International Journal of Business and Technology

Volume 1
Issue 2 *Spring 2013*

Article 6

May 2013

Mobile-cloud Cross Development (McX)

Sachin Ahuja

Mobile-cloud Cross Development (McX)

Sachin Ahuja

North Brunswick, NJ 08902, USA

Abstract. There is a multitude of Mobile Operating Systems (MOSs) with iOS, Android, Windows Phone and, BlackBerry leading the space. New players continue to enter the market. Without a de-facto leader in this space, it has become necessary for businesses & developers to target multiple devices & MOSs in order to establish a relevant presence within their target audience. Cross-platform Mobile Development Tools (XMTs) were born out of this need to reduce developer effort in creating mobile applications by providing "write once run anywhere" (WORA) functionality. However, most of these tools sacrifice performance, features or maintainability in order to provide WORA functionality. Furthermore, these tools only attempt to manage the user interface and related client-side functionality. Most mobile applications need to follow the same principals that guide development of non-mobile web or desktop apps. Typical apps are deployed using an n-tier, cloud-based strategy with substantial functionality delegated to cloud resources. Given the above, there are two parts of an application's anatomy that don't get much attention – the cloud middleware functionality, and the database/model management features. In this paper I address these problems through creation of a Mobile-cloud Cross Development (McX) tool-chain that includes a type-safe meta-programming language, an integrated cloud node and, an active compiler. In order to effectively understand the problem with the current state of the art, I use 3 of the leading XMTs alongside the developed McX tool-chain and compare the effectiveness of each. The paper further introduces the language; it's grammar and semantic structure, and provides discussions on how this approach fits the future of cross-platform, cloud-integrated mobile application development along with the associated issues and areas for further research.

Keywords— software engineering, mobile applications, cross-platform development, metaprogramming, domain-specific language

1. Introduction

The need for Cross-platform Mobile Tools (XMTs) owing to multiple Mobile Operating Systems (MOSs) and fragmentation even within various versions of a given MOS has been well documented and researched [1] [2] [4] [5] [7] [8] [9] [10] [13]. All these tools are aimed at enhancing developer productivity and reducing an application's Time-to-Market (TTM). However, impact of these tools on other important factors like performance, size, and feature-parity with purely native applications has been found wanting [2]. Since most XMTs are architected as interpreters [2], they also lag the native MOS Software Development Kits (SDKs) in new features and capabilities.

In addition to the above mobile-specific issues, there is a larger issue associated with application development itself. Almost all "real world" applications are more than just a mobile front-end with some interesting client-side functionality. Typical mobile applications have all the ingredients of a traditional web or desktop system – domain models, business logic and persistent databases. In order to provide these and other mobile-specific back-end services, there has recently been an influx of Mobile Back-end as a Service (MBaaS) offerings that provide a wide variety of back-end features, from user management to push notifications. These MBaaSs provide substantial benefit to developers through extremely high level of reuse and robust infrastructure. Most applications today also connect to a variety of third-party in-premise or hosted systems for business-relevant functionality. Almost all of these third-party systems expose their functionality either via web services or through a standardized Application Programming Interface (API) implemented in various leading programming languages. Even with the advent of these MBaaS services, web services and APIs, there always are application and/or business specific requirements that need to be handled through custom developed back-end services running typically on middleware infrastructure.

Given all of the above, a typical mobile application today, more than ever, needs to follow distributed n-tier architecture with a robust middleware that enhances decoupling and modularity while at the same time providing reliability, scalability and maintainability necessary for an enterprise-grade application

Identifying this architecture also brings to light the various tools and systems needed in creating the application:

Protocol server for communication from/to the mobile application.

Application or middleware server for hosting the custom business logic and glue-code necessary for communicating with MBaaSs, Web services, APIs and databases. Database servers to provide persistent storage. Additionally, various application frameworks are needed to provide functionality that has become a norm for Rich Internet Applications (RIAs): server and client-side Model-View-Controller (MVC) frameworks, view-level logic templates, object-relational mapping (ORM) frameworks etc.

