Evaluation of Cross-Platform Mobile Frameworks

April 2013

Brandon M. Zimmerman, Arizona State University

Abstract—To aid in making a decision on selecting a cross-platform mobile framework, the differences between several prominent frameworks are discussed in this project. The following criteria were used in the comparison: cost, underlying mechanism, performance, user interface, device-specific functionality, web services, code maintenance, and convenient aspects. To demonstrate the capabilities of the Monocross Framework, a cross-platform mobile application is presented that makes use of the Bing Maps API, location services, and platform-agnostic storage.

Index Terms—Android, iOS, BlackBerry, Windows Phone, Symbian, webOS, Cross-Platform, C#, iOS, Mobile, Mobile Framework, Mono, .NET.

I. INTRODUCTION

The mobile user-base of 2013 is a non-homogenous mixture of several platforms. Due to this fact, it is exceedingly important to target multiple platforms to have the same market share that competitors already possess. In other words, developing for a single platform puts the author at a great loss in terms of market share and ultimately the profit that would be derived from the application. As of Q4 2012, the two platforms with the largest market share are Android at 69.7% and iOS at 20.9% [Gartner]

Historically, the approach that was relied on to create applications for multiple platforms was simple: developers would write native applications targeted at one platform at a time. This approach is very cumbersome and involves implementing the same requirements multiple times using different code. Further, this approach is no longer a viable option due to obvious reasons such as money and time constraints.

Today, the approaches that are being used in the industry utilize cross-platform mobile frameworks that are designed to allow developers to create a single codebase and re-use it across multiple mobile platforms. The purpose of this paper is to identify the differences of the various cross-platform mobile frameworks in an effort to allow companies to make an educated decision when selecting a cross-platform framework.

To exemplify the use of a cross-platform framework the Monocross framework has been selected to create a mobile application for the two mobile platforms with the highest marketshare -- iOS and Android.

II. RELATED WORK

There are dozens of cross-platform mobile frameworks in existence. The scope of frameworks compared in this report is limited to those that provide all of the following functionality: iOS support, Android support, storage manipulation, geolocation, and WiFi communication. Five mobile frameworks besides MonoCross currently provide all of this functionality [Markus Falk].

A. Titanium

Appcelerator Titanium is a cross-platform framework that employs the JavaScript language [Appcelerator]. It supports three mobile platforms: iOS, Android, and BlackBerry. Development can be done using Windows, Mac, or Linux, but due to Apple licensing it is required that an Apple device be used to create iOS applications. The Integrated Development Environment (IDE) that is provided is called Titanium Studio and is based upon the popular open-source IDE Eclipse.

1) Cost

Appcelerator provides their Titanium product free of charge for the vast majority of their users. However, Appcelerator does charge for support and also plugins obtained from their app store.

2) Underlying Mechanism

To execute JavaScript code on any device an interpreter is needed. The interpreter used on the iOS platform is JavaScriptCore and the interpreter used on Android and BlackBerry is Mozilla Rhino [Titanium]. These interpreters are bundled along with the source code during the build process for their respective platforms.

3) Performance

Titanium does not offer end-users native performance because the source code is interpreted at runtime rather than compiled before execution [Titanium].

4) User Interface

Titanium allows users to create native user interfaces for all three mobile platforms by allowing users to simply put platform-specific JavaScript files into appropriately named
source code directories such as “android” and “iPhone” [Appcelerator Quickstart]. Titanium has support for both Sencha Touch and jQuery mobile frameworks; developers may use these JavaScript frameworks for special effects including fade-in, fade-out, hide element, and show element.

5) Device-Specific Functionality

Common wireless standards such as Bluetooth and Near Field Communication (NFC) are not available out of the box. To use Bluetooth functionality a plugin must be obtained from Appcelerator Marketplace, which is Appcelerator’s app store. At the time of writing, Bluetooth plugins for iOS and Android range between $250 and $350 per platform. Similarly, the NFC plugin can be obtained through Appcelerator Marketplace for $10 per month.

