Automatic Code Generation for SATIN Platform

Migrating SATIN Apps to Native Environments Using Titanium Appcelerator

Daniel Buldón Blanco
2014

Master of Science (120 credits)
Computer Science and Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Automatic code generation for SATIN platform

Migrating SATIN apps to native environments using Titanium Appcelerator

Daniel Buldón Blanco
AUTOMATIC CODE GENERATION FOR SATIN PLATFORM
Migrating SATIN applications into native environments using Titanium Appcelerator

Daniel Buldón Blanco

Luleå University of Technology

Computer Science, Electrical and Space Engineering
Luleå University of Technology
El proyecto desarrollado durante el periodo de enero a junio de 2014 en la Universidad de Luleå tiene como objetivo la conversión automática de aplicaciones web generadas automáticamente mediante la plataforma SATIN a entornos nativos, concretamente iOS y Android.

SATIN es un proyecto creado en 2009 que involucra personal investigador y profesores de varias universidades de Suecia junto con empresas del sector tecnológico, como CodeMill. El propósito con el que fue creado fue el de investigar y desarrollar herramientas que facilitasen al usuario final la creación de servicios móviles. Para ello desarrollaron la plataforma SATIN, la cual permite diseñar y crear aplicaciones web a usuarios sin conocimientos previos de programación.

Esto se consigue a través de un editor gráfico basado en componentes. Para crear una nueva aplicación, el usuario sólo tiene que elegir los componentes que le gustaría incluir, y unirlos convenientemente. El resultado de este proceso es una aplicación web (página web programada en HTML5, JavaScript y CSS) que será alojada en los servidores de SATIN y que será accesible a través de su url, la cual le será proporcionada al usuario al finalizar la creación de la aplicación.

El propósito del proyecto que he desarrollado es, por tanto, convertir las aplicaciones web generadas utilizando la plataforma SATIN en aplicaciones nativas para iOS y Android de forma automática.

El hecho de poder generar aplicaciones nativas automáticamente a partir de aplicaciones web existentes representaría numerosas ventajas para la plataforma SATIN. En primer lugar, cabe señalar que las aplicaciones web no son fácilmente comercializables debido a que normalmente están alojadas en servidores de terceros, y no existe un mercado preparado para este tipo de negocio. Sin embargo, las aplicaciones nativas para dispositivos móviles tienen un gran mercado. Por
tanto, un usuario de dicha plataforma podría no sólo crear sus propias aplicaciones, sino también comercializarlas, sin necesitar conocimientos de programación.

Por otro lado, el hecho de generar aplicaciones nativas permitiría utilizar de forma más eficiente las APIs y características nativas de cada entorno.

El trabajo se ha organizado en tres bloques:

- Investigación de las plataformas y *frameworks* comerciales que pudieran ser utilizados para cumplir la meta del proyecto. Se valoró especialmente que fueran gratuitos y que estuvieran respaldados por una comunidad de desarrolladores.

- Desarrollo de un programa que migrase las aplicaciones web creadas con la plataforma SATIN a iOS y Android. En este punto era especialmente importante diseñar el programa de tal forma que fuera fácilmente integrable en SATIN.

- Análisis de los resultados y comparatira con la herramienta seleccionada por Manuel Cerrillo Bermejo en su proyecto.

**Proceso de investigación y selección de herramientas**

Como resultado del proceso de investigación, se seleccionó Titanium Appcelerator como *framework* que se utilizaría en el proceso de conversión de la aplicación web.

Titanium es un kit de desarrollo de software (SDK) pensado para el desarrollo de aplicaciones para diferentes plataformas. El desarrollador solo tiene que escribir la aplicación una vez utilizando HTML5, CSS y JavaScript, y utilizando este SDK, es capaz de generar aplicaciones nativas en todas las plataformas soportadas.

En este momento, Titanium soporta el desarrollo de aplicaciones sobre las siguientes plataformas:

- Android
- IOS
- Tizen
- Blackberry

**Desarrollo del programa**

El lenguaje escogido para desarrollar el programa ha sido Java. La principal razón para escoger este lenguaje ha sido el soporte multiplataforma del que disponen los programas diseñados con él.

La plataforma SATIN está en continuo desarrollo, por lo que se quería asegurar que incluso ante cambios importantes, como una migración del servidor a otro sistema operativo o cambios en la versión del mismo, el programa diseñado
seguiría funcionando correctamente. Igualmente, para migrar las aplicaciones web a plataformas iOS, es necesario disponer de un Mac o bien de un sistema con Mac OS X virtualizado, ya que es la única forma de generar aplicaciones para dicho sistema. Por tanto entre los requisitos que se debían cumplir para el programa estaba el soporte en múltiples plataformas.

Para lograr la conversión de aplicaciones web, era necesario utilizar las características de Titanium SDK que permiten el desarrollo multiplataforma. El programa acepta los siguientes parámetros:

- Nombre de la aplicación.
- Url de la aplicación.
- Plataforma en la que se va a desplegar: android o ios.

También se ofrece al usuario la posibilidad de utilizar su propia clave para firmar las aplicaciones generadas (en el caso de aplicaciones Android). Para ello, se puede llamar al programa con tres parámetro más:

- Alias de la clave.
- Localización del almacén de claves (keystore).
- Contraseña del almacén de claves.

El flujo de ejecución del programa es como sigue:

- En primer lugar se crea un nuevo proyecto para una aplicación utilizando Titanium SDK

- Una vez se ha creado el proyecto para la aplicación, se procede a modificar el código de la misma para que cargue el contenido de un archivo html que contendrá nuestra aplicación web.

