and is related to the *ModelField* entity which represent a data entity field.

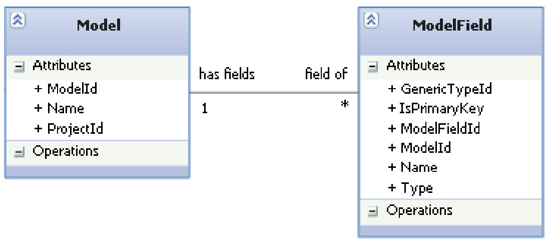

Figure 7.6: Fragment of the Voind datamodel for the persistent storage.

This specification is used in the code generation process as an Object Relational (OR) mapper, which generates code to handle persistent storage with.

7.5 Generating Applications for Different Operating Contexts

As stated in [11], a generator is a program that takes a higher-level specification of a piece of software and produces its implementation. As the previous sections of this chapter focussed on defining and persisting the higher-level specification of a mobile application, this section focusses on how to use this specification to generate implementations per mobile platform with.

In order to generate code for a specific project, targets have to be defined which are related to the targeted platforms. Specifying targets defines for which platforms code is being generated and for which screen resolutions.

The following properties of a target have to be defined:

- **Name** - Name of the target
- **Project** - Project the target belongs to
- **Platform** - Platform the template belongs to
- **Width** - Width of the screen in pixels
- **Height** - Height of the screen in pixels

After one or more targets have been defined for the code generation process, tasks have to be defined for each target. A task is used for the generation of a specific set of files and corresponds with a single template.

The following properties of a task have to be defined:

- **Name** - Name of the task
- **Target** - Target the task belongs to
- **Template** - Template used in the code generation process
- **Type** - Type of datasource
- **Filename format** - Format used for specifying the filename
- **Output directory** - Relative directory to store the output in

7.5.1 Template System

Although the platform specific implementation of user-interface, webservice and persistent storage components are present using mappings, more platform-specific code fragments are needed to generate a complete application. In order to do this, a template system is developed which uses code file templates to generate the implementations code files with.

The template system allows you to define your own templates, which can be connected to a datasource of type Form, Webservice or Model. By specifying a datasource for a template, the code generator knows on which type of datasource it should iterate during the code generation process. During this process, the code generator iterates through the items of the datasource for a specific project, and then replaces the template syntax with the implementation values.

The syntax used in the templates is used to iterate to the corresponding datasources.

The following properties of a template have to be defined:

- **Name** - Name of the template
- **Platform** - Platform the template belongs to
- **Markup** - Markup of the template

The following code fragment is an example of the markup of a template, which is used to generate Java code files with for Activity classes in Android.

```
package com.<[RootNamespace]>;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.*;
import android.widget.*;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.app.ProgressDialog;

public class <[FormName]> extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.<[FormName LowerCase]>);

        this.hookupEvents();
    }

    private void hookupEvents()
    {
    <[Foreach Controls]><[ControlType]> <[ControlName]> =(<[ControlType
        ]>)findViewById(R.id.<[ControlName]>);
    <[Foreach Events]>
      <[ControlName]>.setOn<[EventName]>Listener(<[ControlName]>_<[
          EventName]>);<[NextForeach]>
    <[NextForeach]>
    }

  <[Foreach Controls]>
    <[Foreach Events]>private On<[EventName]>Listener
    <[ControlName]>_<[EventName]> = new On<[EventName]>Listener() { <[
        EventBody]>;
    };
    <[NextForeach]><[NextForeach]>

  /* User Code Region *//* End User Code Region */
}
```

Listing 7.3: Template example for generating Java code files for Android Activity classes.

7.5.2 Code Generation Process

The code generation process deals with combining the higher-level specification of a mobile application with the corresponding templates in order to generate code files per platform with. The first step in generating code is to select the project to generate code for. After that, a selection have to be made from targets belonging to the selected project in order to define for which platforms code has to be generated. When the code generation process starts, the higher-level representation of the project's application is retrieved from the database. By iterating through the tasks of the project, the corresponding template files are combined with the higher-level representation in order to generate the platform-specific code files. Generates code by combining generic models with templates and injection of user code.

7.5.3 Code Generation Output

The code generation process is concerned with combining higher-level specifications with templates in order to produce code for different implementation platforms. The output of this process is stored into the database using revision numbers. When code is generated, it is stored into the database by the datamodel presented in Figure 7.7. By opening the generated output in the Voind webapplication, the output can be downloaded to the development environment. Downloading the output can be done per single code file, or by downloading the complete sets of files generated for a specific platform.

The *Project* entity is used to persist Voind projects and represents the cross-platform mobile application development projects. Targets have to be defined per project using the *Target* entity, which represent the output platforms for the code generation process. Individual tasks have to be defined per target using the *Task* entity, which correspond with the generation of files per output platform. The entities *ProjectOutput*, *TargetOutput* and *TaskOutput* are used to persist the output of the code generation process with. Each time code is being generated, the output of the generation process is being persisted using a revision number.

7.6 Workflow Overview

The first step to create an application using Voind is defining a project by specifying a project name and the root namespace. After a project has been created, targets have to be added to the project which define for which mobile platform code has to be generated. Adding targets to a project can be also be done at a later time in an existing project, for example when a new platform is added to the system. When the targets of a project have