6) Web Services

Formats such as SOAP, XML, and JSON are supported for communicating with web services. JSON is the preferred format due to the simplicity of serializing and de-serializing it using JavaScript [Dealing with SOAP]. Data can be serialized and de-serialized using the built-in stringify and parse methods of the JSON class.

7) Code Maintenance

Projects utilize a single codebase written in three well-known languages. All code is identical between each target mobile platform except for platform-specific views. Another framework that Appcelerator offers known as Alloy helps promote both code re-usability and maintenance by separating the presentation layer, view, and business logic using the Model-View-Controller (MVC) paradigm.

8) Convenient aspects

Being that Titanium only requires knowledge of JavaScript, the barrier to entry is very low. This means that Titanium should be easily accessible to people other than developers such as web designers.

Appcelerator provides what the company refers to as a Mobile Backend as a Service (MBaaS) through their Cloud Services offering which is included in their free tier. Cloud Services is essentially a collection of REST endpoints that allow end-users to perform activities including check in, authentication through facebook, and file/photo upload to a 20GB cloud storage area that is also provided for free.

B. PhoneGap

The PhoneGap mobile framework is developed by Adobe Systems [PhoneGap]. It allows the creation of mobile applications using technologies that web designers are familiar with such as HTML5, CSS3, and JavaScript. Currently, PhoneGap supports seven mobile platforms including: iOS, Android, BlackBerry OS, WebOS, Windows Phone, Symbian, and Bada [Apache Cordova]. Operating systems that can be used for development include, but are not limited to: Windows, Mac, and Linux. The restriction that iOS apps be developed using Apple hardware can be alleviated by the use of a cloud compiler. Any IDE can be used for development. However, the author recommends using a free and open-source IDE such as Aptana due to its code-highlighting abilities for HTML, CSS, and JavaScript.

1) Cost

PhoneGap is completely free and open-source. All applications including ones intended for commercial use can be developed using PhoneGap. The framework is community-supported through means such as Google Groups and Internet Relay Chat (IRC). However, support can also be obtained for $249.99 to $19,999 per year depending on amount of developers and desired support level [PhoneGap Support].

2) Underlying Mechanism

PhoneGap uses Apache Cordova as its back-end to allow access to device-specific functionality. To be able to do this Apache Cordova provides a uniform set of JavaScript libraries that each have a different implementation for each platform [Apache Cordova]. For instance, Apache Cordova provides a JavaScript library for the iOS platform for accessing the accelerometer, which uses code native to that particular platform, and it also provides the same JavaScript library for all of the other platforms it supports as well, but with different implementations.

Implementations of Cordova vary for each platform and can be examined directly by viewing the Cordova source code available at: https://git-wip-us.apache.org/repos/asf. However, for the iOS platform it is known that a UIWebView is used to render the application’s HTML and CSS [Cordova Plugin]. The JavaScript libraries mentioned above, which provide device-specific functionality are simply called from JavaScript scripts that are located in the application’s HTML.

3) Performance

Due to the fact that PhoneGap applications are Hybrid, performance is limited by the speed of the web view of the platform the application is being executed on. Depending on the complexity of the HTML being rendered, the application may be perceived by the user to be significantly slower than an application written natively for the platform.

4) User Interface

Applications made with PhoneGap do not necessarily look like they are designed for any certain platform. PhoneGap also has support for both Sencha Touch and jQuery mobile frameworks.

5) Device-Specific Functionality

The PhoneGap API does not support Bluetooth, NFC [PhoneGap API]. However, a recently updated Bluetooth PhoneGap plugin for Android can be downloaded from here: https://github.com/phonegap/phonegap-plugins/tree/master/Android/Bluetooth.

6) Web Services

The PhoneGap API also does not support web services [PhoneGap API]. To contact a web service a plugin such as jQuery mobile must be used. The jQuery mobile web framework supports SOAP, XML, and JSON [find source]. Again, JSON would be the preferred data format due to the use of JavaScript.