- Posteriormente se descarga el código de la aplicación web, así como todos los recursos necesarios para su funcionamiento. Para ello se utilizan librerías especializadas. Una vez se ha descargado, se modifica el código fuente de la aplicación para apuntar a los recursos locales, y se guarda con el formato necesario para ser cargado correctamente.

- Por último, se despliega la aplicación en una de las plataformas soportadas: Android y iOS.

Como resultado de la ejecución de dicho programa, se van a obtener dos posibles salidas, dependiendo de la plataforma de despliegue seleccionada:

- Android: En este caso se obtendrá la aplicación (archivo .apk) firmada con una clave por defecto de Titanium, o con la clave indicada por el usuario al ejecutar el programa. También se obtendrá el archivo correspondiente sin firmar.
iOS: El resultado será una estructura de proyecto de Xcode. Será necesario, por tanto, abrir dicho proyecto con Xcode, compilarlo, y firmarlo con el AppleID del usuario.

Análisis de resultados

Para analizar el comportamiento de las herramientas utilizadas se crearon una serie de aplicaciones pensadas para probar ciertas funciones básicas de un dispositivo móvil. Las aplicaciones creadas fueron las siguientes:

- “Timer” → Aplicación básica que genera un mensaje de alerta cuando finaliza un contador.
- “Random Number Generator” → Genera números aleatorios y los representa dinámicamente en una gráfica. Permite probar el comportamiento ante interfaces más elaboradas con cambios dinámicos.
- “My Position” → Muestra la posición del usuario en un mapa.
- “Shake App” y “FacesUpDown” → responden a cambios de posición y al movimiento del dispositivo con diferentes mensajes de alerta.

Tras probar dichas aplicaciones, se llegaron a las siguientes conclusiones:

- Las aplicaciones generadas no son puramente nativas, sino aplicaciones híbridas en las que la aplicación web está empaquetada sobre una base nativa. Las aplicaciones se muestran como si fueran ejecutadas sobre un navegador, lo que puede provocar que ciertas funciones del dispositivo no sean accesibles a la aplicación.

- El rendimiento de las aplicaciones generadas es aceptable, especialmente en el caso de interfaces dinámicas.

- Es posible acceder a la localización, posición GPS orientación y acelerómetro del dispositivo.

- Es posible generar automáticamente aplicaciones híbridas a partir de las aplicaciones web diseñadas con SATIN.

Estos resultados se compararon con los obtenidos por la herramienta seleccionada (Apache Cordova) por Manuel Cerrillo Bermejo para la realización de su proyecto.

Como resultado de esta comparación, se pudo ver que Apache Cordova es más eficiente a la hora de crear las aplicaciones nativas, creando aplicaciones más ligeras. En cuanto a las demás características, ambas herramientas son similares, y no se aprecian diferencias entre las aplicaciones generadas con Titanium Appcelerator y las generadas con Apache Cordova en materia de rendimiento y características accesibles.
To my family and friends

for making it possible
ABSTRACT

The development of web applications has become very important in the last years. The use of smartphones and mobile devices which keep connected to the Internet permanently has also grown, leading to an increasing demand of applications for those devices. This has resulted in the creation of several platforms that allow the user to create his own applications easily, with no need of programming skills. One of this platforms is SATIN. SATIN Platform allows the user to create responsive web applications that, once created, are stored in SATIN’s servers, and may be accessed via their urls from any kind of device, mobile or not, no matter which architecture or operating system is running.

Web applications become a powerful and elegant solution to avoid the problems of cross-platform development, but have a problem that, up to now, is still unsolved. This problem is the difficulty of selling those web applications, as there is no appropriate market, nor business plans to commercialize them. This leads to the necessity of migrating those web applications into native environments in order to be able to commercialize the products developed with SATIN Platform.