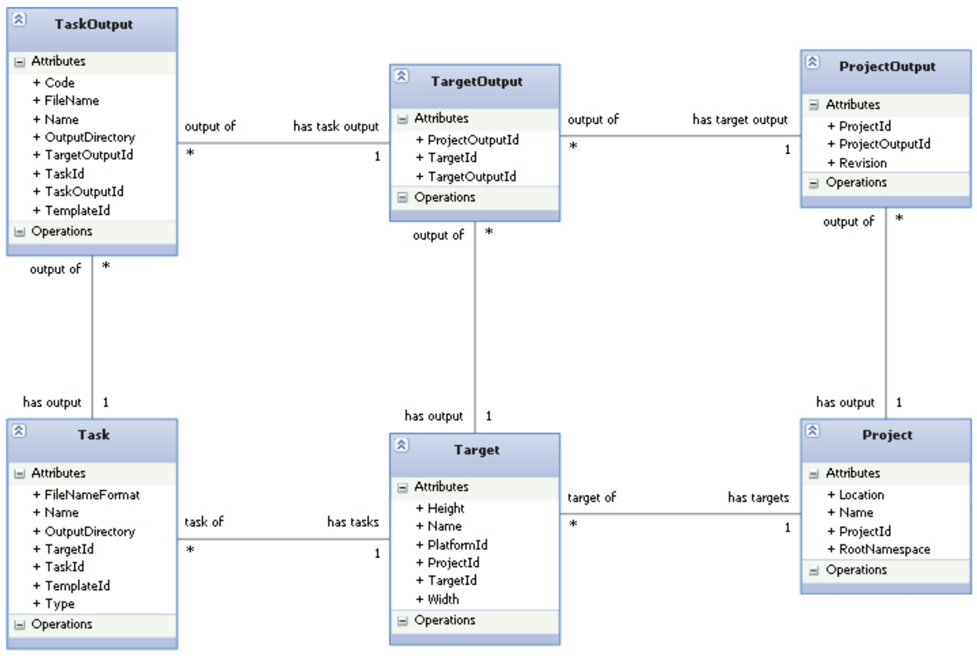

Figure 7.7: Fragment of the Voind datamodel for the code generation process.

been defined, tasks have to be created which determine which files are being created in the code generation process.

After a project has been created, higher-level specifications of user-interface forms, web services and data models can be added. Adding such specifications is merely a matter of adding items in the Voind web application and specifying the required properties. In case of a user-interface form, a name, project and higher-level XML representation have to be specified. In case of a web service, a name, project and url of the web service have to be specified. Using the WSDL synchronization routine, all required information from a web service can be synchronized automatically. In case of a data model, a name, project and the corresponding fields have to be specified. After specifying user-interface forms, web services and/or data models, code can be generated. Generating code is merely a matter of specifying for which platforms code has to be generated and starting the code generation routine.

7.7 Summary

In this chapter the development and design of the Voind application has been presented. The first step in the development process consisted of finding a way to generically define a mobile application by abstracting the user-interface, persistent storage and communication with web services. Defining user-interfaces is done using a XUL-based interface language, which can be used to specify the forms of a mobile application in an abstracted fashion. Communication with web services is specified in a generic model by abstracting the operations, types and parameters for a web service, which can then be used to generate proxy classes for communication with a web service. The persistent storage of an application can generically be defined by specifying a data model, which is then used to generate data access objects (DAO) with. Having a way to generically define the artifacts of a mobile application (like [15]), the next step consisted of persisting those abstracted artifacts into a database. This is done by designing a data model that incorporated generic components, platform specific components and the mapping between them. A database has been developed using this data model, which is connected to the Voind application to persist the generic and platform specific meta information with. Several relevant fragments of the Voind data model have been presented in this chapter. By combining the abstracted meta information of a mobile application with platform specific meta information, a mapping is made from abstracted artifacts to platform specific implementations. This is done using a template system, which can be used to define code templates per output platform with. The code generator of the Voind application then combines platform specific templates with the meta information of an application to generate code per output platform with.

Chapter 8

Evaluation

The following chapter presents the evaluation of the thesis project. The first section reflects on the goals of the thesis project based on the requirements for cross-platform mobile application development. The second section reflects on the Obymobi case study, which presents the development of the Obymobi application using the Voind application.

8.1 Project Goals

In order to evaluate the developed product and the process of developing the product, the goals of the thesis project have to stated. An overview of the requirements for the Obymobi application is presented below, which serves as a base for the project goals.

8.1.1 Requirements

The requirements of the Obymobi application are listed below which are used to evaluate the cross-platform development capabilities of the Voind application.

- Obymobi should run on the most popular mobile platforms in order to cover a marktsegment as large as possible
- Obymobi development should follow the write once, run everywhere paradigm
- Obymobi should be able to use device-specific features such as phonebook, GPS, etc.
- Obymobi should have a native look-and-feel

8.1.2 Goals

The following goals have been specified based on the requirements mentioned above and the research questions as mentioned in Chapter 1.

Development of a cross-platform development framework which:

- Enables cross-platform development for the most popular mobile platforms

- Minimizes development effort when developing for multiple platforms
- Adheres to the write once, run everywhere paradigm
- Enables the use of device-specific features
- Delivers applications which have a native look-and-feel

8.1.3 Evaluation Scenarios

In order to evaluate the Voind application on cross-platform development capabilities based on the requirements mentioned above comparison criteria have been defined. Using these comparison criteria, the results of the thesis project are evaluated. A case study has been conducted to develop the Obymobi application using the Voind application. The following scenarios are being used to evaluate the results of the case study:

- **Development of an initial release** - When a first release of an application is developed using Voind, the lines of code needed to be written using Voind is compared with the lines of code needed to be written during manual development for different operating contexts.
- **Extending a release with new features** - When new features are added to an existing application using Voind, the lines of code needed to be written using Voind is compared with the lines of code needed to be written during manual development for different operating contexts.

8.2 Evaluation of Voind

The following section presents an evaluation of Voind by presenting an overview of the application and comparing it with cross-platform development tools from industry. To evaluate Voind, the same criteria are used as when evaluating the existing cross-platform development tools as presented in Chapter 5. These criteria are:

- **Platform support** - Which platforms are supported when transforming a generic model of an application into platform specific versions
- **Device-specific feature support** - Which device-specific features are supported per mobile platform
- **Work-flow to define the generic model** - How is the generic model of an application defined per cross-platform development tool
- **Work-flow to transform the generic model to platform specific versions** - How is the generic model of an application transformed into platform specific versions
- **Output of the generation process** - What is the output of the transformation process

The Voind application consists of a web application, which can be used for cross-platform code generation for mobile applications. Voind is based on the following components:

- **Architecture** - A data driven, N-tier structured architecture on top on a SQL Server 2008 database
- **Mapping System** - A mapping system for connecting higher-level user-interface components to platform-specific user-interface components
- **Parser** - A parser used to parse XML based user-interface specifications with
- **Template System** - A template system for the specification of code templates
- **Code Generator** - A code generator which generates platform-specific code files with
- **Snippets** - A snippet system to specify platform specific code snippets

8.2.1 Platform support

The Voind application uses an XUL based language for the definition of user-interfaces, which can be extended by defining new user-interface components. This is an advantage over existing cross-platform development frameworks in terms of extensibility. Where other cross-platform frameworks have a static character in which the supported components and platforms are statically bound to such a framework, Voind can be extended by the user by adding new generic components and mapping them to platform specific components.