7) Code Maintenance

PhoneGap applications use a single codebase across all platforms written in three languages: HTML, CSS, and Javascript. The code being interpreted on each platform is the same. There is not anything built-in to separate what will appear on each platform.
Sencha Touch can be used to separate the data model, presentation layer, and business logic using the built-in Model-View-Controller (MVC) paradigm [Sencha Touch MVC].

8) Convenient aspects

PhoneGap offers a cloud compiler known as PhoneGap Build. This means that an IDE is not necessarily required to create PhoneGap applications. PhoneGap Build supports GitHub repositories or .zip files as sources of projects to build. To get started a developer may simply fork the phonegap-start repository into their own local GitHub, modify the name of the application in the config.xml file, commit and sync the repository, and then specify the URL of the repository as a new source for a PhoneGap Build at [https://build.phonegap.com/apps](https://build.phonegap.com/apps) [PhoneGap Getting Started]. The application will then build for all seven platforms and provide download links to the packages for each platform.

C. Rhodes

Motorola develops the cross-platform Rhodes framework [Rhodes]. It uses the scripting language Ruby and with recent additions JavaScript. JavaScript is not yet fully capable of doing everything that Ruby is capable of, but Motorola claims that entire applications can now be built using JavaScript alone. Several mobile platforms are supported including: iOS, Android, BlackBerry, Windows Mobile, and Windows Phone. Development can be done using Windows, Mac, or Linux, but again, due to Apple licensing it is required that an Apple device be used to create iOS applications, unless the cloud compiler RhoHub is used to build the application. RhoMobile Suite is the IDE Motorola provides to use, but any editor capable of editing Ruby or JavaScript can be used along with the command line.

1) Cost

Rhodes is free for open-source applications, however a license is required for commercial applications. A single commercial application license, which supports an unlimited number of users, is $500 for the first year and then $200 to continue receiving updates for subsequent years [Rhodes Licensing]. RhoConnect Push Server (formerly known as RhoSync), which is not required, costs $5000 per 100 users.

2) Underlying Mechanism

For the Ruby Language, Rhodes makes use of the Ruby Virtual Machine to interpret Ruby and render HTML.

3) Performance

Motorola makes the claim that Rhodes applications written in Ruby are faster than Android Java applications. Further, they back this claim by saying that this is possible due to the fact that Rhodes is written in C++ [Rhodes Architecture].

4) User Interface

Native-looking controls such as tab bars, tool bars, date/time pickers are available for use by developers [Rhodes Architecture]. All controls are rendered in HTML and then displayed by the use of a web view control of the platform the application is being ran on. For this reason, the user interface is not fully native, but should be considered as being hybrid. If Rhodes applications are carefully designed, the user may not be able to perceive a difference between a native app and app created using Rhodes.

Web frameworks such as jQuery Mobile and Sencha Touch can be used with Rhodes [Rhodes Architecture]. Additionally, Rhodes provides built-in access to the JQTouch framework.

5) Device-Specific Functionality

All device-specific functionality that one would expect when evaluating a mobile framework is available with Rhodes including: accelerometer access, compass access, Bluetooth access, NFC access (Android-only), and vibration support [Rhodes API Support].

6) Web Services

HTTP and HTTPS web services can be accessed using built-in facilities [Rhodes Web Services]. Allowed formats for communication are XML and JSON. With the Python language the AsyncHttp class provides Asynchronous web service communication.

7) Code Maintenance

With Rhodes, there is a single code-base written in either Ruby or JavaScript. Platform-specific views can be created to display a different looking UI on each platform [Rhodes CSS]. MVC is provided out of the box with Rhodes so code is clearly separated for everyone that uses this framework.

8) Convenient aspects

Motorola provides tools such as RhoConnect Push, which is primarily aimed at enterprise customers. RhoConnect Push provides the ability to have an application send push notifications to users when changes are detected on the server side rather than having clients look for them [RhoConnect Push].

RhoHub allows developers to build Rhodes apps in the cloud [RhoHub]. As previously stated, this allows Windows users wanting to develop iOS applications to easily get around Apple’s requirement. Additionally, this is useful because it means that developers do not have to download and update SDKs.

