Ain Shams Engineering Journal (2017) 8, 163–190

Ain Shams University

Ain Shams Engineering Journal

www.elsevier.com/locate/asej
www.sciencedirect.com

ELECTRICAL ENGINEERING

Taxonomy of Cross-Platform Mobile Applications Development Approaches

Wafaa S. El-Kassas [*], Bassem A. Abdullah, Ahmed H. Yousef, Ayman M. Wahba

Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Egypt

Received 13 September 2014; revised 30 May 2015; accepted 3 August 2015
Available online 23 October 2015

KEYWORDS

Cross-platform mobile development;
Interpretation approach;
Cloud computing;
Compilation approach;
Component-Based approach;
Model-Driven Engineering

Abstract The developers use the cross-platform mobile development solutions to develop the mobile application once and run it on many platforms. Many of these cross-platform solutions are still under research and development. Also, these solutions are based on different approaches such as Cross-Compilation approach, Virtual Machine approach, and Web-Based approach. There are many survey papers about the cross-platform mobile development solutions but they do not include the most recent approaches, including Component-Based approach, Cloud-Based approach, and Merged approach. The main aim of this paper is helping the researchers to know the most recent approaches and the open research issues. This paper surveys the existing cross-platform mobile development approaches and attempts to provide a global view: it thoroughly introduces a comprehensive categorization to the cross-platform approaches, defines the pros and cons of each approach, explains sample solutions per approach, compares the cross-platform mobile development solutions, and ends with the open research areas.

1. Introduction

In the third quarter of 2014, the sales of smartphones to end users grew 20.3% to reach 301 million units. Besides, smartphones accounted for 66% of the total mobile phone market. Gartner expects that by 2018, nine out of 10 phones will be smartphones [1]. Smartphone users increase every year because of the variety of mobile applications (Apps) offered to the users in the App stores. With the huge number of smartphone users, there is a growing need to develop more Apps that serve their needs in different fields such as education [2,3], health [4], and tourism [5]. There are many smartphone vendors in the market and each vendor uses a specific platform. These platforms include Android, Windows Phone, iOS, BlackBerry OS, Symbian, and others. A comparison between three of the most common mobile platforms (Android, iOS, and Windows Phone) is shown in Table 1.

The aim of any mobile development company is to target as many users as possible by providing the same App for different platforms. Each platform vendor provides the developers with different Integrated Development Environments (IDEs), programming languages, APIs, and Apps distribution market

[*] Corresponding author.
E-mail addresses: wafaa.elkassas@gmail.com (W.S. El-Kassas), babdullah@eng.asu.edu.eg (B.A. Abdullah), ahassan@eng.asu.edu.eg (A.H. Yousef), ayman.wahba@eng.asu.edu.eg (A.M. Wahba).

Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.asej.2015.08.004
2090-4479 © 2015 Ain Shams University. Production and hosting by Elsevier B.V.

Table 1 Some differences between the top three common platforms [6].

	Virtual machine	Programming language	User interface	Memory management	IDE	Platform	Devices	App market
Android	Dalvik VM	Java	XML files	Garbage collector	Eclipse	Multi-platform	Heterogeneous	Google Play Store
iOS	No	Objective-C	Cocoa Touch	Reference counting	XCode	Mac OS X	Homogenous	iTunes Apps Store
Windows Phone 7	CLR	C# and .Net	XAML files	Garbage collector	Visual Studio	Windows Vista/7	Homogenous	Windows Phone Store

(store). Therefore, the mobile development company has to choose one of two alternatives to develop the same App for different platforms. The first alternative is that the developers work together to produce one App for a specific platform at a time. This alternative will support the different platforms sequentially, but wastes a lot of development time in learning and getting familiar with the different platforms' development environments. Another alternative is that the developers are divided into separate teams and each team works for a specific platform. This alternative will support the different platforms in parallel, but may require more developers than the first alternative thus more costly. There is always a need to deliver the App faster and more economically by saving the development time and efforts. This problem leads to the existence of the cross-platform mobile development solutions. The main concept of the cross-platform solutions is to develop the App once and run it anywhere. The traditional software development lifecycle is shown in the left part of Fig. 1. The cross-platform solutions extend the software development lifecycle by writing the App once and deploying it many times to support different platforms as shown in the right part of Fig. 1. Many of the cross-platform mobile development solutions are still under research and development. Some solutions are available in commercial use but till now the final solution that solves the entire cross-platform mobile development problem does not exist.