8.2.2 Device-specific feature support

Any device specific feature can be implemented using Voind although additional knowledge of a programming language is needed per implementation platform. By adding platform specific code to UI forms or data model entities, additional user-defined code is persisted into a Voind project which is used in the code generation process. In this way, end-users are able to implement any device specific feature.

Voind also consists of a snippets component, which allows developers to dynamically define platform specific code snippets. These snippets can correspond to device specific features and can be inserted when writing user-defined code.

Existing cross-platform development tools from industry offer static APIs to make use of device specific features. Using such APIs, there is no additional programming knowledge of an implementation context needed.

8.2.3 Work-flow to define the generic model

Voind is implemented as a web application which allows the application to be accessible on every platform using a standard web browser. This allows that all applications created using Voind are stored in a central repository.

The persistent storage of the Voind application is based on a SQL Server 2008 database underneath the web application. The database acts as a central repository in which users can store their information like platform specific controls and custom user mappings. The advantage of having a central repository compared to a decentral storage of data is that the community can help extending the Voind repository by adding new information like new platforms, controls, mappings, etc.

Defining a generic model of an application is done by adding user-interface, data model entity or web service specifications using the Voind web application. The following subsections will evaluate how such specifications can be added.

User-Interface

A user-interface form can be defined by specifying the controls of a form using a XUL based language. Listing 8.1 presents an example of such a user-interface specification.

```
<window>
 <box orient="vertical" id="bTop" width="fill" height="100%">
     <label id="lblSearchLocation" text="Zoek op naam of plaats" width="
         fill" height="5%"></label>
     <textbox id="tbSearchLocation" width="fill" height="10%" ></textbox>
        <button id="btnSearchLocation" text="Zoeken" width="fill" height=
            "10%" style="Button" onClick="getLocations()"></button>
 </box>
</window>
```

Listing 8.1: Code example of a user-interface specification.

The XUL based language used to define user-interface forms in Voind offers a more lightweight and simpler way of specifying such forms if we compare it to existing cross-platform development tools. Listing 8.2 presents a user-interface specification from PhoneGap, which consists of standard HTML code combined with CSS and JavaScript. In Chapter 5, three cross-platform development tools from industry were evaluated. For each of these tools, a code example for implementing an equivalent user-interface form was presented. To illustrate the simplicity of a Voind user-interface specification, a comparison is made between a Voind user-interface specification and the most common way of specifying user-interface specification from one of the existing tools. As standard HTML combined with CSS and JavaScript is the most used way of specifying a user-interface opposed to specifying it completely in JavaScript or combined HTML and Ruby, the PhoneGap UI specification is used for comparison.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta name="viewport" content="width=320; user-scalable=no" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <title>Obymobi</title>
    <link rel="stylesheet" href="master.css" type="text/css" media="
        screen" title="no title" charset="utf-8">
    <script type="text/javascript" charset="utf-8" src="phonegap.0.9.4.js
        "></script>
    <script type="text/javascript" charset="utf-8" src="main.js"></script>