The purpose of this thesis is, then, to study the possibility of integrating a commercial tool like Titanium Appcelerator, which automatically generates native applications from JavaScript code, within the SATIN Platform. In this way, a final user without a deep knowledge in programming applications would be able to develop and commercialize applications according to his needs. It would be also possible for enterprises to have a tool to create easily and fast new applications for their business.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESUMEN</td>
<td>4</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>10</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>14</td>
</tr>
<tr>
<td>1.1 Aim of the project</td>
<td>14</td>
</tr>
<tr>
<td>SATIN PROJECT</td>
<td>15</td>
</tr>
<tr>
<td>2.1 Component Portal</td>
<td>15</td>
</tr>
<tr>
<td>2.2 SATIN Editor</td>
<td>16</td>
</tr>
<tr>
<td>APPCELERATOR</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Titanium</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1 Alloy MVC framework</td>
<td>20</td>
</tr>
<tr>
<td>3.1.2 Structure of the Classic Titanium project</td>
<td>22</td>
</tr>
<tr>
<td>REQUIREMENTS</td>
<td>24</td>
</tr>
<tr>
<td>4.1 Supported Operating Systems for Titanium SDK</td>
<td>24</td>
</tr>
<tr>
<td>4.2 Required software</td>
<td>24</td>
</tr>
<tr>
<td>4.2.1 Java Development Kit</td>
<td>24</td>
</tr>
<tr>
<td>4.2.2 Node.js</td>
<td>24</td>
</tr>
<tr>
<td>4.2.3 Android Software Development Kit</td>
<td>25</td>
</tr>
<tr>
<td>4.2.4 Xcode</td>
<td>25</td>
</tr>
<tr>
<td>PROPOSED SOLUTION</td>
<td>26</td>
</tr>
<tr>
<td>5.1 Structure of the solution</td>
<td>26</td>
</tr>
<tr>
<td>5.1.1 Java program</td>
<td>26</td>
</tr>
<tr>
<td>5.1.2 Titanium</td>
<td>26</td>
</tr>
<tr>
<td>DEVELOPMENT OF THE PROPOSED SOLUTION</td>
<td>27</td>
</tr>
<tr>
<td>6.1 JAVA</td>
<td>27</td>
</tr>
<tr>
<td>6.1.1 Java Virtual Machine</td>
<td>27</td>
</tr>
<tr>
<td>6.1.2 Java Class Loader</td>
<td>28</td>
</tr>
<tr>
<td>6.1.3 Motivation for using Java in the project</td>
<td>28</td>
</tr>
<tr>
<td>6.2 Tools used</td>
<td>29</td>
</tr>
<tr>
<td>6.2.1 Android Developer Tools</td>
<td>29</td>
</tr>
<tr>
<td>6.2.2 Titanium Studio</td>
<td>29</td>
</tr>
</tbody>
</table>
6.2.3 Titanium Command-Line Interface ................................................................. 29
6.3 Structure of the program .................................................................................... 29
  6.3.1 Main.java ........................................................................................................ 30
  6.3.2 BuildManager.java ......................................................................................... 30
  6.3.3 CommandExecutionHelper.java ................................................................... 31
  6.3.4 ProcessOutputManager.java ......................................................................... 31
6.4 Running the program .......................................................................................... 32
6.5 Output obtained.................................................................................................... 32
  6.5.1 Android application ....................................................................................... 32
  6.5.2 iOS application ............................................................................................. 32
RESULTS..................................................................................................................... 33
  7.1 Examples of applications .................................................................................. 33
  7.2 Analysis of the results ..................................................................................... 36
COMPARISON WITH APACHE CORDOVA .............................................................. 38
CONCLUSIONS AND FUTURE WORK .................................................................... 39
REFERENCES............................................................................................................ 40
APPENDIX 1: INSTALLATION GUIDE .................................................................... 42
  1.1 Installation of Oracle Java Development Kit .................................................... 42
    1.1.1 Linux .............................................................................................................. 42
    1.1.2 Mac OS X .................................................................................................... 43
    1.1.3 Windows ..................................................................................................... 43
  1.2 Installation of Node.js ....................................................................................... 43
    1.2.1 Linux .............................................................................................................. 43
    1.2.2 Mac OS X .................................................................................................... 44
    1.2.3 Windows ..................................................................................................... 44
  1.3 Installation of Android SDK ............................................................................. 44
    1.3.1 Linux .............................................................................................................. 44
    1.3.2 Mac OS X .................................................................................................... 44
    1.3.3 Windows ..................................................................................................... 45
  1.4 Installation of Xcode.......................................................................................... 45
  1.5 Installation of Titanium SDK ........................................................................... 45
  1.6 Configuration of Titanium CLI ......................................................................... 46
TABLE OF FIGURES

Figure 1. Component Portal ................................................................. 15
Figure 2. SATIN Editor ..................................................................... 16
Figure 3. Titanium Command-Line Interface ......................................... 18
Figure 4. Titanium Studio project templates ......................................... 19
Figure 5. Model-View-Controller pattern ............................................. 20
Figure 6. Folder structure of an Alloy project ....................................... 21
Figure 7. Folder structure of the default project ...................................... 22
Figure 8. Java architecture .................................................................. 28
Figure 9. Screenshot of Timer application ........................................... 33
Figure 10. Design of Graphical Random Number Generator app ............ 34
Figure 11. Screenshot of Graphical Random Number Generator application 34
Figure 12. Screenshot of My Position application .................................. 35
During the last few years the mobile industry has grown rapidly reaching a huge number of users. With more people consuming mobile products, specially applications, the necessity to offer more personalized services in order to fulfill in a more accurate way all the needs of the customer has also increased.

As a response to this increment in the demand of these products, several services intended to let the user create his own applications have been launched.

On the other hand, some other companies have focused on simplifying the process of creating a new application, making it possible to build applications for several native platforms from the same base code.

The intention of this thesis is to join both visions, creating a program that will allow the user to migrate a web application designed and created using SATIN Platform into Android and iOS environments.

1.1 Aim of the project

The aim of this project the study of the available platforms suitable for migrating web applications into native ones and the integration of one of them into the SATIN Platform.

Adding the possibility to build a native application greatly broadens the extent of the SATIN Project, enabling users to sell the applications that they create.

In this project we will create a program that will take the web application generated by SATIN Platform and will migrate it into Android and iOS.

However, the SATIN Platform is still under development, as it is the number of components available. This means that the applications that can be developed at the moment will not be as complex as they could be if they were developed directly for a native platform, but will be complex enough to test the availability of the tools under study.
The SATIN project started in 2009 with the purpose of approaching the design and creation of mobile services to those users with a lack of programming knowledge or skills.

They developed the SATIN Platform: a toolkit that allows the end user to build his own web applications in an easy way.

The SATIN Platform core consists of a growing number of components, each of them having a simple functionality. By linking these components we can easily create mobile applications.

2.1 Component Portal

The Component Portal is the tool provided by SATIN to create and integrate new components into the SATIN Project.

As we said previously, components are the main part of the SATIN Platform. We could imagine a component as a method or function in any other programming language. These components are intended to have simple functions. In fact, the simplest it is a component, the easiest it is to maintain and use. As methods, the components have inputs and outputs, which are defined in the creation of the component.

New components must be described in a XML file according to a certain guideline to prevent errors and unexpected behaviors. In this XML file it will be indicated, among other parameters, the number of inputs and outputs that the new component will have. If we continue with the previous analogy, these inputs and outputs would be respectively the arguments and the return value of the method.