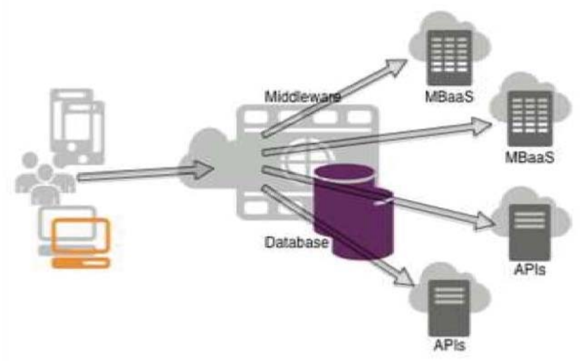


Fig 1. Typical Architecture of a Mobile Application

Understanding the expanse of architecture needs coupled with application needs provides us with appreciation of the fact that XMTs cover but a very small portion of the overall application development lifecycle. In fact, connectivity and data-interchange limitations of these XMTs may even negatively impact developer productivity.

2. Mobile-cloud cross development tools (mcx)

Even with the shortcomings noted above, and given the fact that they cover a fairly small portion of the application development lifecycle, the appeal of XMTs is on the rise purely because the perceived benefit of "write-once run anywhere" (WORA) functionality is very high. Since the primary benefits of WORA are enhanced developer productivity and source code maintainability, we can safely assume that a tool-chain that reduces development effort at each layer of the architecture, that provides the WORA functionality and, that drastically reduces limitations, will be desired and used by the development community at large. Such a tool-chain is what we call Mobile-cloud Cross Development (McX) tool-chain.

Following are the necessary features of a McX tool-chain:

1. Provide WORA functionality
2. Generate optimized native applications instead of low-performing interpreted byte-code.
3. Provide strong access to device functionality
4. Provide capability to write server-side code in the same language as that used for writing the mobile code.
5. Provide integrated middleware to host the custom business logic and glue code for integrating with MBaaSs, web services and APIs.
6. Provide integrated model management framework to obviate the need for external objectrelational mapping tools and frameworks.
7. Provide integrated database access layer.
8. Provide integrated protocol server to facilitate communication between the mobile application and middleware.
9. Provide support for extending the tool-chain to gain access to new features and capabilities introduced by MOS vendors as well as cloud infrastructure providers.
10. Provide support features such as testing and deployment.

Bonus points for providing mobile specific and general back-end services such as push notification, geo-location, connectivity with external APIs, authentication, user management etc.

In order to understand the current state of the art, we take three of the leading XMTs that also provide some level of cloud integration and evaluate them against the 10-point McX checklist introduced earlier in this section.

Appcelerator Titanium: Developers use the Titanium Studio Integrated Development Environment (IDE) and the JavaScript based Titanium SDK to build cross platform apps. The IDE is built on top of the popular Eclipse IDE.

All source code, including user interface, is written using JavaScript. Following the company's acquisition of Cocoafish MBaaS provider, a number of back-end services are integrated within the SDK and are easily accessible from within the IDE.

Table 1. Titanium feature checklist

Feature	Support
WORA	Yes
Native Apps	No
Access to Device Functionality	Yes
Write Server-side Code	No*
Middleware	No
ORM	No
Database Access Layer	Yes
Protocol Server	No
Extensibility	No
Testing and deployment support	No*
* Since Titanium uses JavaScript, server-side code and deployment can be done using NodeJS or similar server-side JavaScript technology stack. However, setting up and integrating those into applications is non-trivial and would be up to the developers.	

Motorola Rhomobile Suite: This suite consists of three parts: RhoStudio, a plug-in for the popular Eclipse IDE, it provides the development environment for Rhomobile applications; RhoConnect, the mobile middleware provides connectivity to off-the-shelf as well as custom back-end systems developed in Ruby, Java or .NET; and RhoElements, and HTML5 GUI library that is deployed within web-views of native apps to create the cross-platform presentation layer. Client-level business logic is written in the Ruby programming language.

Table 2. Rhomobile feature checklist

Feature	Support
WORA	Yes
Native Apps	No
Access to Device Functionality	Yes
Write Server-side Code	No*
Middleware	Yes
ORM	No
Database Access Layer	Yes
Protocol Server	No
Extensibility	No
Testing and deployment support	No*
* Since Rhomobile uses Ruby, server-side code and deployment can be done using Ruby on Rails or similar server-side Ruby technology stack. However, setting up and integrating those into applications is non-trivial and would be up to the developers.	