  </head>
  <body onload="init();" id="stage" class="theme">
    <h1>Zoek een locatie</h1>
    <h2>Zoek op naam of plaats</h2>
    <input id="tbInput" type="text" class="input large"></input>
    <br />
    <a href="#" class="btn large" onclick="searchLocation();">Zoek</a>
  </body>
</html>
```

Listing 8.2: Example of a PhoneGap HTML layout file.

When specifying a UI form using Voind (as presented in Listing 8.1), less lines of code are needed compared to specifying a UI form using PhoneGap.

By specifying mappings from the generic controls to the platform specific controls in the application, the code generator is able to transform the generic specifications of user-interfaces to platform specific implementations for different operating contexts. All of the mappings in the application are completely dynamic which means that they can be maintained by the end-user itself. An advantage of having dynamic mappings compared to static mappings is that these mappings can be adjusted to make use of user-defined types. It is very common for a software development company to have a set libraries containing user-defined types which are used in their software development process. Existing cross-platform development tools like Titanium Mobile have some support for generating UI components, although these components are always the native platform components. Using user-defined mappings, developers are allowed to use their own libraries in the code generation process.

Persistent Storage

Specifying generic models for persistent storage is done by adding data model entities to a Voind project. By specifying the entities and their corresponding fields, classes are generated in the code generation process which are object-oriented representations of the data using in the application. PhoneGap and Titanium Mobile do not support the generation of persistent storage code. Rhodes does support the generation of persistent storage code. Data model entities and corresponding can be added to Rhodes using the command line. Adding data models entities and corresponding fields via the command line can be labeled as not very end-user friendly if it is compared to adding models and fields via the Voind web application.

Future work involves the development of a synchronization mechanism, which allows Voind users to connect to a database and synchronize the models with the tables and fields of a database. Unfortunately, the scope of this thesis project did not allow the development of such a synchronization mechanism.

Web Services

Generating code around web services is not supported by PhoneGap, Titanium Mobile or Rhodes. Therefore, defining generic web service models is not supported for these tools.

Defining a generic model of a web service in Voind can be done by synchronizing the Web Service Description Language (WSDL) file of a web service in Voind. All web service operations, parameters and corresponding types will be synchronized into generic models then which are used to generate code with.

Work-flow to transform the generic model to platform specific versions

The work-flow to transform a generic model of an application to platform specific versions consists of running the code generator for a Voind project. First, the desired output targets have to be selected which correspond to the output platforms. After that, the code generation routine has to be started by pressing a button on the web application. The abstracted models of a Voind project are then combined with the template files of the output platforms and code files are being generated. These code files are then stored in the database with a revision number and can be downloaded to your development environment for compilation. Existing tools from industry like PhoneGap and Titanium Mobile use black-box build services to transform a generic model into platform specific versions. When such a service is used, source files have to be uploaded to an online build service which then creates the source files or deployable binaries. In the latter case, it is not possible to make changes to the generated platform specific source code.

8.2.4 Output of the generation process

The actual code generating for different operating contexts is implemented in the code generator, which combines the persisted models with the platform specific templates to generate platform specific code files with. By iterating through the user-defined models, pre-defined constants are replaced in the template files with the mapped components per output platform. This allows a user to generate code files per output platform so that these files can be compiled to a native application per output platform. This is different compared to the existing cross-platform solutions from industry today as such tools produce hybrid applications in which the user-interface is defined in HTML and is rendered into a WebView component incorporated in a native application. The drawback of such hybrid applications is that they are not 100% completely native and support for device-specific features relies on the support for such features by the implementation of the tool. An advantage of such hybrid applications is that they do comply to the ’write once, run everywhere’ paradigm.

8.2.5 Summary

Developing mobile applications using Voind has the following advantages over existing cross-platform development tools from industry:

- **Extensible user-interface types** - Voind allows users to extend the user-interface components available for development. This overcomes the static character of existing cross-platform development tools from industry.
- **Use of user-defined types** - Voind allows users to map abstract components onto user-defined types instead of preset types, which allows users to use their own library types during development.
- **Native output** - Voind generates native source code which is then compiled into a 100% native application.

However, developing applications using Voind also has some drawbacks compared to existing tools from industry. An overview of the advantages of such tools is presented below.

- **Write once, run everywhere** - Cross-platform development tools from industry today like PhoneGap or Titanium Mobile allow users to develop mobile applications according to the 'Write once, run everywhere' paradigm by developing hybrid applications using JavaScript.
- **No additional programming knowledge needed** - As the cross-platform development of an application can be done using a single language (i.e. JavaScript), no additional programming knowledge of other languages is required.

A risk of developing an application for different operating contexts using separate development is that the different applications are getting out-of-sync when they evolve.

8.3 Obymobi case study

The development of the Obymobi mobile application has been used as a case study to design and develop the Voind application for cross-platform mobile development. As mentioned in Chapter 2, the latest approach in the development of the Obymobi mobile application was targeted at developing native applications for popular smartphone platforms.

The Obymobi back-end is currently not only used by the Obymobi mobile application, but is also used by other self-ordering solutions using tablet computers. This causes that the Obymobi back-end is constantly evolving and that changes to the underlying back-end also have to be committed to the Obymobi mobile application. Examples of such are changes to the underlying datastructures or extensions to the Obymobi web services. Using the Voind application, such changes can easily be committed to the Obymobi application as the main artifacts of the application are defined in a generic fashion which can be modified and used in the code generation process.

Using Voind, three main artifacts of the Obymobi application can be defined in a generic way, namely: the user-interface, persistent storage and web services. By specifying user-interface forms in the Voind application, user-interfaces are generated for the Obymobi application which handle the interaction with the end-user. The listing below presents a code example of the user-interface specification from which the resulting user-interfaces specifications are presented in Listing 8.4 and Listing 8.5.

```
<window>
 <box orient="vertical" id="bTop" width="fill" height="100%">
     <label id="lblSearchLocation" text="Zoek op naam of plaats" width="
         fill" height="5%"></label>
     <textbox id="tbSearchLocation" width="fill" height="10%" ></textbox>
        <button id="btnSearchLocation" text="Zoeken" width="fill" height=
            "10%" style="Button" onClick="getLocations()"></button>
 </box>
</window>
```

Listing 8.3: Code example of a user-interface specification.

Listing 8.4 presents a fragment of generated code for a user-interface form for the Windows Phone platform. It has been generated using the user-interface specification presented in Listing 8.3. Using the dynamic mapping component from the Voind application, the user-interface is translated into platform specific code for the Windows Phone platform.

```
<phone:PhoneApplicationPage
     x:Class="Obymobi.WindowsPhone7.SearchLocation"
  ...
     shell:SystemTray.IsVisible="True">

    <!--LayoutRoot contains the root grid where all other page content is
        placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent" VerticalAlignment=
        "Top">
  <StackPanel Orientation="vertical" x:Name="bTop" Width="fill" Height="
      768" >
    <TextBlock x:Name="lblSearchLocation" Content="Zoek op naam of plaats"
        Width="fill" Height="38" ></TextBlock>
    <TextBox x:Name="tbSearchLocation" Width="fill" Height="76" ></TextBox
        >
    <Button x:Name="btnSearchLocation" Content="Zoeken" Width="fill"
        Height="76" Style="{StaticResource Button}" ></Button>
  </StackPanel>
 </Grid>
</phone:PhoneApplicationPage>
```

Listing 8.4: Code example of generated XAML code for the Windows Phone platform.

Listing 8.5 presents a fragment of generated code for a user-interface form for the Android platform. It has also been generated using the user-interface specification presented in Listing 8.3. Using the dynamic mapping component from the Voind application, the user-interface is translated into platform specific code for the Android platform.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent" android:layout_height="fill_parent"
      android:orientation="vertical">
 <LinearLayout android:orientation="vertical" android:id="@+id/bTop"
      android:layout_width="fill" android:layout_height="480" >
  <TextView android:id="@+id/lblSearchLocation" android:text="Zoek op
      naam of plaats" android:layout_width="fill" android:layout_height="
      24" ></TextView>
  <EditText android:id="@+id/tbSearchLocation" android:layout_width="
      fill" android:layout_height="48" ></EditText>
  <Button android:id="@+id/btnSearchLocation" android:text="Zoeken"
      android:layout_width="fill" android:layout_height="48" style="
      @style/Button" ></Button>
 </LinearLayout>
</LinearLayout>
```

Listing 8.5: Code example of generated XML code for the Android platform.

