Alternatives to Native Mobile Development

Tomas Thelander
Alternatives to Native Mobile Development

Abstract
This study looks at alternatives to creating applications for the common mobile operating systems using their respective native languages. Many proposals on how to bridge the differences of the platforms exist, with Appcelerator Titanium being one of them, offering native applications for several platforms with one common code.

Titanium development is here compared to standard Android development by creating identical applications using the two technologies and comparing the development processes.

Titanium shows great promise and is able to perform the same operations as Java, with significantly less code required. The Titanium application also works on other platforms, but with some additional work required.

The application created with Titanium does not match standard Android development when developing for a single platform. However, when developing for multiple platforms it may be a suitable alternative, at least when developing applications without advanced functionality.
Sammanfattning

Denna studie tittar på alternativa sätt att skapa applikationer för de vanligaste operativsystemen utöver de standardspråk som officiellt stöds av tillverkarna av de olika plattformarna. Det finns många förslag på hur man ska kunna överbrygga de olikheter som finns gällande utveckling för de olika populära operativsystemen, och Appcelerator Titanium är en av dem, med vilken man kan utveckla maskinvaruspecifika applikationer till olika plattformar med samma kod.

I denna studie jämförs utveckling med Titanium med med vanliga Java-utvecklingen för Android genom att skapa identiska applikationer med de två metoderna.

Titanium verkar mycket lovande och är kapabel till att genomföra samma saker som Java, med betydligt mindre kod. Applikationen som skapades med Titanium fungerar dessutom på andra plattformar, även om det krävs lite extra arbete för att det ska fungera.

Applikationen som skapades med Titanium når inte upp till samma nivå som den som skapades med Androids standard Java, men när man utvecklar samma applikation för flera plattformar så kan det vara ett passande alternativ för att drastiskt kunna dra ner på utvecklingstiderna, åtminstone om man utvecklar en applikation utan några mer avancerade funktioner.
Alternatives to Native Mobile Development

Table of Contents

1. Introduction........................................................................................................................................5
2. Background........................................................................................................................................5
   2.1 Native Applications.......................................................................................................................5
   2.2 Web Applications..........................................................................................................................6
   2.3 Hybrid Applications.......................................................................................................................6
   2.4 Appcelerator Titanium..................................................................................................................7
3. Methods..............................................................................................................................................7
4. Results................................................................................................................................................8
   4.1 The Application..............................................................................................................................8
   4.2 Development Environment............................................................................................................9
      4.2.1 Standard Android....................................................................................................................9
      4.2.2 Titanium................................................................................................................................10
   4.3 Internet Connectivity....................................................................................................................10
      4.3.1 Standard Android....................................................................................................................10
      4.3.2 Titanium................................................................................................................................11
   4.4 Location Awareness.......................................................................................................................13
      4.4.1 Standard Android....................................................................................................................13
      4.4.2 Titanium................................................................................................................................14
   4.5 Data Storage..................................................................................................................................16
      4.5.1 Standard Android....................................................................................................................16
      4.5.2 Titanium................................................................................................................................17
   4.6 User Interface................................................................................................................................18
      4.6.1 Standard Android....................................................................................................................18
      4.6.2 Titanium................................................................................................................................19
   4.7 Cross-Platform Compatibility........................................................................................................19
5. Discussion and Conclusion..................................................................................................................20
References............................................................................................................................................22
Alternatives to Native Mobile Development

1. Introduction

Mobile handheld devices have gained much popularity during the last couple of years, and the ability to create applications for these devices have become an enormous industry and many companies around the world feels the need of being part of that industry. But these mobile devices are based on different platforms which makes it difficult and expensive if the company wants to reach as many people as possible.

The purpose of this study is to look at a common alternative to platform-specific development called Appcelerator Titanium and compare it to native development for the Android platform. By creating two identical applications that uses several features that are closely associated with modern smartphones using the two different frameworks it will be determined if the alternative method is able to produce applications as capable as when using the standard method of development.

2. Background

When Apple launched the iPhone in 2007, it marked a shift in how mobile phones are used. Mobile phones had been a part of the everyday life for many years already, but they had mainly focused on making phone calls or sending short text messages. There had been a few smartphones before the iPhone, but they focused almost exclusively on email and had touch screens which could only be used with a pen-like accessory called a stylus [4]. What the iPhone did differently was being accessible to everyone, and offers something interesting for everyone, at first with a competent web browser capable of delivering the majority of the Internet to the palm of your hand, and later with an extensive application market.