Apache Cordova: This HTML5 based framework provides JavaScript based bridged access to native device features. Coupled with one of the many mobile specific JavaScript UI libraries like JQuery Mobile or Dojo, developers can create hybrid native apps using a pure web stack – HTML5, CSS3 and JavaScript. Since this framework uses pure web stack, it is quite popular within the developer community.

Table 3. cordova feature checklist

Feature	**Support**
WORA	Yes
Native Apps	No
Access to Device Functionality	Yes
Write Server-side Code	No*
Middleware	No
ORM	No
Database Access Layer	No
Protocol Server	No
Extensibility	No
Testing and deployment support	No*
* Since Cordova uses JavaScript, server-side code and deployment can be done using NodeJS or similar server-side JavaScript technology stack. However, setting up and integrating those into applications is non-trivial and would be up to the developers.	

3. The proposed tool-chain

The simple analysis provided above shows that none of the XMTs provide more that 50% of the functionality needed to have any significant positive impact on developer productivity. It further shows that all the leading systems work as interpreters of byte-code that reduces the performance of created applications. Based on similar research [1], a Domain Specific Language (DSL) was deemed to be one of the most effective approaches in providing a relevant level of abstraction to the developers. Given the complexities of mobile application development as discussed above, a relevantly high level of abstraction is exactly what can solve the most amount of problems faced in creating cloud-connected, cross-platform mobile apps. However, going a step further, this language needs to be more "Turing Complete" than a DSL because the "domain" it caters to is "programming".
The proposed tool-chain, then, aims to provide a level of abstraction high enough to mask most application programming expressions and constructs. It also provides integrated support for mobile application development best practices that obviate the need for using most oft-used frameworks and plugins. It also provides contextual behaviour to source-code that is normally implemented through intelligent use of design patterns.

Components of the mcx tool-chain

The components of the proposed McX tool-chain enable creation of cloud-integrated mobile applications through semantic evaluation of source-code that is translated into pure native mobile applications and middleware components based on context.

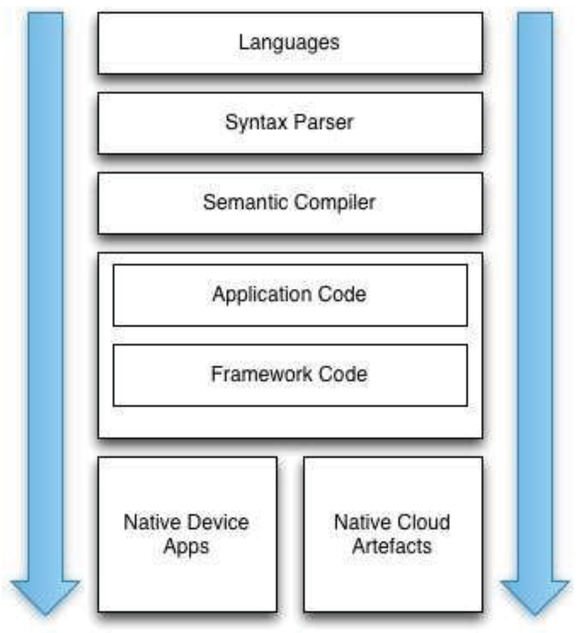


Fig 2. Components of the McX Tool-chain

The Languages

The core of the proposed McX tool-chain is a meta-programming language called "Mobile app semantic code" (Masc). As the name suggests, the language provides a lot of semantic or contextual behaviour to source-code based on its expected location and purpose of deployment.

Masc is the language for implementing logic on both the client and cloud. It provides type safety and limited object-orientation that enhances predictability and error discovery. At the same time, the language provides the fluidity of scripting languages that aid in more agile and fast-paced development cycles.

The syntactic structure of Masc follows the offside rule like that found in Python. It provides all relevant programming constructs like variable definition, method definition, loops and decision constructs, instance and type level artefacts as well as referential integrity.

A specialized form of Masc called "Mobile app markup language" (Maml), is also provided to define user interface elements. Maml provides a much simpler way for developers to define user interfaces. It also enforces and facilitates view development best practices through semantic intervention.

The Compiler

The compiler operates in three phases. In the first phase, the compiler parses the input Masc and Maml files to generate an Abstract Syntax Tree (AST). In the second phase, the compiler validates the semantics and contextual rules. The contextual rules refer to specialized behaviour that is afforded to Types based on their position in the source tree. As an example, a Masc file in the Application's "view" folder may not have access to artefacts in the "cloud" folder. This specialized contextual validation is driven by a strict set of rules that are dependent on the structure of the source tree. Thus, one limitation of this approach is the inflexibility of the source tree structure. However, the proposed structure follows the most desired format based on discussions with several developers.