As the Obymobi application currently does not use a database to store information retrieved from the back-end web service with, the generation of persistent storage code is currently only used for the generation of data model entities with. However, these entities are used when communicating with the underlying web service to store the retrieved information from the web service in object orientated model represention in. Using the data model meta information from Voind, entity classes, entity base classes and entity collection classes are generated which allow a developer to use model representations according to the Model-View-Controller (MVC) design pattern.

An example of a generated entity base class is presented below. Entity classes are being generated by Voind and are used in an application as object representations of data. Although these classes are not very complex to write, generating these classes saves time and is less error phrone.

```
package com.obymobi.entitybaseclasses;

public class CustomerBase
{
 public CustomerBase()
 {
 }

 private Integer mCustomerId;
 public Integer getCustomerId()
 {
  return this.mCustomerId;
 }
 public void setCustomerId(Integer value)
 {
  this.mCustomerId = value;
 }

 private String mName;
 public String getName()
 {
  return this.mName;
 }
 public void setName(String value)
 {
  this.mName = value;
 }

 private String mPhonenumber;
 public String getPhonenumber()
 {
  return this.mPhonenumber;
 }
 public void setPhonenumber(String value)
 {
  this.mPhonenumber = value;
 }
 ...
}
```

Listing 8.6: Code example of generated entity base class.

Communicating with the Obymobi back-end is essential for the Obymobi mobile application as information regarding the menu of a venue has to be retrieved and orders need to be sent. By synchronizing the WSDL description of the Obymobi web service using Voind, the operations and corresponding data types of the web service are defined in a abstract way. This meta information is then used in the code generation process to generate proxy classes for communicating with the Obymobi web service with.

Listing 8.7 presents a code fragment of a generated proxy class for communicating with the Obymobi web service.

```
package com.obymobi;

...

public class ObymobiTypedWebService {

 private String mUrl = "http://secure1.obymobi.com/ObymobiTypedWebservice
     .asmx";

 public ObymobiTypedWebService()
 {
 }

 public ObyTypedResultOfMobileClientStartupInfo GetObyVersion(String
     customerId , String hashedPassword , String version , String
     phoneInfo)
 {
  // Create the name value pairs
  List<NameValuePair> parameters = new ArrayList<NameValuePair>();
  parameters.add(new BasicNameValuePair("customerId", customerId));
   parameters.add(new BasicNameValuePair("hashedPassword", hashedPassword
       ));
   parameters.add(new BasicNameValuePair("version", version));
   parameters.add(new BasicNameValuePair("phoneInfo", phoneInfo));

  ObyTypedResultOfMobileClientStartupInfo result = (
      ObyTypedResultOfMobileClientStartupInfo) this.getFromWebservice("
      GetObyVersion", parameters, new GetObyVersionHandler());
  return result;
 }

 public ObyTypedResultOfCustomer AuthenticateCustomer(String phonenumber
     , String encryptedPassword)
 {
  // Create the name value pairs
  List<NameValuePair> parameters = new ArrayList<NameValuePair>();
  parameters.add(new BasicNameValuePair("phonenumber", phonenumber));
    parameters.add(new BasicNameValuePair("encryptedPassword",
        encryptedPassword));

  ObyTypedResultOfCustomer result = (ObyTypedResultOfCustomer)this.
      getFromWebservice("AuthenticateCustomer", parameters, new
      AuthenticateCustomerHandler());
  return result;
 }

 ...
}
```

Listing 8.7: Code example of generated web service proxy class.

In general, by specifying the user-interface, persistent storage and web service artifacts for the Obymobi application in an abstracted way, 'plumbing' code can be generated which would be very time consuming when it is done by hand. The generation of code is noy only time saving, but also less error phrone.

The following subsection presents code metrics for the development of an initial release of an application using Voind. The subsection after that presents code metrics for the development of new features to an existing application. The metrics are based on a case study in which a prototype of the Obymobi application is developed using Voind.

8.3.1 Development of the initial release

The development of the initial release of the prototype of the Obymobi application using Voind is presented here. The prototype is developed for the platforms Android and Windows Phone. Below is an overview of the metrics for the development of the initial release of the Obymobi application prototype.

Android

Table 8.1 presents an overview of the amount of files and lines of code needed when developing the Obymobi application using Voind for the Android platform. The template files and corresponding lines of code (LOC) represent the amount of template files needed for code generation and the amount of lines of code they consist of. The generated files and corresponding files of code (LOC) represent the amount of files generated in the code generation process and the amount of lines of code which are generated. The platform specific files and corresponding lines of code (LOC) represent the amount of platform specific files needed for the prototype application and the amount of lines of code they consist of.

As Voind only supports the generation of the most important artifacts of a mobile application, custom code has to be written in order to implement the business logic of an

	User-interface	Persistent storage	Web Services
Template files / LOC	2 / 34	3 / 56	3 / 300
Generated files / LOC	12 / 322	54 / 2547	3 / 9007

Table 8.1: Android metrics for generated code

application. Table 8.2 presents the metrics for the required custom user code for the Obymobi application prototype.

	Amount of files	Lines of code
Custom platform specific code	15	3894

Table 8.2: Android metrics for custom platform specific code

The initial release of the Obymobi application on the Android platform is implemented using the following number of lines of code: Based on the metrics presented above, 11876

Component	Lines of code
User-Interface	322
Persistent Storage	2547
Web Services	9007
Custom Code	3894
Total	15770

Table 8.3: Numbers of lines of code for the initial release of the Obymobi application on the Android platform

lines of codes can be generated of the total of 15770 lines of code. This means that 75% of the source code is generated for the Obymobi prototype on the Android platform.

Windows Phone

Table 8.4 presents an overview of the amount of files and lines of code needed when developing the Obymobi application using Voind for the Windows Phone platform.