2.1 Native Applications

From a developers standpoint, the iPhone changed a lot when it comes to mobile development. In the early 21st century, it became possible to install small programs on your mobile phone, thanks to the Java Micro Edition platform [5]. At the time before the smartphones gained widespread popularity, Java was the most popular method if you wanted applications for mobile phones. While Java ME was not truly platform independent (some device-specific customization was necessary) it made it quite an easy task to deliver an application to several different handsets [5].

The iPhone was initially not going to support third-party native applications, instead it was then-CEO Steve Jobs' wish that applications should be built using platform independent web technologies [2, 4]. However, Apple later released a Software Development Kit (SDK) in early 2008. This began a trend within the industry and now almost all major smartphone operating systems offer an SDK for third-party native developing [4, 5, 6].

While native programming enables you to create great applications that can use all the functionality
that the device offers, it has several drawbacks, with the biggest being the very nature of the mobile industry [9]. The different mobile platforms all offers their own SDK, with development being done with different languages and using different APIs (Application Programming Interface). This means that a developer that want his or hers application to reach as many customers as possible needs to translate the entire code to work with the new platform, which is a tedious and expensive process, not to mention that each new application also needs to be maintained [4, 5, 6, 9]. This process may be impossible for independent developers or small companies. In fact, Vic Gundotra, VP of engineering at Google stated that “even Google was not rich enough to support all of the different mobile platforms from Apple's App Store to those of the BlackBerry, Windows Mobile, Android and the many variations of the Nokia platform” [4].

2.2 Web Applications

There is only one thing that all these different mobile platforms have in common, and that is a web browser [4]. Web applications are applications created with modern web technologies (HTML, CSS, JavaScript) and are intended to be displayed inside a web browser. Because of this, they are also platform independent, as long as the browsers adhere to modern web standards.

Web applications offer several advantages over native programming, with cost effectiveness being one of the most often mentioned. Because web applications needs to be created only once while native applications needs a separate version for each platform, it drastically cuts down on development time. Because it is only needed to hire one or two web developers instead of multiple developers that knows the specific native languages, the company also saves money on personnel costs [8, 10].

However, web applications suffers from the lack of ability to use device functionality (GPS, accelerometer, camera etc.), limited performance and no universal way of monetize the applications [8, 9, 10]. Web applications will continue to get better with technologies such as HTML5 that allows for offline use and better integration with device functionality, and ever faster JavaScript interpreters [4, 10]. There may not even be an either-or situation, it is probable that the two will be able to coexist [7].

2.3 Hybrid Applications

Hybrid applications are a way of uniting native and web, and take advantage of the advantages of both techniques. In hybrid applications most or the entire user interface is made using web technologies, displayed in a browser (or, more accurately, a WebView) and wrapped inside a native application. It then appears to be just as any other application on the device and may even be sold on device specific marketplaces such as Apple AppStore or Google Play [9].

The fact that the JavaScript inside the web portion of the hybrid application and the outer native wrapper may communicate with each other is used by platforms such as PhoneGap or Sencha Touch
Alternatives to Native Mobile Development

to allow the JavaScript to use the device hardware thus being able to take advantage of everything a
native application may.

2.4 Appcelerator Titanium

The Titanium development platform by Appcelerator offers ways of creating applications for
several operating systems using only one code base. It may be used to create hybrid applications or
it may be used to compile JavaScript source code into native code, thus creating truly native
applications for several types of devices using one codebase [1].

3. Methods

The target platform for this study was chosen to be the Android operating system. The main reasons
for this was that an Android device was easily accessible at the time of the study, and that Android
is now one of the most popular platforms for mobile devices.

In order to compare Titanium with native Android development, two applications was created, one
for each method. They was created to be as identical as possible and to take advantage of several of
the device's capabilities. Because native development with Java is by far the most used method of
developing for Android, and the only one that is officially supported, this study took the native
application as a point of reference, and compared the Titanium application to it.

Four areas was chosen as cornerstones for the comparison, with the first three being Internet
connectivity, location awareness and data storage, and the fourth being the user interface for
presenting everything. These four was chosen because they represent the crucial parts of what a
smartphone is expected to be capable of, and by looking into these things the insight that’s needed
for this study will be gathered.