This paper is structured as follows: related work is presented in Section 2. An overview about mobile application development is found in Section 3. Cross-platform mobile development approaches are categorized and sample solutions (per approach) are described in details in Section 4. The cross-platform mobile development solutions are compared in Section 5. The open research areas are introduced in Section 6. The conclusion is presented in Section 7.

2. Related work

There are many survey papers about the cross-platform mobile development approaches and solutions. In [8], Holzinger et al. concentrate on the mobile User Interface (UI) issues. The mobile devices have different screen sizes and different aspect ratios, so efforts are required to ensure that the interface is well scalable. The authors encourage the developers to clearly separate the UI definitions from the rest of the code base. The paper compares the advantages of the HTML5 web Apps versus the native Apps. To decide whether to use web or native Apps, the authors define a set of selection factors including the App's intended features, the target audience, and the development team skills. The paper briefly describes three cross-platform frameworks: JQuery Mobile, PhoneGap, and Titanium Mobile. After using these frameworks to produce cross-platform Apps, it is both the responsibility of the designers and the developers to ensure that the Apps run properly on all the target platforms and devices because none of these frameworks can substitute thorough Apps testing which can be done on actual devices or on various configurations of the emulators provided by the platform IDEs.

In [9], Raj and Tolety classify the cross-platform mobile development approaches into four approaches: **Web** Approach, **Hybrid** Approach, **Interpreted** Approach, and **Cross-Compiled** Approach. These approaches were described along with the advantages and challenges of each approach. The authors recommend that the developers select the appropriate approach based on the App types. The App types are classified by the authors into **server data-driven**, **sensor/IO based, standalone**, and **client–server** Apps. This paper focuses only on a subset of the existing cross-platform mobile development approaches but does not explain how the existing solutions implement these approaches.

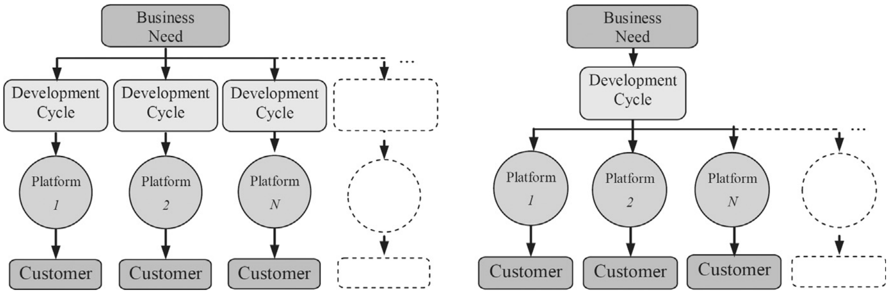


Figure 1 Traditional and cross-platform development models [7].

In [10], Smutny categorizes the mobile web and native Apps into four different configurations: **Native** Apps, **Hybrid** Apps, **Dedicated web App** tailored to a specific platform, and **Generic mobile App** which is a mobile website that runs on any platform. The mobile web development frameworks provide common characteristics to the delivered App including the following: (1) **cross-platform** as it supports different platforms, (2) **lightweight** by lowering the file size inside the framework due to current bandwidth limitations, (3) **optimized for touch-screen devices**, and (4) **uses HTML5 and CSS3 standards**. The title of this paper "Mobile development tools and cross-platform solutions" is a little bit misleading. We expected that the paper explains in details the different cross-platform mobile development approaches, but it mainly focuses on the development of a mobile web App (English-Czech dictionary for the automatic control terms) and briefly describes several HTML5 mobile development frameworks including Sencha Touch, jQTouch, jQuery Mobile, Wink, M-Project, Jo, Titanium, PhoneGap, and DHTMLX Touch.