D. Corona

Corona is a cross-platform framework developed by Corona Labs [Corona]. It supports the following mobile platforms: iOS, Android, NOOK, and Kindle Fire. Development for iOS can only be done on a Mac. There is no official IDE for Corona development, but many text editors that support Lua syntax highlighting exist. The IDE required for developing iOS applications is XCode [Corona FAQ]. Android applications are developed using the Eclipse IDE, which is available on Windows, Mac, and Linux.

1) Cost

Corona is free to try until a user wishes to publish an application [Corona FAQ]. At that point, the user has the option of either obtaining an Indie subscription for $199 per year, which lacks some premium features and plugin support, or they can choose the Pro subscription for $349 per year.

Corona Enterprise is another package that adds the ability to make native API calls on Android and iOS [Corona FAQ]. Corona Labs requests that users contact them to obtain pricing information for Corona Enterprise.
2) **Underlying Mechanism**

Corona applications rely on the Lua interpreter, OpenGL, and OpenAL [Corona Games]. The Lua interpreter is very small at under 200KB when fully compiled with all standard Lua libraries [Lua].

3) **Performance**

Due to its relative speed of execution among other interpreted scripting languages, the creators of Lua claim that other scripting languages try to compare themselves as being “as fast as Lua” [Lua].

4) **User Interface**

Corona provides for both Native and Hybrid approaches. The native library of the Corona API provides support for completely native controls which are not rendered through OpenGL like all other controls, but rather using the mobile OS directly [Corona API]. For instance, the NewTextField method of the native library will render a text field on both Android and iOS using the underlying native API for each respective platform.

5) **Device-Specific Functionality**

Some features that are available on iOS are not available on Android such as camera access, activity indicator, orientation changes, and limited support for OpenAL audio [Corona Device Specific].

6) **Web Services**

HTTP and HTTPS are supported for contacting web services [Corona Web Services]. Formats supported for communicating objects are JSON and SOAP. Web service communication is accomplished using the request method of the built-in network class.

7) **Code Maintenance**

All code is written in a single scripting language called Lua that is already known by some game developers.

Code is not separated using any design-pattern such as MVC. Also, the capability to create platform-specific views is not provided.

8) **Convenient aspects**

Corona provides cross-platform cloud services through a product called Corona Cloud [Corona Cloud]. Figure 1 shows that Corona Cloud handles authentication, leaderboard management, achievement management, push notifications, and more for development ease across multiple platforms. This is another example of a Mobile Backend as a Service (MBaaS). Using these pre-existing services may greatly reduce the scope of a development project when designing a mobile application such as a multiplayer game.

---

**E. Marmalade**

Marmalade is a cross-platform framework developed by Idealworks3d [Marmalade]. It supports the following mobile platforms: iOS, Android, BlackBerry PlayBook OS, and bada [Marmalade platforms]. No official IDE is provided. Development can be done on Windows and Mac, but a Mac is required to deploy to an iOS device.

1) **Cost**

Marmalade can be licensed for between $149 per year and $1499 depending on support and features required [Marmalade Buy]. The higher priced tiers include support for platforms other than iOS and Android.

2) **Underlying Mechanism**

The Segundo Embedded Execution Environment (S3E) is a set of APIs that allows for low-level access to a device to control things such as vibration [Marmalade API Documentation]. Marmalade applications are compiled into binary form and then bundled along with an S3E for the specific platform. This is referred to as a loaderbinary system where the loader is the S3E and the binary is the user’s compiled application.

3) **Performance**

Due to the fact that Marmalade applications are written in C++, code is compiled into machine language against each target platform [Marmalade API Documentation]. Applications created with Marmalade should offer performance that is nearly the same on the iOS platform as an Objective-C application. On other platforms, such as Android, Marmalade may outperform the intended language that applications are to be written in (Java for Android).