Things that was taken into account when comparing was the amount of code needed to perform a
specific task, complexity and any potential differences in approach that the frameworks takes when
implementing a feature. Environmental differences, like capabilities of the IDE, was also looked
into, and any work that was required besides the actual coding was included.

When testing the different areas, only basic operations were implemented. When testing Internet
connectivity the application was able to fetch and present a simple set of external data. For location
awareness the current location was retrieved and displayed on a map and for the data storage test
the application saved small amounts of data on the SD-card and using an SQL database. When
testing the user interface capabilities a simple user interface was created utilizing basic controls and
means of presentation such as buttons and text labels.

The native development was performed by following the official guidelines set up by Google and
by using the officially supported Android SDK and the Android Development Tools plug-in for the
Eclipse IDE (Integrated Development Environment), while the Titanium development was using the
Alternatives to Native Mobile Development

Titanium SDK and the Titanium Studio IDE. The development and testing was done using an Acer Liquid smartphone running Android 2.3.7.

4. Results

The results section will start by presenting the application that was created, before a look at the development environment and the tools used when developing with the different techniques. We will then look at the three areas of comparison individually (Internet connectivity, location awareness and data storage), looking first at how it is done with standard Android Java and then with Titanium JavaScript while comparing the two. The fourth area, the user interface, will both be present when presenting the three previous areas as well as having a section of its own near the end.

Table 1: How the study will be performed

<table>
<thead>
<tr>
<th>Result Arrangement</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Environment</td>
<td>The environments of the two techniques.</td>
</tr>
<tr>
<td>Internet Connectivity</td>
<td>Retrieving data from a public API.</td>
</tr>
<tr>
<td></td>
<td>Presenting the data using ListView and WebView.</td>
</tr>
<tr>
<td>Location Awareness</td>
<td>Retrieving the current location.</td>
</tr>
<tr>
<td></td>
<td>Displaying the location on a map.</td>
</tr>
<tr>
<td>Data Storage</td>
<td>Accessing files on the external storage.</td>
</tr>
<tr>
<td></td>
<td>Retrieving and inserting data into a SQLite database.</td>
</tr>
<tr>
<td>User Interface</td>
<td>How the user interface is created and managed.</td>
</tr>
<tr>
<td>Observations</td>
<td>Presenting the application</td>
</tr>
<tr>
<td>Cross-Platform Compatibility</td>
<td>Titanium’s ability to work on different platforms is tested.</td>
</tr>
</tbody>
</table>

4.1 The Application

The application created using both standard Android and Titanium is a relatively simple one, designed not to perform a specific task but just to test the capabilities of the frameworks by making use of several features that are closely associated with smartphones. The application is able to access an API over the Internet and retrieve and parse JSON-formatted data, figure out the location of the handset using the built-in GPS chip and storing and retrieving data locally on the device, both as text files on the file system and using the support in Android for SQLite relational databases.
Alternatives to Native Mobile Development

All of this functionality are bound together using a simple user interface made up mostly of simple elements such as buttons and labels and different views for presenting data, such as a list for displaying the data received over the Internet and a map for displaying the current location. Even though it is simple, it will give a good look into how the development process works for the different frameworks, and how capable they are in utilizing the capabilities of the device.

The two applications were designed to look and function as similar as possible. And they are almost identical, except for a few minor graphical differences, but the functionality is exactly the same. The only major difference is that Titanium includes a mandatory splash screen that is changeable but not removable, and this is not present in the standard Android application. They are comparable in performance, none of them appear to lag in any way during normal usage, this, however, might not have been the case if the applications performed more CPU intense operations instead of relatively simple ones used in this study.

When the applications are inspected in the device's application manager there is one thing that clearly sets the two apart. The application developed using standard Android Java is, when all data is cleared, 52 kilobytes big. This is normal for an application with only basic operations and with no high definition graphics. The Titanium application is bigger, more than 200 times bigger, as it weighs in on around 12 megabytes. The big difference in sizes appears to be due to the V8 JavaScript engine libraries that are included in the Titanium application installation file.

The rest of the result section will compare the different parts of the two development processes in detail.

4.2 Development Environment

The development environment is the collection of software and resources available to help the developer work with the platform. It can include a utilities collected in a software development kit, an integrated development environment with a code editor, debuggers and other features and online documentation of the platform and its features.

4.2.1 Standard Android

When a developer wants to start to work with the Android platform, a natural starting point is the Android Developers site at http://developer.android.com, which contains guides on how to use the software development kit (SDK), tutorials, API references and videos about the platform that together gives the developer all the information needed to start creating native applications for Android.