	User-interface	Persistent storage	Web Services
Template files / LOC	2 / 55	3 / 46	1 / 18
Generated files / LOC	12 / 367	54 / 2892	1 / 111

Table 8.4: Windows Phone metrics for generated code

Table 8.5 presents the metrics for the required custom user code for the Obymobi application prototype.
The initial release of the Obymobi application on the Windows Phone platform is implemented using the following number of lines of code: Based on the metrics presented above,

	Amount of files	Lines of code
Custom platform specific code	16	3712

Table 8.5: Windows Phone metrics for custom platform specific code

Component	Lines of code
User-Interface	367
Persistent Storage	2892
Web Services	111
Custom Code	3712
Total	7082

Table 8.6: Numbers of lines of code for the initial release of the Obymobi application on the Windows Phone platform

3370 lines of codes can be generated of the total of 7082 lines of code. This means that 48% of the source code is generated for the Obymobi prototype on the Windows Phone platform.

As the Voind application generates platform specific code, all output is truly native as native user-interface code is being generated. This is a big difference compared to existing cross-platform development solutions from industry, which mainly focus on delivering hybrid applications with web-based user-interfaces.

More complex mobile user-interfaces which do not exist of standard platform controls cannot be generated by the Voind application yet as it a complex matter to abstract such custom controls into a generic model. Currently, the Voind application only supports a limited subset of user-interface controls which can be used to generate a basic user-interface with. Support for more controls should be added when needed. However, this can be done in the Voind application by extending the meta information.

8.3.2 Extending a release with new features

Extending an existing release of an application with new features using Voind is presented in this subsection. The Obymobi case study now focussed on extending the Obymobi application with a survey functionality. In order to implement this functionality, the following modifications had to be made:

- **One new user-interface component** - A new user-interface form had to be added to able and end-user to fill in a survey.
- **Five new datamodel entities** - New entities had to be added to the datamodel for an object representation of the data retrieved from the web service.
- **Two new web service operations** - For retrieval of surveys and and submitting of surveys answer, two new web service operations had be be implemented.

- **Business logic** - Business logic had to be added to displaying surveys and processing the survey results.

An overview of the corresponding metrics per platform for extending the Obymobi prototype with the survey functionality is presented in the following subsections.

Android

The following part reflects on extending the Obymobi prototype with the survey feature on the Android platform. By adding the new user-interface component, datamodel entities and the web service operations to Voind, new code could be generated for the Android platform. The business logic for the survey feature had to be implemented by hand. An overview of the metrics for implementing the survey feature on the Android platform is presented below.

Table 8.7 presents an overview of the amount of files and lines of code needed when adding the survey feature to the Obymobi application using Voind for the Android platform.

	User-interface	Persistent storage	Web Services
Template files / LOC	2 / 34	3 / 56	3 / 300
ΔGenerated files /ΔLOC	2 / 57	15 / 351	3 / 538

Table 8.7: Android metrics for generated code when adding the survey feature

Table 8.8 presents the metrics for the required custom user code for adding the survey feature to the Obymobi application prototype.

	Amount of files	Lines of code
ΔCustom platform specific code	2	380

Table 8.8: Android metrics for custom platform specific code when adding the survey feature

Extending the Obymobi prototype with the survey feature is implemented using the following number of lines of code: Based on the metrics presented above, 946 lines of

Component	Lines of code
User-Interface	57
Persistent Storage	351
Web Services	538
Custom Code	380
Total	1326

Table 8.9: Numbers of lines of code for extending the Obymobi application with the survey feature on the Android platform

codes can be generated of the total of 1326 lines of code. This means that 71% of the source code is generated for extending the Obymobi prototype with the survey feature on the Android platform.

Windows Phone

As a new user-interface component, datamodel entities and web service operations were added to Voind in order to generate code for the Android platform, new code could also be generated for the Windows Phone platform. The business logic for the survey feature had to be implemented by hand for Windows Phone as well. An overview of the metrics for implementing the survey feature on the Windows Phone platform is presented below.

Table 8.10 presents an overview of the amount of additional files and lines of code needed when adding the survey feature to the Obymobi application using Voind for the Windows Phone platform.

	User-interface	Persistent storage	Web Services
Template files / LOC	2 / 55	3 / 46	1 / 18
ΔGenerated files /ΔLOC	2 / 83	15 / 377	1 / 10

Table 8.10: Windows Phone metrics for generated code when adding the survey feature

Table 8.11 presents the metrics for the required custom user code for adding the survey feature to the Obymobi application prototype.

	Amount of files	Lines of code
ΔCustom platform specific code	2	298

Table 8.11: Windows Phone metrics for custom platform specific code when adding the survey feature

Extending the Obymobi prototype with the survey feature is implemented using the following number of lines of code: Based on the metrics presented above, 470 lines of codes

Component	Lines of code
User-Interface	83
Persistent Storage	377
Web Services	10
Custom Code	298
Total	768

Table 8.12: Numbers of lines of code for extending the Obymobi application with the survey feature on the Android platform

can be generated of the total of 768 lines of code. This means that 61% of the source code is generated for extending the Obymobi prototype with the survey feature on the Windows Phone platform.

8.3.3 Summary

The previous sections of this thesis presented an overview of the metrics when developing the initial release of an mobile application compared to developing a new feature for an existing application. By comparing the number of lines of code (LOC) which can be generated using Voind with the number of LOC which have to be written be by hand, a percentage can be determined which represents the ratio between generated code and manually developed code.
As Android does not have native support for web services and persistent storage, 75% of the source code could be generated when developing the initial release of the Obymobi application. When a new feature was added to the Obymobi application, 71% of the source code could be generated.
As Windows Phone does have native support for web services, the percentage of code that could be generated is much lower than on the Android platform as web services are supported natively on Windows Phone. 48% of the source code for an initial release of the Obymobi application could be generated for the Windows Phone platform. When the Obymobi application for the Windows Phone platform was extended with a new feature, 61% of the source code could be generated.

8.4 Reflection on Obymobi Requirements

A reflection on the development of the Obymobi application using Voind is presented below, which is based on the requirements of the Obymobi application specified in Chapter 2.

Obymobi should run on all mobile platforms in order to cover a marketsegment as large as possible
The scope of this thesis project allowed the development of a cross-platform mobile application framework prototype which can be used to generate code with for the mobile platforms Android and Windows Phone 7. However, as the support for different operating contexts can be extended in the Voind application, more platforms can be added without having to extend the Voind application itself.