4) **User Interface**

Native and hybrid user interface approaches are available with Marmalade. The normal version of Marmalade is capable of having mobile platforms render native controls. Web Marmalade allows users to write in HTML5, which is similar to PhoneGap.

5) **Device-Specific Functionality**

Marmalade offers all of the device-specific functionality that has come to be expected of cross-platform mobile
frameworks except for Bluetooth and NFC, [Marmalade API Documentation].

6) **Web Services**

The ability to contact web services is provided by the built-in IwHTTP class [Marmalade Web Services].

7) **Code Maintenance**

This framework does not offer much in terms of code maintenance. Design patterns such as MVC are not used.

8) **Convenient aspects**

Marmalade Quick allows developers to use the Lua scripting language instead of C++ to create Marmalade applications [Marmalade Quick]. Although it is not yet available, Marmalade Juice promises to allow users to port Objective-C iOS application to Marmalade [Marmalade Juice]. This will then allow these applications to work for both iOS and Android.

### III. MONOCROSS APPLICATION

#### A. Application Description

The Monocross demonstration application provides a Point of Interest (POI) search capability based on GPS coordinates using data provided by Bing Maps. It is capable of filtering based on keyword, radius, category/subcategory, payment method, parking options. It is also capable of filtering on these criteria for restaurant POIs: price, cuisine, and atmosphere. For hotel POIs it can filter on reservation, hotel rate, and amenities. Figures 2, 3, and 4 show how the user interfaces of the application look for iOS and Android.

#### B. Technology Overview

The following technologies were used in this project and they will be explained in subsequent sections: Monocross, Xamarin Mono For Android, Xamarin MonoTouch, Windows Communication Foundation, and the Bing Maps API.

#### C. Monocross

Monocross is an open-source cross-platform mobile framework which uses the Mono framework – an open-source implementation of Microsoft’s .NET Framework and the C# language. As shown in Figure 5, Monocross utilizes the MVC design pattern to separate the data model (Model), presentation (View), and business logic (Controller) layers. This leads to reusable data model and business logic layers, and a platform-specific presentation layer. The platform-specific presentation layer affords developers the ability to use native controls if they choose; HTML5 is also an option.
Fig. 5. MonoCross MVC Design Pattern [MonoCross]. Shows the relationship between model, view, and controller. A platform container is created for each platform which contains the views.

To use Monocross on the Android platform Xamarin Mono for Android is required. Similarly, for the iOS platform Xamarin MonoTouch is required. A perpetual Xamarin business license currently costs $999 per developer per platform and provides 1 year of updates [Xamarin]. Academic licenses for students and professors are only $99 per developer per platform. Without a license, it is impossible to deploy applications to a mobile device that are greater than 32KB in size or applications that make calls to native third party calls using P/Invoke [Xamarin FAQ].

D. Xamarin Mono for Android

Mono for Android works by deploying a mobile version of Mono directly to mobile device along with the application source code [Xamarin Mono for Android Architecture]. This mobile Mono runtime runs side by side with the Dalvik Virtual Machine and is able to interpret C#. The Mono runtime and Dalvik Virtual Machine are both written in C and are able to communicate with each other using Java Native Interface (JNI) bridges.

E. Xamarin MonoTouch

MonoTouch works by Ahead of Time (AOT) compiling C# code directly into native arm code using the Mono runtime code generator with the --full-aot option [AOT]. Two code generation engines can be used: Traditional, LLVM-optimized [MonoTouch Compilation].

F. Windows Communication Foundation

Windows Communication Foundation (WCF) is Microsoft’s framework for web service communication. The Bing Maps API is a WCF SOAP service.

G. Bing Maps API

Microsoft’s Bing Maps API comprises four WCF web services: imagery, geocode, route, and search. Only search and geocode were used for this application so the others will be considered out of scope. The search web service is what is used to obtain POI search results. It provides data such as entity name, phone number, and address. The geocode web service is capable of performing a geocode operation, which is using an address to obtain GPS coordinates to obtain an address or it can perform the opposite, which is known as a reverse-geocode operation.