Google recommends using the Eclipse IDE when developing for Android and together with the Android Development Tools (ADT) plug-in for Eclipse that integrates it with the Android SDK and Eclipse's extensive plug-in repository it results in an environment that has just about anything needed. Eclipse offers all the functionality that is to be expected from a modern IDE and the
Alternatives to Native Mobile Development

integrated Android SDK customizes it to allow for testing on devices and emulators, auto-completion of code according to the Android API, preview of the user interface, etc. There are, however, alternatives to Eclipse, although not officially supported by Google. For example, the community created nbandroid plug-in brings Android support to the popular NetBeans IDE.

4.2.2 Titanium

The Appcelerator website offers much the same as the Android Developers site. The API documentation, the tutorials and the videos are all present, as is to be expected.

While standard Android development relies on the third-party IDE Eclipse, Appcelerator has their own IDE to use for Titanium development; Titanium Studio. Titanium Studio is based on the Aptana Studio IDE, which in turn is based on Eclipse, so anyone used to Eclipse will soon feel at home with the Titanium IDE.

One difference worth noting is that Appcelerator requires the developer to register on their website before downloading the SDK and IDE, and Titanium Studio prompts for login information upon opening the first time. It is free for independent developers and small companies without the need for extensive support. Titanium Studio also allows for deployment of the application on either a device connected via USB or an emulator, much in the same way as Eclipse does for native applications.

The fact that Titanium Studio is based on Eclipse and that it makes great use of the Android SDK means that it can offer functionality that is in most ways comparable to that of native development. One thing that is missing is the ability to preview the user interface without building the application.

4.3 Internet Connectivity

An Internet connection is a central part of a modern handheld device, and it is important that it is easy for the developer to access information on the Internet, because in our modern society, the information is not located on individual devices, it is located in the so-called “cloud” of Internet services.

The comparison of the techniques in respect to Internet connectivity was performed by retrieving information from a public REST (Representational state transfer) API over HTTP (Hypertext Transfer Protocol). The information is in JSON-format (JavaScript Object Notation) and is parsed and presented.

4.3.1 Standard Android

Native Android programming is done using the Java programming language, and although it is not entirely the standard Java developed by Sun Microsystems and now owned by Oracle, when it comes to basic operations such as Internet access, they work almost exactly the same way.
Java is a highly object-oriented language so for every kind of operation there is several types of objects that needs to be used. For example, this is a typical approach in order to retrieve information from the Internet; first, a URL object must be created that point to the resource in question, a URLConnection object is then used to open a connection to the resource pointed out by the URL object. Then, an InputStreamReader object reads the stream of data returned from the URLConnection, after which a BufferedReader converts the data into text (see illustration 1). Despite this extensive use of different objects, it is a pretty straightforward process and the fact that the different objects throws different exceptions makes it easy to troubleshoot.

One thing that differs from standard Java is that the use of Internet must be declared in the manifest (an XML file containing information about the application), this is because the user of the application must approve the Internet usage upon installation.

The information retrieved in this particular case was information about blog posts, formatted in the human-readable form of JSON. JSON is often seen as an alternative to XML when transmitting data between applications. Even though JSON is based on JavaScript, it is now supported in a wide variety of programming languages, and Java is no different, the org.json-package is accessible from within Eclipse without the need for any further configuration, and the JSONArray and JSONObject classes easily converts JSON text into Java objects.

As for presenting the data retrieved, a ListView seemed appropriate, since the data was a list of blog posts (see illustration 2). A ListView is a user interface element that takes the form of a vertical list of items, displaying at minimum a line of text for each item. The creation of a ListView that displays JSON data turned out to be more difficult than anticipated, as it was needed to create a new class that extends the ArrayAdapter class in order to define how the data should be interpreted and how the ListView should be displayed. When an item in the list is clicked (or touched) by the user, the entire blog post is displayed using a WebView, which is a view that contains HTML-formatted text. Creating a WebView turned out to be incredibly easy, with creating an object and telling it what HTML to display being the only steps needed.

More about the user interface in standard Android and Titanium in the User Interface section later in this paper.

4.3.2 Titanium

Since the language used with Titanium is JavaScript instead of Java, the coding style is radically different to that of standard Android. While the Android Java wholeheartedly accepts the highly object-oriented paradigm of standard Java, Titanium does the same with JavaScript's dynamic and weakly-typed prototype-based paradigm and makes heavy use of functionality such as anonymous functions and closure.