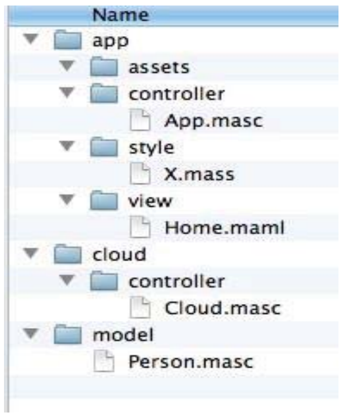


Fig 3. Structure of a McX App

The third phase of the compilation process produces the source code for the various native applications as well the middleware components. These are enhanced by framework code to provide pattern-based best practices support. One of the most important features of the compiler is that it operates as an active console that constantly watches changes in the source tree and triggers the compilation sequence for each changed file as it changes. This provides a level of agility in a type-safe stack that is comparable to scripting languages.

The Cloud Node

The tool-chain also includes an integrated cloud node complete with a protocol server and a middleware. The middleware components generated by the compiler are deployed to this node and can be instantly tested, built and deployed to production environments as well. The framework compoennts of the cloud node provide features that enable connectivity with APIs, MBaaS providers and Web services along with access to database servers.

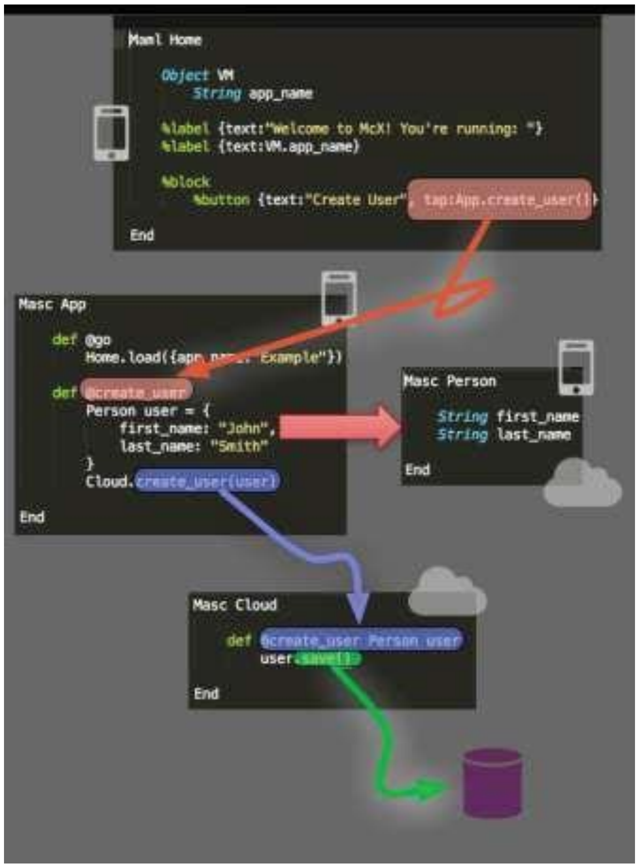
The Framework Components

We have introduced the notion on framework components in B & C above. These components are pre-created native source libraries that provide some compelling behaviour or service to the created applications. Following are some of the included framework components:

Component	**Location**
Model-view View Model (MVVM)	Mobile
Client-side MVC	Mobile
ORM	Cloud
ORM	Client
Network communication	Client
API, Web Service, MBaaS Integration	Cloud
Authorization and Authentication	Both
Pagination and Caching	Both
Encryption/Decryption	Both

A MCX Example

Here’s a small example that aptly shows the features of the tool-chain.



```
Maml Home

    Object VM
        String app_name

    %label {text:"Welcome to McX! You're running: "}
    %label {text:VM.app_name}

    %block
        %button {text:"Create User", tap:App.create_user()}

End
```

```
Masc App

    def @go
        Home.load({app_name: "Example"})

    def @create_user
        Person user = {
            first_name: "John",
            last_name: "Smith"
        }
        Cloud.create_user(user)

End
```

```
Masc Person

    String first_name
    String last_name

End
```

```
Masc Cloud

    def @create_user Person user
        user.save()

End
```