Components source code is written using JavaScript, due to the versatility and portability of this language, as well as the great support that it has in mobile services. Once the XML description and the code of the component is done, it can be submitted and used for building applications.

Figure 1. Component Portal
2.2 SATIN Editor

The SATIN Editor is the drag-and-drop editor used by the end user to create the new applications. It contains a list of the available components classified into several categories. These components can be dragged into a canvas that will contain all the components of the application, and linked to others.

The application result of this process is stored in SATIN servers and can be accessed via its URL from different platforms: PC, tablet, smartphones. (1) (2) (3) (4) (5) (6) (7)
APPCELERATOR

In this section we will present the platform analyzed and used during this project

Appcelerator Inc. is a company focused on mobile technology which offers two main products:

- Titanium: open source software development kit for cross-platform development. It is the framework used during this thesis.

- Appcelerator Platform: Thought as an enterprise product, adds to the main features of Titanium other tools for getting analytics, deploying and testing the applications. (8)

3.1 Titanium

Titanium is a cross-platform software development toolkit which provides APIs for iOS, Android, Windows, Tizen, Blackberry and HTML5.

This SDK makes possible the creation of complex native application using only JavaScript code, and deploying into one of the supported platforms.

Requires a free registration for being able to download and use it.

Appcelerator provides two different ways of using Titanium SDK.

- Titanium Studio: It is an IDE based on Eclipse and Aptana which facilitates the creation of mobile, web and desktop applications. It supports HTML5, JavaScript, CSS, PHP5, Ruby-on-Rails and Python.

  This IDE has only been used in this project to test Titanium SDK and create simple applications in order to learn the operation of the SDK. It is not necessary for this project, as it only provides a graphical interface to access the features of the SDK, and a programming environment to create new applications. (9)
Titanium Command-Line Interface (CLI). This interface allows the user to manage any Titanium projects from the terminal. This is the method used in this project to access the features of Titanium SDK needed for migrating web applications into native ones.

Figure 3. Titanium Command-Line Interface
At the moment of creating a new mobile application project, the user will be given the option of selecting among several types of project, some of them including templates to ease the development of the new application. The user will be able to choose one of the following templates between one of the two available project structures: Classic or Alloy.

- **Classic**
  - Default project: no template. This is the one used in this project.
  - HTML-based Application: Used for applications that will use just an HTML view.
  - Master/Detail Application: Meant to be the starting point for the development of applications with hierarchical data or stack of windows.
  - Single Window Application. Application with only one view, mostly a blank canvas.
  - Tabbed Application: Application with several windows organized in tabs.

- **Alloy**
  - Default Alloy Project: Hello World application developed using Alloy framework
  - Two-Tabbed Alloy Application: Application with several windows organized in tabs, developed using Alloy framework
3.1.1 Alloy MVC framework
Titanium Alloy is a framework that facilitates the creation of complex applications following the Model-View-Controller (MVC) design pattern. (11)

Most of the complex web applications developed nowadays are designed and built using this MVC pattern as a base, making it possible to divide the application into three parts:

![Model-View-Controller pattern](image)

Figure 5. Model-View-Controller pattern (12)

**Model**
Contains and manages the data objects of the application. It handles the accesses to the data, either consults or modifications, checking permissions and implementing the access privileges required for the application.

It receives the commands from the Controller indicating which data needs to be accessed, and the Model sends to the View the required data to be displayed. (12)

**View**

Presents the Model to the user via a User Interface (UI). (12)
Controller

Answers to events mostly triggered by the user. It sends requests to both the Model and the View to request changes in the way that the data is displayed, or to request changes in the data model. (12)

Structure of Alloy project

As we can see in Figure 6, the project will contain separate folders for models, views and controllers of the application.

A requirement in any Titanium project is the configuration file named “tiapp.xml”, which will be explained deeply in the next points.

Consideration about Alloy regarding the project

Alloy is a powerful framework totally integrated into Titanium that can lead to a faster and easier development of applications.

Although Alloy framework could have been used for migrating web applications into native ones, it is based in Model-View-Controller pattern. This makes it powerful, and easy to use, but not suitable for our project, due to the fact that we have to use the output of the code generation process carried by SATIN servers in order to create a web application. This means, then that we have to work with the HTML code of the application, together with the resources needed for it to work properly.
Structuring all these resources into a MVC structure and linking them again correctly could have been possible, but very costly. Due to this big disadvantage, it was decided not to use Alloy framework, but use instead the Classic default project template provided by Titanium.

Using this template allowed us to work with the resources of the web application in a much easier way.

3.1.2 Structure of the Classic Titanium project

A basic Titanium project has the following structure:

![Folder structure of the default project](image)

There are two main files that any Titanium project must contain. These files are the configuration file “tiapp.xml” and the file “app.js”, containing the application code.

**Tiapp.xml**

This is a configuration file needed by Titanium. Contains the following information:

- guid: global unique identifier used by Appcelerator to manage the statistics collected for the application.
- Id: the application id. In this project, the id of all the applications created will be com.satin.appname.
- Name: Name of the application as it appears once installed on the mobile device.
- Analytics: whether or not the collection of statistics is allowed.
- Deployment-target: platforms in which the project could be deployed.
- Icon: indicates the file containing the icon of the application.
- Publisher: the publisher of the application
- SDK-version: version of the SDK needed.
Most of this values are selected automatically by Titanium platform. In the case of the publisher, it will include the name or company whose account is being used to build the application. (13)

**App.js**

This is the file containing the Titanium application that will be migrated into a native platform.