When retrieving information from the Internet, it is easy to see that Titanium has been inspired by the XMLHttpRequest object of Ajax, and the result is an elegant and simple to use solution that
Alternatives to Native Mobile Development

requires few lines of code in order to function correctly (see illustration 1). It is not needed to express the need to use Internet in any way, Titanium takes care of this when it creates the manifest during compilation. JSON, being based on a subset of JavaScript, is deeply integrated into Titanium, and parsing the retrieved data is a matter of a single line of code.

The creation of a user interface is also very different from standard Android development, and since JSON is a big part of the JavaScript language, there is no need to tell the ListView (called TableView in Titanium) how to parse it, it is done automatically. And WebViews are easily created, with the difference from standard Android being that it doesn't accept the HTML to be displayed as a string, it must be converted to binary data first which is odd, but easily done using the createBuffer method.

```javascript
var xhr = Ti.Network.createHTTPClient(
    onload : function(e) {
        // Success
        Ti.App.Posts = JSON.parse(this.responseText);
    },
    onerror : function(e) {
        // Error
    },
    timeout : 5000
);
xhr.open('GET', 'http://www.thethee/api/posts');
xhr.send();

try {
    URL api = new URL("http://www.thethee/api/posts/");
    URLConnection tc = api.openConnection();
    BufferedReader in = new BufferedReader(new InputStreamReader(
        tc.getInputStream()));

    String line;
    while ((line = in.readLine()) != null) {
        JSONArray ja = new JSONArray(line);

        for (int i = 0; i < ja.length(); i++) {
            JSONObject jo = (JSONObject) ja.get(i);
        }
    }
} catch (MalformedURLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (JSONException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Illustration 1: Retrieving Internet data with Titanium (top) and Java (bottom)
4.4 Location Awareness

Another important feature in modern mobile devices is the ability to determine its location. This is usually performed via a GPS chip (high precision) or via mobile networks and/or WiFi networks (low precision).

This section was tested simply by determining the location on the user and displaying the location on a map within the application.

4.4.1 Standard Android

Since we are using a functionality that may be abused in order for malicious applications to gain sensitive information about the user, the usage of the current location must be declared in the manifest of the application. There are two different permissions that may be applied for independently or together: coarse location (networks) and fine location (GPS).

The retrieval of the location in standard Android is a three step process that starts with defining a LocationListener object that triggers when a location is found (see illustration 3). The next step is to request the location service from the device, which is returned in the form of a LocationManager object. And lastly, pass the listener to the manager and declare which method to use (GPS or network).

The location that is retrieved consists of a set coordinates, (latitude, longitude and altitude), which is
Alternatives to Native Mobile Development

not very meaningful until you point it out on a map. Luckily, Android, being created by Google, provides an API to integrate Google Maps into your application (see illustration 4). This library is not a part of the standard Android API, so it first needs to be installed using the Android SDK Manager. Then it must be declared in the manifest that the application uses that particular library. The last thing that needs to be done before the actual implementation can begin is to get an API-key for the Maps service. These are provided for free on Google's developer site.

When the preparations are done, the actual implementation of a MapView is similar to the creation of any other kind of view, a MapView is defined in an XML-file describing the layout, and an activity is set to use that particular XML-file as its content. This only creates a map, to display the current location it is needed to create a class that manages “overlays”, which is graphics intended to be displayed on the map. When the LocationListener then triggers, an overlay is created with an image that will be displayed on the map as parameter and the location in latitude and longitude added to it. For the map to display the newly added image it needs to be invalidated, which means it is forced to be redrawn. It is also possible to control the map from the program code, using the MapController object, which offers methods for moving the map to focus on a certain point, and to set the level of zooming among other things.

4.4.2 Titanium

Titanium continues to use the easiest possible ways of accessing services, once again there is no need to ask permission for accessing the location, Titanium does that automatically, and it raises the level of abstraction several steps compared to Java. The current location is retrieved by calling only one method called getCurrentPosition, taking as an argument an anonymous callback function (see illustration 3).

When creating a MapView in Titanium it requires that the Maps library is installed, just as standard Android, but it does not require a specific definition in the manifest about which libraries are used. An API-key for the Maps library is also provided by default, but only for development purposes, for a real key the same steps as for standard Android development are needed.

The creation of a map is also performed by calling a single method with several arguments available to customize it. If an argument called userLocation is set to true it will automatically pin the location of the user on the map. This, however, only works if the application has previously retrieved the location. It is possible to create more advanced annotations on the map, but this simple operation provides a very similar result to that of standard Android described above.
Alternatives to Native Mobile Development