Fig 4. McX in Action

The anatomy of this app is the following:

1. The view (Maml) effortlessly engages the Application Controller (App.masc) following an MVC pattern.
2. The Application Controller has access to the domain Model (Person.masc) and can easily create an instance of the same.
3. The Model is the shared between the Mobile and Cloud deployments. This not only acts as the domain representation, but also can be used as data-interchange between the layers.
4. The Cloud controller (Cloud.masc) resides on the cloud middleware. However, engaging it from within the App controller hides all the complexities behind network communication, asynchronous threads and object serialization as well as deserialization.
5. The Cloud controller simply saves the object in the persistent datastore. Again, this functionality is provides at a very high level of abstraction through included framework components.

Lets validate this against the 10-point McX checklist.

Table 4. MCX feature checklist

Feature	**Support**
WORA	Yes
Native Apps	Yes
Access to Device Functionality	Yes
Write Server-side Code	Yes
Middleware	Yes
ORM	Yes
Database Access Layer	Yes
Protocol Server	Yes
Extensibility	Yes
Testing and deployment support	Yes

Conclusions

Based on the research, we've gleaned a few facts:
Developing a mobile application is quite similar in approach to developing a traditional web or desktop application.
With multiple MOSs complicating the client-side and a combination of MBaaS, Web
Services, APIs and custom needs complicating the server-side; the complexity of developing a mobile application is even higher than that for developing traditional web or desktop application. The current cross-platform tools only cover a small portion of the overall application development lifecycle.
Developer productivity may even be reduced because of the limitations introduced by the XMTs.
The solution, then, is a tool-chain that reduces complexity at each layer of the application's architecture. The developer tools of the future will need to be provide higher levels of abstraction and may even become specialized to certain industries and business use cases.

References

1. Dean Kramer, Tony Clark and Samia Oussena, MobDSL: A Domain Specific Language for multiple mobile platform deployment, *Proceedings of the IEEE International Conference on Network Embedded Systems for Enterprise Applications, 2010*
2. Julian Ohrt and Volker Turau, Cross-Platform Development Tools for Smartphone Applications. Computer, 2012
3. Hoang T. Dinh, Chonho Lee, Dusit Niyato and Ping Wang, A Survey of mobile cloud computing: architecture, applications, and approaches. Wireless Communications and Mobile Computing, 2011.
4. Tarkoma, S.; Lagerspetz, E.; , "Arching over the Mobile Computing Chasm: Platforms and Runtimes," *Computer* , vol.44, no.4, pp.22-28, April 2011
5. Gavalas, D.; Economou, D.; , "Development Platforms for Mobile Applications: Status and Trends," *Software, IEEE* , vol.28, no.1, pp.77-86, Jan.-Feb. 2011
6. Balagtas-Fernandez, F.T.; Hussmann, H.; , "Model-Driven Development of Mobile Applications," *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on* , vol., no., pp.509-512, 15-19 Sept. 2008
7. Smutny, P.; , "Mobile development tools and cross-platform solutions," *Carpathian Control Conference (ICCC), 2012 13th International* , vol., no., pp.653-656, 28-31 May 2012
8. Bin Zhang; Tian-gang Xu; Wei Wang; Xia Jia; , "Research and implementation of cross-platform development of mobile widget," *Communication Software and Networks (ICCSN), 2011 IEEE*
    a. *3rd International Conference on* , vol., no., pp.146-150, 27-29 May 2011
9. Anand Iyer, Amul Jadhav, Nilesh Dhangare, "Common Platform for Mobile Application", Advances in Computer Science and its Applications, vol.1, no.2, pp.174-184, 2012
10. Biao Pan; Kun Xiao; Lei Luo; , "Component-based Mobile Web Application of Crossplatform," *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on* , vol., no., pp.2072-2077, June 29 2010-July 1 2010
11. Andre Charland and Brian Leroux: , "Mobile Application Development: Web vs. Native", Communications of the ACM, vol.54, Issue 5, pp.49-53, May 2011
12. Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta: , "Applying Model-Based Techniques to the Development of UIs for Mobile Computers", Proc. of the 6th International Conference on Intelligent User Interfaces, pp.69-76, 2001
13. Tony Wasserman. "Software Engineering Issues for Mobile Application Development" *FoSER 2010* (2010).