Titanium provides a big amount of APIs to access all the features of a mobile device, resulting in a very powerful tool for developing new applications. However, in this project we have to work with applications that were not created using these APIs. To use all the potential of Titanium SDK it would be necessary to rewrite the web application using Titanium APIs.

As this is not viable, a different approach is needed. The application that we have to migrate contains basically HTML, JavaScript and CSS. A solution to the problem would be to load the whole application as a “web view”, which displays the content of a web page.

With this solution, the application does not need to be rewritten, but it is just wrapped into a web view, missing the wide field of possibilities offered by Titanium SDK.
In order to run Titanium Software Development Kit, it is necessary to fulfill several prerequisites.

4.1 Supported Operating Systems for Titanium SDK

In the following table we can find the supported OS, as well as the version.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple Mac OS X</td>
<td>10.8.4 and later</td>
</tr>
<tr>
<td>Windows</td>
<td>7, 8</td>
</tr>
<tr>
<td>Linux Desktop</td>
<td>Ubuntu 12.04 LTS (Precise Pangolin)</td>
</tr>
</tbody>
</table>

4.2 Required software

4.2.1 Java Development Kit

It is important to remark that only Oracle JDK will suffice. Titanium does not work with other versions such as OpenJDK, which can be usually found in Linux machines.

Knowing that, in the following table it is indicated which version of Oracle JDK is needed for each of the supported operating systems.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Minimum JDK version needed</th>
<th>Package Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple Mac OS X</td>
<td>Java 6 (also know as 1.6) revision 10</td>
<td>64 bits</td>
</tr>
<tr>
<td>Windows</td>
<td>Java 6 (also know as 1.6) revision 10</td>
<td>32 bits (x86/i586)</td>
</tr>
<tr>
<td>Linux Desktop</td>
<td>Java 6 (also know as 1.6) revision 10</td>
<td>32 and 64 bits</td>
</tr>
</tbody>
</table>

4.2.2 Node.js

Node.js is needed to run all Titanium components. This package can be download and installed from its official web site. Although there is no indication about minimum Node.js version required, it is highly recommended to update it to its last version once installed. This process will be explained in the next section.
4.2.3 Android Software Development Kit

Android SDK is a set of development tools used to create Android applications. It is used in this project to migrate the web applications into Android environments.

4.2.4 Xcode

Xcode is an IDE developed by Apple, used to develop applications for iOS devices. It only runs on Mac OS X machines, meaning that for migrating web applications into iOS environments it will be necessary to run the program on a Mac device.
5 PROPOSED SOLUTION

5.1 Structure of the solution

The program contains two different parts working together. To migrate the web application into a native environment, Titanium SDK has been used.

However, as more steps are required to have a functional native application, we have written a Java program which access and uses the Titanium SDK and accomplish the rest of requirements. We will briefly talk about them in the following points, and will be analyzed in the following section.

5.1.1 Java program

The Java program runs the required Titanium Command Line Interface (CLI) commands to access the Titanium SDK and create and build the project, and it modifies the content of the file app.js, in charge of wrapping the web application into a web view.

It will also download the source code of the web application, as well as all the required resources, in order to make it available without Internet connection.

5.1.2 Titanium

Titanium SDK handles the creation of the project, creating the folder structure and key files such as the configuration and the app.js files, which will have to be modified, as they contain the default values.

It is used to build the application into the chosen native environment.
The program developed as a solution to migrate SATIN web applications into native environments has been developed over Java.

6.1 JAVA

Java is one of the most extended and widely used programming languages nowadays.

When talking about Java, we must start by saying that it is an Object Oriented (OO) programming language. Object Oriented Programming aims to create and maintain big projects easily, increasing the success probabilities when developing using OO paradigms, as well as improving the reuse of code by creating generic objects that may be used in several projects.

Another of the main characteristics of Java, and one of the reasons why it has become so popular, is the multi-platform support. This basically means that the developer only has to write the program once, but he will be able to run it over different platforms (as it is said in the slogan created by Sun Microsystems, WORA: Write Once, Run Anywhere).

These multi-platform features are provided by the use of the Java Virtual Machine (JVM), one of the fundamental pieces of Java. (18)

6.1.1 Java Virtual Machine

The Java Virtual Machine (JVM) is a process virtual machine, also known as application virtual machine. This virtual machine runs over a process inside a host operating system (OS), and execute only one process. The purpose of this kind of machines is to provide a runtime environment other than the one provided by the OS and hardware, and independent from it.

To obtain this independence from the platform in which we want to run the Java application, the Java source code is compiled, generating in this process Java binary code named “bytecode”, which contains machine instructions for the Java Virtual Machine. This bytecode will run over a JVM, which will interpret it, or will generate native code from it using a Just In Time (JIT) compiler. By using JIT compilers, we achieve greater speed than interpreting the bytecode, and it is the most used method in JVM nowadays.

JVMs are native applications that must be installed specifically for each platform. They include specific libraries to access certain features of the system in which they are running on, such as graphical parameters, or thread management. However, they act as a bridge between the host OS and hardware where the Java application has to run and the Java program that we want to run, making it possible to reach high portability. (18) (19)
6.1.2 Java Class Loader

It is the part of the Java Runtime Environment that handles the class load. Classes are usually loaded into the Java Virtual Machine on runtime, and only if demanded. This increases the efficiency.

Another feature that must be taken into account is the late binding. Java programs have the capacity of changing their data types dynamically or at runtime, adjusting the program size and therefore increasing the efficiency in comparison to early-binding languages such as C or C++.