```javascript
Titanium.Geolocation.getCurrentPosition(function(e) {
    if (e.error) {
        alert(e.error);
        return;
    }
    var longitude = e.coords.longitude;
    var latitude = e.coords.latitude;
});
```

```java
final LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        int lat = (int) (location.getLatitude() * 1E6);
        int lng = (int) (location.getLongitude() * 1E6);
        GeoPoint point = new GeoPoint(lat, lng);
        locationManager.removeUpdates(this);
    }
    public void onStatusChanged(String provider, int status, Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {};
    locationManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER,
        0, 0, locationListener);

Illustration 3: Retrieving current location with Titanium (top) and Java (bottom)
Alternatives to Native Mobile Development

4.5 Data Storage

Many applications require some way of saving data for use later on, data that should be permanent and not deleted until the application does not need it anymore.

Android offers several ways of storing data, either on a limited amount of internal memory for small amounts of data, or on the external memory which can take the form of a removable Secure Digital (SD) memory card or as built in flash memory. The external memory is several gigabytes large and can thus hold sizable quantities of data, however, it may not always be accessible, for example if it is mounted as a storage device by a PC. Android also offers the possibility of creating SQLite relational databases.

The data storage capabilities of standard Android development and Titanium will be examined by both accessing and reading a file on the external storage and creating and managing an SQLite database.

4.5.1 Standard Android

As with most other features of the device, it must be declared in the manifest that the application intends to use the external storage, however, in this case it is only necessary if it intends to write files to the external storage, reading files does not require permission.

Before the external storage is accessed it must be determined if it is mounted or not by calling the
Alternatives to Native Mobile Development

Environment.getExternalStorageState method. Reading and writing with Android Java is very similar to reading and writing files when programming with Java on a PC. First, the external storage must be retrieved by calling the getExternalStorageDirectory method which returns a File object representing the root directory of the external storage. Using this File object and the FileReader and BufferedReader classes, any file can be accessed.

In order to use a SQLite database in Android, it is recommended that a class is created to represent that database. This class should extend SQLiteOpenHelper and implement the methods onCreate and onUpgrade. The onCreate method is called automatically when the tables in the database is accessed but not yet present and it should create all tables intended to populate the database, which is done using the common SQL language. In this case a single table is created intended to hold a date and time for each time a button is clicked. Additional methods should be added to the class depending on the purpose of the database. In this case, addVisit, which adds the current time and date to the table, and getLatestVisits, which returns the five latest table posts, was added. When this class is implemented all that is needed is to create an object of it and call the method that performs the wanted operation.

4.5.2 Titanium

Checking if external storage is present is done approximately the same way in Titanium as in Java, but actually accessing files is even simpler than in standard Android. Reading text from a file can be done in only one or two lines of code.

There is one major difference between standard Android and Titanium in this regard; Android Java allows access to the entire external storage file structure, while Titanium only allows the application to read or write from an application-specific subdirectory, which is a major drawback.

Accessing a database is also a very simple task in Titanium, a database object is retrieved from the Titanium framework and this object is then used to directly execute SQL statements.

In the application, the data retrieved from the file and the database is then printed on the screen (see illustration 5).
Alternatives to Native Mobile Development

4.6 User Interface

How the user interface for an application is created differs greatly for the different development technologies, and here follows a comparison on the general principles behind user interface management on standard Android and Titanium.

4.6.1 Standard Android

When programming in Java for Android, the user interface layout is defined in XML files, these files determine what elements should appear on the screen, where on the screen they should be, how they should look and many other things.

The programmer then creates an activity, which takes the form of a Java class. An activity is a screen and it contains different elements such as buttons or input fields. An action by the user, for example a click on a button, may take the user to another activity that displays a different set of controls.

When a new activity is created its content is set to one of the XML layout files, which then controls what is displayed on the screen. Each element in the XML layout should have an identification.
string. This identification makes it possible for the programmer to access the element from anywhere inside the activity and change some of its properties.