Obymobi development should follow the write once, run everywhere paradigm
The most important artifacts of a mobile application can be defined using higher-level specifications, which are used in the code generation process to generation code per platform with. This does not completely adhere to the write once, run everywhere paradigm as the defining business logic of an application a in generic fashion is a very complex task.

Obymobi should be able to use device-specific features such as phonebook, GPS, etc.
As the injection of user defined code is possible using Voind, device-specific features can be implemented, although that requires platform specific programming knowledge. Based on the case study for the development of the Obymobi application, 71% - 75% of the source on the Android could be generated using Voind. For the Windows Phone platform, 48% - 61% could be generated.

Obymobi should have a native look-and-feel
As the Voind framework generates 100% native code, applications generated using Voind will have a native look-and-feel by default.

8.5 Limitations of Voind

The following limitations of the Voind applications have been pointed out in during the Obymobi case study:

- Programming knowledge of multiple platforms is still needed to develop in order to write specific logic and setup a development project.
- Business logic and device specific features can not be persisted dynamically in a generic model yet. Therefore, those have to implemented by hand or using code snippets in Voind.

Chapter 9

Discussion and Conclusion

9.1 Cross-Platform Development Issues

When developing applications for different operating contexts i.e. platforms, developers have to deal with the various diversities of certain platforms. Third party application developers are often confronted with the issues related to developing applications for multiple platforms. Separate development of an application per implementation platform is one option, however this requires programming knowledge of all implementation platforms and often results in applications getting out-of-sync during development. The 'write once, run everywhere' paradigm tends to be the way to go for cross-platform development, however existing solutions which follow this paradigm have other issues that should be considered when a cross-platform development solution is needed.

The main issue in cross-platform mobile application development is the operating context diversity when developing native applications. Different platforms offer a different set of features which makes it hard to create a generic abstraction of those platform without losing platform specific features in the abstracted model. Another issue is the difference in screen resolutions for the various platforms, which implies that sizing and positioning should be done in a relative matter to the screen resolution. Some platforms like Android do offer support for relative sizing and positioning opposed to other platforms like Windows Phone 7 which does not.

Several tools for cross-platform mobile application development exist in industry today, however the most promising tools are all focussed on delivering hybrid applications in which the user-interface is specified using HTML and is rendered in a native web browser during runtime. Although these tools adhere to the 'write once, run everywhere' paradigm, access to device-specific features are limited to the features implemented in such a cross-platform development tool. As the user-interface is rendered into a web browser, performance is not as good as it would be using a native user-interface.

Generating native code per implementation platform is a tedious task, especially because the differences in user-interfaces of several platforms are hard to abstract in a generic model.

9.2 Cross-Platform Code Generation using Voind

As the current cross-platform solutions from industry today focus on delivering hybrid application, the development of the Voind application was focussed on delivering native applications.

9.3 Contributions

This Voind application opposed to existing cross-platform development tools contributed in the following ways:

Dynamic
Dynamic in the matter that the output code generation can be influenced. For example, by specifying own library types.
Voind prototype has proved that is it possible to define an application in a generic way using a dynamic abstraction model.

Extensible
An advantange of Voind opposed to existing cross-platform development solutions from industry today is that it can easily be extended when new platform features are released as the domain models can be extended by the end-user. This also allows that the model can be extended by a community.

Complete
Voind covers not only the definition of user-interfaces, but is also focussed on generating other parts of mobile applications like webservice connectivity and persistent storage.

Unique selling points of Voind are:

- Extensibility for more platforms
- Extensibility for more features by extending snippets
- Customizable by editing templates
- Webapplication c.q. webbased, so can be used from every platform
- Not only focussed on user-interface, but also webservices and models

9.4 Conclusions

A reflection on the research questions of this thesis is presented below:

What current solutions for cross-platform mobile application development exist?
As we've seen during this thesis project, there are several ways of doing cross-platform development for mobile applications nowadays. Although there are some promising tools in industry today, they are mostly likely to focus on delivering hybrid applications in which an application is merely as shell for a web browser to display a mobile webpage in.

How can development effort be minimized while maintaining multiple versions of an application for different platforms?
The Voind application developed during this thesis project focusses on delivering 100% native applications by generating application artifacts into platform specific code. By dynamically storing domain models into the application it overcomes the static character of existing cross-platform development solutions as it can be extended by the end-user. However, as defining the business logic of an application using abstractions is a complex task in order to generate a complete application, some programming knowledge per targeted platform is required.
Using the development of the Obymobi application as a case study, experiments showed that 48% to 75% of the source code could be generated using Voind for an initial release of an application. When adding a new feature to an application, 61% to 71% of the source code could be generated. Therefore, generating code using Voind saves development time and is less error phrone.

How is it possible to make use of device-specific features while developing a cross-platform application?
Although current cross-platform development solutions offer support for device-specific features per platform, this support is limited an does not provide full access to the device. Native development does offer support for device-specific features as native code is generated which can be extended by the end-user.

How to deal with mobile device heterogeneity during cross-platform mobile application development?
Eventually, cross-platform development nowadays comes down to a trade-off between native development for different platforms opposed to using a cross-platform development tool which delivers hybrid applications. As mentioned in [9], hybrid applications are cheaper to develop and deploy than native applications, but can they match the native user experience? Advantages of existing tools are that it's often cheaper to develop a hybrid application opposed to develop several native applications for different platforms. Time-to-market is also a benefical factor, which is more likely to be fast for hybrid applications than for native applications. However, hybrid applications do have drawbacks as they do not offer full access to the device it runs on and does not have a native look-and-feel. Performance is also better for native applications, which is an important factor in terms of usability.

9.5 Future work

The following extensions to the Voind application did not fit in the scope of this project but can be used as a reference for future work when extending the application.

- Generation of empty development projects per supported platform
- Better editor (syntax highlighting) for the templates and custom code, intellisense etc.
- Additional cross-platform (i.e. Java) desktop application so generated code can directly be written to disk
- Extension of supported user-interface components
- Definition of device-specific features in higher-level specifications
- Generation of code for persistent storage on the device (OR-mapping)
- Generating / updating database structures per platform using the higher-level specification
- Generation of deployable, binary files

Bibliography

[1] Microsoft software developer network.