![Figure 8. Java architecture (21)](image)

6.1.3 Motivation for using Java in the project

We could sum up the reasons that led us to use Java in the following points:

- Platform portability: Since we want a program able to generate both Android and iOS applications, and taking into account that for generating iOS applications you need to have a Mac device, the multi-platform support was a must.

- Java is one of the most used and supported programming languages nowadays, with powerful and wide libraries backing it, which makes it suitable for many different types of project.
6.2 Tools used

6.2.1 Android Developer Tools

The program has been designed and implemented using the Android Developer Tools provided by Eclipse Foundation, which include Eclipse 4.2.1 together with the Android SDK.

6.2.2 Titanium Studio

In order to test the Titanium platform, we have used Titanium Studio, the IDE provided by Appcelerator, which is also a modified version of Eclipse IDE containing all the libraries needed for cross-platform development. Although this IDE also brings the possibility to test on emulated devices, some stability problems of the emulator made us change and test the applications on a real device. For this task we used a Google Nexus 4 smartphone with Android 4.4.2.

6.2.3 Titanium Command-Line Interface

Titanium CLI is used by the Java program to create a new project every time that we want to migrate a new application, and to build the application into a native environment.

6.3 Structure of the program

The program basic flow is as it follows:

- A new Titanium project is created. This will create the folder structure and the app.js file.
- Change the content of the app.js file (which is the default content) for the code needed to wrap the web application into a web view.
- Download all the resources needed to grant that the application will not have to access the internet once installed on the phone. These files will be placed under the Resources folder.
- Download the source code of the web application and save it in a file named src.html.
- Build the application into one of the supported native platforms: Android or iOS.
6.3.1 Main.java

This class contains the main method of the program, and is the one that has to be run. The program needs at least three parameters:

- application name
- url where the application is located
- platform (Android or iOS)

However, it accepts also three more parameters, in case that the user wants to provide its own key to sign the Android application. In that case, the three extra parameters would be:

- keystore path
- keystore password
- alias

This option is only available for Android development. In iOS it is necessary to sign it after creating the application with the key associated to the AppleID.

6.3.2 BuildManager.java

This class contains the main functionality of the program. It does the following tasks:

Creation of the project.

With the given application name, it will execute a Titanium CLI command to create a titanium project.

`Titanium create --workspace-dir <localdir> --id <com.satin.appName> --platforms <desiredPlatform> --name <appName> --no-banner --force`

In case that the execution of this command finishes with a nonzero exit value, it will show the content of both the standard and error outputs in the console, to help debugging.

Modification of app.js.

The file “app.js” contains the code of the Titanium application. In this case, as desired result is to migrate an existing web application into a native one, this file will have to be changed. To achieve the migration, we will include the source code of the web application wrapped as a web view.

Download the source code and the resources.

Using the given application url, this part of the program will download the source code of the application as it may be seen in the web browser. To be able to run the application without
Internet connection, it will download as well all the resources associated to the application, and modify the source code of it to point to the relocated resources.

**Build the application.**

with the platform given as a parameter, it will execute a Titanium CLI command to build the project into the selected platform.

```
titanium build --platform <desiredPlatform> --build-only --force
```

In case the user wants to sign the Android application with its own key, then the command is as it follows:

```
titanium build --platform <desiredPlatform> --build-only --force --keystore <keystore path> --alias <alias> --store-password <keystore password>
```

As it happened in the creation of the project, if this command does not finish correctly, the standard and error outputs will be displayed.

6.3.3 CommandExecutionHelper.java

This class handles the execution of commands over different operating systems transparently for the user. It uses a `ProcessBuilder` object to launch a terminal in the background and run the commands received as a parameter.

It stores the exit value and the content of the standard output and standard error output of the last command executed.

6.3.4 ProcessOutputManager.java

This class was created to prevent blocking when trying to read the outputs, both the standard and the error, of a process.

It works together with the CommandExecutionHelper class, saving the content of the output so it can be reachable for the correct methods in the program.
6.4 Running the program

The program is packed into a .jar file. It can be launched by running the following command:

```
java -jar AppGenerator.jar <AppName> <url> <platform [android | ios]>
```

or

```
java -jar AppGenerator.jar <AppName> <url> <android> <keystore path> <keystore password> <alias>
```

If running on Linux machines, super user permissions are needed.

6.5 Output obtained

The execution of the program will lead to the creation of a folder structure containing the resources and source code of the web application. The build process will create an extra “build” folder, which will contain an “ios” or “android” subfolder depending on the platform chosen when the program was run.

6.5.1 Android application

The program creates an unsigned version of the application, as well as a version signed either with the user’s own key or with the Titanium key. Both of them can be found under the “bin” folder.

In case that the user did not choose to sign the application with its own key, it is possible to do it after the build process using external tools.

6.5.2 iOS application

The result obtained for iOS applications is a Xcode project for this application. This means that it will be necessary to open that project with Xcode to build and sign the application.

For signing the application, an Apple ID will be required.
RESULTS

In this section we will present the results observed analyzing the applications generated using Titanium SDK.

7.1 Examples of applications

Titanium SDK has been tested by creating several applications, each of them has something that makes it interesting in a certain way.

The developed applications are the following

- **Timer**: This application is a basic graphical timer.

![Timer application screenshot](image)

*Figure 9. Screenshot of Timer application*

This application was mainly used to test the build process and behavior of basic components. It worked correctly both on Android and iOS.
- **Graphical Random Number Generator**: This application generates random numbers and displays them in a bar graph, allowing the user to save the data.