4.6.2 Titanium

Titanium have a completely different approach, no XML files are used and instead everything is done from the program code. First a window is created, which is roughly analogous to an activity. A view is then added to the window. A view can be a wrapper for different elements like buttons, but can also be more complex like MapViews, similar to the concept of views in standard Android. Objects of the different types of elements can be created and then added to the view and they will be displayed.

One major difference that immediately makes itself noticed is that the elements does not have any identifications that makes them accessible once they are created. Instead, to be able to access an element after it has been created and added to the screen the developer would have to rely the feature of JavaScript called closure, which means that a function created inside another function inherits the local variables of that parent function, so in effect, the newly created function can access variable outside its local scope. This feature makes it possible to create a new element and assign it to a variable, and then create a new function that is to be called on a specific event, and that function now has access to the element and may change its properties.

Android-specific features such as menus when the menu button is clicked and Toast-notifications are all available in Titanium, with little to no effort required to implement them.

4.7 Cross-Platform Compatibility

Titanium can not only develop native applications for Android, the main selling point for the platform is that an application developed with Titanium will work on Android, iOS and as a mobile website, all with the same codebase. Since developing for iOS requires a computer running Mac OS X and none was available, the application was compiled as mobile website, published on a private web server and tested on both a PC and the browser of an Android smartphone.

Because it is made for the purpose of testing Android functionality not everything works, but the section with Internet connectivity was tested. It works reasonably good when viewing it in a browser running on a PC, apart from the fact that there is no way of going back to a page previously displayed without reloading the page. However, when tested in a browser running on an Android smartphone, the website was too slow to be usable, too much JavaScript was executed and the site appeared very unresponsive. The device which it was tested on is relatively old so it is probable that it works better on a newer smartphone. It is clear that there is a lot of extra programming needed for a Titanium application to work on several platforms.
Alternatives to Native Mobile Development

5. Discussion and Conclusion

In this study, two identical applications was developed specifically for making this comparison which made it possible to get a good insight into the development process and to choose which areas to focus on, something that greatly helped the process compared to performing the study with applications already developed and/or by a different developer.

Titanium offers a way of developing for mobile platforms without the need to learn different languages for each platform, instead relying on JavaScript and an extensive API. This study has specifically looked on the differences between Titanium and standard Android development using Java.

When the capabilities of Titanium are tested by implementing common features the differences between JavaScript and Java are immediately shown. JavaScript is a simpler and more dynamic language with less complex structures, while Java is heavily object-oriented with many different classes that can be difficult to remember how to use, but with its well defined structure it offers a stability and understandability that JavaScript and Titanium cannot match.

These differences are directly seen when comparing tasks between the techniques, to do the same thing in Titanium as in Java almost always requires fewer lines of code and less complex structures, a few calls to built-in methods with object primitives and anonymous functions as arguments performs the same operations that would need new classes to be created and existing classes to be used in order to do the same thing in Java.

This means that for a developer with comparable experience in the two techniques probably could program faster with Titanium than Java while still produce similar results. However, the things mentioned here as advantages may in effect become drawbacks, the structure present in standard Android Java that is missing from Titanium may add complexity but it also gives more control over the application, especially when more advanced applications are to be built, it is for example probably impossible to create any fairly advanced games using Titanium. That together with Titanium's flaws, most notably the large size of the applications, makes it an inferior choice compared to Java when developing for a single platform.

The situation changes when the application needs to work on several mobile platforms. One of Titanium's main objectives is to offer cross-platform compatibility with only one codebase. There are plenty of things needed to be taken in consideration when using Titanium to reach several platforms, like that some user interface elements only exist for one of the platforms and must be implemented differently on the other, but it is a lot faster when compared to the alternative of developing native applications for each platform.

To write an application in Java for Android and then go on and write the same application again in Objective-C for iOS is an expensive task in both hours, and hence the salary of the developers, and in that it needs specialized knowledge about the platform, which might bring the need to hire
Alternatives to Native Mobile Development

additional people to the company or send existing employees to get expensive education. In this case Titanium might be a good choice, since it has been demonstrated that it can be used to create native applications that access the device's functionality.

It has been noted that while the “write once and run everywhere” approach is a noble one but not very easy to realize and tends to fall short of their ambitions [5, 9]. This appears to be true but while Titanium is not the answer to all problems with mobile development, it performs better than expected. If Titanium and other technologies like it continues to evolve they might have a future as an alternative when dealing with simple applications on multiple platforms, but they will likely never become a true alternative when creating advanced applications with a need for customization and low-level operations.
Alternatives to Native Mobile Development

References