H. Application Architecture

The MonoCross application was structured according to the reference design that MonoCross has provided through samples. As such, the project is separated into several projects on both platforms. These projects are separated as follows: a handful of platform-specific framework-related projects that are provided by MonoCross, a shared project that contains the controllers, a shared project that contains the models, and a platform-specific container project that contains the views. Figure 6 shows that all of the business logic of the application is shared between both platforms including the controllers, models, and web service related code.

![Fig. 6. Shared Business Logic. Image shown is for the iOS application. The Android application references the same files, but the projects have the “.MD” extension instead of the “.MT” extension. VirtualEarthWebServices.cs is a dynamically generated WCF proxy. ServiceConfiguration.cs is used for settings of the proxy such as bindings and API key. Finally, App.cs is used a platform agnostic entry point.](image)

Models used in this application such as Location (shown in Figure 7) are populated by the controllers, which utilize the VirtualEarthWebServices proxy to obtain data. In the case of the Location model, LocationController populates it, and then passes it into LocationView.
The only differences in user-created code across the two platforms lie in the platform container projects. Figures 8 and 9 show that the same four views exist in both platform container projects. The views on each platform are implemented using the MonoCross framework projects for that specific platform and references to either Xamarin MonoTouch projects or Xamarin Mono For Android.

In the Android platform container’s Resources directory, XML is used to create menus that appear when the option button is pressed. This XML contains both the icons and text that is displayed. In the iOS platform container, the views themselves take care of this functionality.

The platform agnostic file storage that Monocross provides was employed. This was used to satisfy the goal of persisting the settings across multiple usages. The exact same code is used on both platforms. Monocross makes file storage on the iOS and Android platforms a trivial task by providing encryption by default, providing a “DataPath” property that is set to the correct place for each platform, providing serialization factories, and several file manipulation methods including read, save, and exists. [WROX Multi-platform]. Only five lines of code were used to implement encrypted storage as shown in Figure 10.

LINQ Expressions allow programmers to query or update collections in a way that is similar to writing standard Structured Query Language (SQL) queries. Queries with the where operator were used to easily filter List collections that contained Location objects and a join operator was used to match categories to subcategories. This resulted in a reduction of code written.

Object Initializers in C# allow programmers to provide properties of a class during the creation of an object. This technique was used heavily in the web-service communication code where complex object types are passed. Lines of code were significantly reduced due to this technique even as compared to the C# examples Microsoft provided for use of the Bing API.
B. PhoneGap

Phone also allows web designers to enter mobile application development using skills that they already know such as HTML, CSS, and JavaScript. Native controls are lacking and performance is questionable.

C. Rhodes

The Ruby language is known to be one of the quicker scripting languages and it is also thought by some in the development world to be able to achieve more in less lines of code. This approach seems to be targeted primarily at business applications. It is capable of creating native user interfaces with adequate performance.

D. Corona

This approach seems to be conducive to creating 2D games. While the Lua scripting language is quick to learn and fast executing it is targeted at game developers rather than business application developers. The Corona Cloud API is also clearly targeted at rapidly creating games.

E. Marmalade

Marmalade is capable of reaching the most mobile platforms. However, to create Marmalade applications developers need to use C++, which can be more time consuming than alternatives. As with Corona, the target audience for Marmalade is 2D game developers. There are not many samples targeted at business developers.

F. MonoCross

Due to its use of C# as the language, facilities are provided that increase developer productivity such as LINQ, Object Initializers, and built-in proxy generation in the case of WCF. It is capable of producing a fully native user interface that perform like applications that were written in intended language of the platform.

V. FUTURE WORK

A. Performance Evaluation

Performance of mobile applications is often evaluated using non-precise techniques such as qualitative comparison instead of quantitative. Often it is assumed that developing a mobile application in the language that is native to a platform will significantly outperform alternatives. While this may be true for some cases it may not be true for others as differences in performance may be negligible or non-existent.