![Figure 10. Design of Graphical Random Number Generator app](image)

![Figure 11. Screenshot of Graphical Random Number Generator application](image)

The idea behind this application was to test more complex dynamical graphical interfaces. It was seen that, although it is not as fluent as other natively developed applications, and it takes longer to load and update the graph, it is fluent enough to be used without any problems.
- **My Position:** This application shows the position of the user in a map, and lets the user enable or disable tracking, or update the position manually.

![My Position application](image1.png)

*Figure 12. Screenshot of My Position application*

The purpose of this application was testing the access to GPS and location features of the device.

- **ShakeApp and FacesUpDownApp:** These applications were created in order to test the access to complex features of the mobile device, such as the accelerometer. Both applications display an alert message when a the device is shaked, in the case of the first application, or when it is rotated, in the case of the second. Both applications work correctly. These applications were first tried with sound alerts instead of messages. However, the sound alert did not work.
7.2 Analysis of the results

In this section, we will present the main issues observed in the analysis of the applications generated with Titanium SDK.

The first issue that should be analyzed would be the size of the generated applications. Although in the last years the capacity of smartphones and tablets has increased greatly, including more and more storage capacity and RAM, it is a good practice to try to develop applications taking into account the available resources, designing and building useful and efficient applications.

However, as we are working with automatic code generation tools, we cannot account on the expertise of the developer to polish the application so it does its best using the least resources possible.

The size of the “Timer” and “My Position” applications generated using Titanium SDK is about 8 MB in the .apk file, and slightly higher than 16 MB once installed in the device.

The “Timer” application, being very simple, is as big as the GPS application. Most of the data included in the application are Titanium libraries that should not be necessary to include, making the file to grow bigger that it should be. If we unpack the .apk file using the libraries “Dex2jar”, we will see that those applications include many modules that are not going to use, such as accelerometer modules. (20)

Another problem found that should be pointed out is the way in which Titanium SDK migrates the applications.

Titanium SDK is a framework thought to develop applications for different platforms writing the code only once in JavaScript, instead of migrating existing applications into native environments. The way in which we are using this platform is not the way it is meant to be used, and that leads to results that could be improved.

As it is said previously we use the Titanium SDK as it follows:

- First of all, download the source code of the application, and all the resources, to make it available offline
- After that, wrap both the source code and the resources into a web view.

A web view is a resource used by Titanium to load a web page written in HTML, CSS and JavaScript. This will allow us to load the application as if it was shown in a web browser. However, there is no real native code behind the application logic. The only native code present is the one in charge of loading the HTML file containing the source code of the application.
That leads us to the last point: the extent to which this tool may be used successfully.

As the applications generated with Titanium SDK will be running as if they were displayed in a web browser, there is a limitation to the kind of features that will be able to use.

With the last web browsers, and especially with the use of HTML5, there are a lot of features of mobile devices that can be accessed, such as GPS, and other sensors present in the mobile device. However, we will only be able to use those features that can be accessed from the web browser, due to the fact that the web view used to wrap the application will be an intermediate layer that will prevent the application from interacting directly with the native APIs.

It has been found that applications generated with Titanium Appcelerator can access successfully the GPS and accelerometer of the mobile device.
As it has been said previously, there are many different platforms offering similar services. In this section we are going to compare the results obtained using Titanium Appcelerator with the results obtained using Apache Cordova. The study of Apache Cordova has been taken from the Master Thesis “Code generation for SATIN platform: Migrating SATIN applications to native environments”, written and presented by Manuel Cerrillo Bermejo at the Computer Science, Electrical and Space Engineering department of Luleå University of Technology.

Firstly, we should point that both Apache Cordova and Titanium Appcelerator are free platforms. In order to use Titanium, an Appcelerator account is needed, but it can be created for free.

At the moment, Cordova offers support for more native platforms than Titanium, allowing the user to create applications for Firefox OS and Ubuntu.

According to the results obtained from the analysis of the applications generated using both platforms, it is possible to say that the same web applications generated with SATIN Platform, migrated into Android using both Cordova and Titanium, and tested on the same mobile device have a similar performance. There is no visible difference between the applications generated with Titanium and the ones generated with Cordova regarding the interaction with the user, the load time, and the response time to user events.

However, it is possible to see a big difference in the size of the application. Working with the same SATIN web applications, the native applications generated using Titanium have an .apk file of approximately 8 MB, and about 16 MB once installed in the device. Cordova applications, on the other hand, have an .apk file of 300 KB, and 600 KB once installed. As we can see, there is a big difference in size, that could show that Cordova is better optimized.

As a final result, the applications tested showed that both the applications generated with Titanium and the ones generated with Cordova can access correctly some of the main features of the mobile device. The applications testing the access to the GPS and the accelerometer worked in both cases.
To achieve better results a better integration between SATIN platform and Titanium SDK would be necessary.

Titanium SDK is a very powerful framework and it is only being slightly used. There are still some features that have not been tested in this project, as the databases, and the access to user information, such as messages, contacts or phone calls. It would be necessary to test these features in order to state whether Titanium may be used to develop complex applications or not. At the end of this thesis, it has been proved that Titanium actually works with the main features of the device, specially the physical sensors. However, it still needs further investigation regarding to user data.

Titanium SDK provides complete APIs to work with SQL databases in an efficient way, but as long as we have been working with the source code of the application generated by SATIN platform, it is not possible to use those APIs. To do that, it would be necessary to integrate Titanium SDK in the code generation process of the web application, so it generates a source code 100% compatible with Titanium SDK.

However, it has been proved that in a very first approach, it is possible to migrate from web applications into multiple native platforms. That would mean that a user with no knowledge or skills about programming applications would be able to create applications starting with simple ones, and growing in complexity as the SATIN Platform does, and be able to sell them in multiple markets.
REFERENCES


14. **Appcelerator.** Appcelerator Wiki - Prerequisites. [Online] [Cited: 05 19, 2014.]
https://wiki.appcelerator.org/display/guides2/Prerequisites.