[2] Phonegap. http://www.phonegap.com/.

[3] *Essential XUL programming*. John Wiley and Sons Ltd., Chichester, UK, 2001.

[4] Rhodes specification, 2009. http://wiki.rhomobile.com/index.php/Rhodes_Specification.

[5] P. Alessi. *Professional IPhone and IPad Database Application Programming*. Wrox Professional Guides. John Wiley & Sons, 2010.

[6] Sarah Allen, Vidal Graupera, and Lee Lundrigan. *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*. Apress, Berkely, CA, USA, 1st edition, 2010.

[7] Vander Alves, Ivan Cardim, Heitor Vital, Pedro Sampaio, Alexandre Damasceno, Paulo Borba, and Geber Ramalho. Comparative analysis of porting strategies in j2me games. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 123–132, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Peter Braun and Ronny Eckhaus. Experiences on model-driven software development for mobile applications. In *ECBS '08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 490–493, Washington, DC, USA, 2008. IEEE Computer Society.

[9] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Commun. ACM*, 54:49–53, May 2011.

[10] Krzysztof Czarnecki. Overview of generative software development. In *International Workshop on Unconventional Programming Paradigms*, volume 3566 of *Lecture Notes in Computer Science*, pages 326–341, Le Mont Saint Michel, France, September 2004.

[11] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, June 2000.

[12] Victoria Davis, Jeff Gray, and Joel Jones. Generative approaches for application tailoring of mobile devices. In *ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference*, pages 237–241, New York, NY, USA, 2005. ACM.

[13] Nicholas Rogness Department and Nicholas Rogness. An assessment of design and implementation trade-offs and their impact on mobile applications, 2003.

[14] Bertalan Forstner, Laszlo Lengyel, Tihamer Levendovszky, Gergely Mezei, Imre Kelenyi, and Hassan Charaf. Model-based system development for embedded mobile platforms. In *MBD-MOMPES '06: Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 43–52, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Wolfgang Frank. iphonical, model driven software development for the iphone. World Wide Web publication, 2009.

[16] Gartner. gartner. `http://www.gartner.com/it/page.jsp?id=1689814`.

[17] Chris Greenhalgh, Steve Benford, Adam Drozd, Martin Flintham, Alastair Hampshire, Leif Oppermann, Keir Smith, and Christoph von Tycowicz. Addressing mobile phone diversity in ubicomp experience development. In John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang, editors, *Ubicomp*, volume 4717 of *Lecture Notes in Computer Science*, pages 447–464. Springer, 2007.

[18] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM.

[19] Appcelerator Inc. Appcelerator developer center. `http://developer.appcelerator.com/`.

[20] Apple Inc. ios dev center. `http://developer.apple.com/devcenter/ios/`.

[21] Google Inc. Android developers. `http://developer.android.com/`.

[22] Catherine Keynes. Operating systems 101, 2009.

[23] kSOAP. ksoap. `http://ksoap2.sourceforge.net/`.

[24] Carsten Magerkurth and Thorsten Prante. Towards a unifying approach to mobile computing. *SIGGROUP Bull.*, 22(1):16–21, 2001.

[25] Reto Meier. *Professional Android 2 Application Development*. Wrox Press Ltd., Birmingham, UK, UK, 1st edition, 2010.

[26] Hiroshi Nishimura and Chris Timossi. Mono for cross-platform control system environment, 2006.

[27] Earl Oliver. A survey of platforms for mobile networks research. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(4):56–63, 2008.

[28] Steve R. Palmer and Mac Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 1st edition, 2001.

[29] GOLD Parser. goldparser. `http://www.devincook.com/goldparser`.

[30] Damith C. Rajapakse. Techniques for de-fragmenting mobile applications: A taxonomy. In *SEKE*, pages 923–928. Knowledge Systems Institute Graduate School, 2008.

[31] Axel Spriestersbach and Thomas Springer. Quality attributes in mobile web application development. In *PROFES*, pages 120–130, 2004.

[32] Janos Sztipanovits and Gabor Karsai. Generative programming for embedded systems. In *PPDP '02: Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 180–180, New York, NY, USA, 2002. ACM.

[33] Chris Thompson, Jules White, Brian Dougherty, and Douglas Schmidt. Optimizing mobile application performance with modeldriven engineering. In Sunggu Lee and Priya Narasimhan, editors, *Software Technologies for Embedded and Ubiquitous Systems*, volume 5860 of *Lecture Notes in Computer Science*, pages 36–46. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-10265-3_4.

[34] Jules White and Douglas C. Schmidt. Model-driven product-line architectures for mobile devices.

[35] WSDL2Java. wsdl2java. `http://ws.apache.org/axis/`.

[36] WSDL2ObjC. wsdl2objc. `http://code.google.com/p/wsdl2objc/`.

[37] Pei Zheng and Lionel Ni. *Smart Phone and Next Generation Mobile Computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[38] Pei Zheng and Lionel M. Ni. Spotlight: The rise of the smart phone. *IEEE Distributed Systems Online*, 7(3), 2006.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

ADT: Android Development Tools

API: Application Programming Interface

APK: Android Package

CPU: Central Processing Unit

CSS: Cascading Style Sheets

DAO: Data Access Object

DEX: Dalvik Executable

DFA: Deterministic Finite Automaton

DSDM: Dynamic System Development Method

DSL: Domain Specific Language

DSM: Domain Specific Modeling

ERB: Embedded Ruby

FDD: Feature Driven Development

HTML: HyperText Markup Language

IDE: Integrated Development Environment

LALR: Look Ahead Left-to-right Right-deriviation

LOC: Lines of Code

MVC: Model-View-Controller

NDK: Native Development Kit

OR: Object-Relational

RAD: Rapid Application Development

SDK: Software Development Kit

SOAP: Simple Object Access Protocol

UI: User Interface

WSDL: Web Services Description Language

XAML: eXtensible Application Markup Language

XML: eXtensible Markup Language

XP: Extreme Programming

XUL: XML User Interface Language