To perform an accurate qualitative comparison it would be necessary to obtain a cross-section of performance data that represents the diversity of applications in existence today including the varying mobile platforms, technologies employed, and application types. This data would allow companies to determine which mobile framework would best meet their needs given their specific application type, platform, and technologies they wish to employ. For instance, a game could be compared in terms of frames per second across various mobile platforms for a specific device such as the iPhone 5. However, performance is not simply limited to games or frames per second. Performance tests across mobile frameworks could be carried out to measure read/write speed of storage, time to serialize and/or deserialize objects, web-service connection speeds/delays, and time-elapsed for compute-intensive algorithms to execute.

B. Evaluating Additional Mobile Frameworks

Additional frameworks will satisfy the selection criteria as they develop. Performing this comparison on a regular basis would be needed to keep information up-to-date.

Another mobile framework that could be included in future evaluations is Unity, which allows developers to create 3D games using the C# language.

C. Code Sharing

Future evaluations could examine application development and maintenance from an end-to-end perspective in an attempt to quantify the expense saved by having a single code base instead of two or more.

APPENDIX

A. Exploration with MonoCross

At the time that the project was began Xamarin 2.0 had not been released yet. This meant that MonoDevelop, MonoTouch, and Mono for Android had to be installed manually. With the release of version 2.0, these components are now installed through a single installer.

Exploration began by reading through the ‘BestSellers’ and ‘CustomerManagement’ sample applications, which are a part of the MonoCross 1.2 Examples available from the monocross.net website. BestSellers would compile for both platforms, but was crashing at runtime in both the Android simulator and on actual Android device. CustomerManagement, on the other hand, was unable to compile for Android. The compile-time error message indicated that the project had an inconsistent android v4 support jar. Ultimately, the issue was alleviated by copying the android v4 support jar file (that was generated through compilation of the project) in the bin directory on top of the one in the project’s SupportLib directory.

B. Exploration with Windows Communication Foundation

Windows Communication Foundation is currently at the edge of what Mono currently supports. Although WCF was added as of .NET 3.5 and Xamarin is currently using .NET 4.0, WCF has been classified by the Xamarin team as being in the preview stage. Not all facets of WCF are supported. For instance, only the BasicHttpBinding is currently permitted to be used [Xamarin BasicHttpBinding]. This rules out other protocols such as NetTcpBinding. The limitations of what it supports are similar to that of Silverlight at the time of writing.

Proxy Generation -- With Xamarin 2.0, an “Add Web Reference” right click menu option was added akin to that of Visual Studio. It does claim to support both WCF, but it was unable to import my web service due to an error in the web service. For this reason, to generate the web service proxy I used a Windows machine and this scvutil command: svccutil.exe /target:code /language:csharp /out:VirtualEarthWebServices http://dev.virtualearth.net/webservices/v1/searches
This command outputted a single C# file which was then imported into the solution.

Unexpected Failure -- After importing the proxy, the sample code Microsoft provided to contact a web method on their imagery service was tried. The sample worked correctly on Android, but not iOS. iOS was throwing this error: MonoTouch does not support dynamic proxy code generation. Override this method or its caller to return specific client proxy instance

WCF relies on the ability to dynamically implement the client service interface contained within the C# file the proxy file that was generated above via reflection; MonoTouch does not have support for reflection because the code that is produced is machine-level object code and not an intermediate language. The CreateChannel method and all web methods are normally created at runtime. The reason why Android works is because reflection is still possible because the code in execution is an intermediate language.

To alleviate the WCF issue on iOS, the WCF Channel had to be implemented manually. Providing an implementation for the channel allows the application to avoid the use of dynamic code generation. After overriding the implementation of the Channel the sample on both iOS and Android worked correctly. Figure 11 shows the solution that was used to solve this problem.

```csharp
public partial class GeocodeServiceClient
{
    protected override IGeocodeService CreateChannel()
    {
        return new GeocodeServiceClientChannel(this);
    }
}

private class GeocodeServiceClientChannel : ChannelBase<IGeocodeService>, IGeocodeService
{
    public GeocodeServiceClientChannel(ClientBase<IGeocodeService> client) : base(client) {
    }

generic GeocodeResponse Geocode(IGeocodeRequest request)
{
    return (GeocodeResponse)base.Invoke("Geocode", new object[] { request });
}

generic GeocodeResponse ReverseGeocode(IGeocodeRequest request)
{
    return (GeocodeResponse)base.Invoke("ReverseGeocode", new object[] { request });
}
}
```

Fig. 11. Fixing iOS Code Generation. Geocode and ReverseGeocode methods of the WCF channel are implemented manually to resolve the issue of lacking dynamic code generation.