16. **rtCamp.** Installation of Node.js on Linux. [Online] [Cited: 05 19, 2014.]

17. **Walsh, David.** Upgrade Node.js using npm. [Online] [Cited: 05 19, 2014.]
http://davidwalsh.name/upgrade-nodejs.

18. **Oracle.** Oracle - Java Specifications. [Online] [Cited: 05 19, 2014.]
http://docs.oracle.com/javase/specs/.


APPENDIX 1: INSTALLATION GUIDE

This section contains the steps needed to successfully install and configure Titanium SDK over the different supported operating systems.

1.1 Installation of Oracle Java Development Kit

1.1.1 Linux
From a command line, we have to type these commands in order to add the repositories containing Oracle Java, and installing it.

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
```

Then, depending on which version we want to install, type:

```
sudo apt-get install oracle-java7-installer
sudo apt-get install oracle-java8-installer
```

This command will launch the Oracle JDK installer, which will download and configure it in your system.

In case that we install the 64 bits JDK version, the library “ia32-libs” will be needed for correct operation of Titanium SDK. This library may be installed by running the following command:

```
sudo apt-get install ia32-libs
```

Once the installation is finished, we can check the active Java version by running:

```
java -version
javac -version
```

In case that the active version is not the desired one (that may happened if there are several JDK versions installed), we will have to select the needed one. To do that, we will have to run:

```
sudo update-alternatives --config java
```

This will show a list with all the installed version of JDK, and will ask the user to select the one that wants to be active. (15)
1.1.2 Mac OS X
The installation of Oracle JDK is very simple. From the official web site, we have to download the “.dmg” file. After this is done, by double clicking on the file a new window will appear. To install Oracle JDK, we have to click on the .pkg file.

1.1.3 Windows
To install Oracle JDK in a Windows machine, we have to download the installer of the desired version from the official website, and run it. It will guide us through the installation.

1.2 Installation of Node.js

1.2.1 Linux
To install Node.js over a Linux machine, we need to first add the correspondent repositories by running the following commands:

```
sudo apt-get install python-software-properties
dsudo apt-add-repository ppa:chris-lea/node.js
ndsudo apt-get update
```

Once we have added the repositories, we can install Node.js and npm (official package manager for Node.js) by running (16):

```
sudo apt-get install nodejs
```

In order to check the installed version, we should run:
```
node -v
npm -v
```

In case that the installed version is not the desired one, we can update Node.js using npm (17):

```
sudo npm cache clean -f
sudo npm install -g n
sudo n stable
```

This will install the last stable version of Node.js
However, if we need a specific version instead, we can run the following command indicating the desired version:

```
sudo n 0.8.10
```
1.2.2 Mac OS X
To install Node.js on a Mac OS X machine, we have to download the .pkg file from the official website, and run it. This will install Node platform in our system.

1.2.3 Windows
To install Node.js in a Windows machine, we have to download the installer or binary file from the official website, and run it. It will guide us through the installation.

1.3 Installation of Android SDK

1.3.1 Linux
First we have to download the Android SDK from the official web site. After extracting it to the desired folder, we need to add it to the path.

We can edit the .bashrc file to include at the end the following line:

    export PATH={PATH}:/home/{user}/Development/android-sdk-linux/tools

We have to point to the tools folder of the SDK.

After saving the changes, we must close the terminal.

Once installed, we need to configure it. To launch the Android SDK Manager we have to run the command:

    android

This will open a window from where we will be able to select which levels of the API we would like to install.

1.3.2 Mac OS X
From the official website it is possible to download a compressed file containing the SDK. After extracting and placing it into the desired location in our system, we will have to update the PATH variable to include the SDK.

To do that, we have to edit the file “.bash_profile” and add at the end the following line:

    export PATH=$PATH:/Development/android-sdk-mac_x86/tools

It will be necessary to launch the Android SDK Manager to install the desired APIs and documentation.
1.3.3 Windows
It is possible to download it from the official website. After extracting it, we have to run the “Setup.exe” file, which will install the SDK in our system.

In case the installer does not update the PATH variable, we will have to do it. We will have to add at the end of the current PATH the location of the SDK, which might be similar to:

C:\android\android-sdk_r04-windows\android-sdk-windows\tools

It will be necessary to launch the Android SDK Manager to install the desired APIs and documentation.

1.4 Installation of Xcode

Xcode can only be installed on a device running Mac OS X operating system. The installation process is very simple. Xcode can be found on the App Store and installed free of charge. However, it is necessary an Apple ID to download and install it.

1.5 Installation of Titanium SDK

This step is common for all operating systems. The only things that will differ will be the locations of certain packages, such as the Android SDK, which may be different depending on which operating system is being used.

First of all, we need to install Titanium CLI via npm using the following command:

`sudo npm install -g titanium`

Once we have installed the CLI, we need to log in to be able to download and install Titanium SDK. The following command will ask for the credentials of our Appcelerator account:

`titanium login`

Then we can download and install the SDK by running:

`titanium sdk install`
1.6 Configuration of Titanium CLI

The last step in the installation would be the configuration of Titanium environment. We can configure the main parameters running:

titanium setup quick

This will run a program that will ask for the following parameters:

- Name that will appear as author of the applications.
- Email address used to login.
- Titanium SDK version used.
- Location of the workspace.
- Location of the Android SDK.

After this configuration process, the environment is ready to run Titanium SDK, and therefore, the program developed during this project.