ACKNOWLEDGMENT

B. M. Zimmerman thanks the MonoCross development team for supporting the open source community with regular releases of their Cross-Platform Mobile Framework.

REFERENCES

[Gartner]
http://www.gartner.com/newsroom/id/2335616

[Markus Falk]
http://www.markus-falk.com/mobile-frameworks-comparison-chart/

[Appcelerator]
http://www.appcelerator.com/

[Titanium]

[Appcelerator Quickstart]
http://docs.appcelerator.com/titanium/latest/#!/guide/Quick_Start

[Dealing with SOAP]
http://docs.appcelerator.com/titanium/latest/#!/guide/Dealing_with_SOAP_Web_Services

[PhoneGap]
http://www.phonegap.com

[PhoneGap Support]
http://phonegap.com/support/

[PhoneGap Getting Started]
https://build.phonegap.com/docs/start

[Apache Cordova]
http://cordova.apache.org/

[Cordova Plugin]
http://cordova.apache.org/docs/en/2.5.0/guide_plugin-development_ios_index.md.html

[PhoneGap API]
http://docs.phonegap.com/en/2.5.0/index.html

[SenchaTouch MVC]
http://docs.sencha.com/touch/2-0/#!/guide/apps_intro

[Rhodes]
http://www.rhomobile.com

[Rhodes Licensing]
http://www.rhomobile.com/rhodes/simplified-licensing/

[Rhodes Architecture]
http://docs.rhomobile.com/architecturefaq

[Rhodes API Support]
http://docs.rhomobile.com/rhoelements/apicompatibility

[Rhodes Web Services]
http://docs.rhomobile.com/rhodes/connect-to-web-services

[Rhodes CSS]
http://docs.rhomobile.com/rhodes/css-framework

[RhoConnect Push]
http://docs.rhomobile.com/rhoconnect/push

[RhoHub]
https://app.rhohub.com/

[Corona FAQ]
http://www.coronalabs.com/resources/frequently-asked-questions/

[Lua]
http://www.lua.org/about.html

[Corona]
http://www.coronalabs.com

[Corona Games]
http://www.coronalabs.com/i-want-to-build/games/

[Corona Device Specific]
http://developer.coronalabs.com/content/building-devices-android

[Corona Web Services]
http://docs.coronalabs.com/api/library/network/request.html

[Marmalade]
http://www.madewithmarmalade.com

[Marmalade Platforms]
https://www.madewithmarmalade.com/marmaladesdk/support-platforms

[Marmalade Buy]
http://www.madewithmarmalade.com/buy

[Marmalade API Documentation]
http://docs.madewithmarmalade.com/native/api_reference/s3eapidocumentation.html

[Marmalade Web Services]
http://docs.madewithmarmalade.com/native/api_reference/api/group_iwhttpgroup.html

[Marmalade Quick]
http://www.madewithmarmalade.com/quick

[Marmalade Juice]
http://www.madewithmarmalade.com/juice

[MonoCross]
http://monocross.net/

[Xamarin Mono for Android Architecture]
http://docs.xamarin.com/guides/android/advanced_topics/architecture

[Xamarin FAQ]
http://xamarin.com/faq

[AOT]

http://www.mono-project.com/AOT

[MonoTouch Compilation]
http://docs.xamarin.com/guides/ios/advanced_topics/compilation

[WROX Multi-platform]

[Xamarin BasicHttpBinding]
http://docs.xamarin.com/guides/cross-platform/application_fundamentals/introduction_to_web_services