In [11], Ribeiro and da Silva describe and analyze six cross-platform tools: Rhodes, PhoneGap, DragonRAD, Appcelerator Titanium, mobl, and Canappi mdsl. The main strengths and weaknesses of each tool are highlighted. The authors compare these tools based on these factors: the technology or cross-platform approach, supported platforms, development environment, type of the resulting App, and the device API support. The authors categorize the cross-platform approaches of the surveyed tools in four categories: **Runtime**, **Web-to-native wrapper**, **App Factory**, and **Domain Specific Language (DSL)**.

In [12], Palmieri et al. present a comparison of four cross-platform mobile development tools: Rhodes, PhoneGap, DragonRad and MoSync. The main target is to help the developers to evaluate and understand what is the appropriate tool matching their requirements. The comparison criteria include supported mobile platforms, tool licences, programming languages, availability of API's, accessibility to native API's, architecture, Model-View-Controller (MVC) pattern support, tool extensibility, and the IDEs.

In [13], Xanthopoulos and Xinogalos classify the cross-platform App types into **Web**, **Hybrid**, **Interpreted,** and **Generated** Apps. The authors do a comparative analysis to highlight the advantages and disadvantages of each type. The comparison criteria include market place deployment (distribution) possibility, widespread technologies used to develop Apps, hardware and data access, user interface and look and feel, and user-perceived performance (i.e. loading time and execution speed). The paper ends with a case study to develop a mobile App (RSS Reader) with the cross-platform mobile development tool "Titanium" using JavaScript without any knowledge of the target Android and iOS platforms.

In [14], Ohrt and Turau define two categories for the mobile Apps: **Integrated Apps** which are integrated into the system after installation and can access the system functionalities (include native or interpreted Apps) and **Nonintegrated Apps** which are invoked using separate tools like a web browser and mostly cannot access the system functionalities (include Java Midlets, Flash Apps, or web Apps). The authors provide an overview of nine cross-platform mobile development solutions: Flash Builder, Illumination Software Creator, LiveCode, Marmalade, MoSync, OpenPlug Studio, PhoneGap, RhoStudio, and Titanium. The main considerations of the authors are whether the cross-platform solution satisfies the developer needs and whether the resulting Apps meet user expectations regardless of the solutions' internal details. The nine cross-platform solutions are compared based on the following: the supported platforms of each tool, features and functionalities of the tool, and performance characteristics of a simple evaluation App created on the nine cross-platform solutions along with the native Android App.

In [15], Heitkötter et al. classify the cross-platform mobile development approaches to: (1) approaches that use a runtime environment like **Web Apps**, **Hybrid** approach, or **Self-contained environment** approach, and (2) approaches that generate platform-specific Apps from a common code base at compile time like **Model-Driven solutions** and **Cross-Compiling**. The authors concentrate on the first type of approaches. In order to evaluate the native App development, the authors selected three solutions, one from each kind of runtime environment. This include mobile web Apps, PhoneGap as a hybrid framework, and Titanium Mobile as a self-contained approach. The authors define a list of 14 criteria items from the infrastructure perspective (license and costs, supported platforms, access to platform-specific features, long-term feasibility, look and feel, application speed, and distribution) and development perspective (development environment, Graphical User Interface (GUI) design, ease of development, maintainability, scalability, opportunities for further development, and speed and cost of development) for these solutions evaluation.

In [6], Perchat et al. propose a new tool using the Component-Based and Cross-Compiler approaches. The authors classify the cross-platform solutions as follows: the **Cross-Compilers based solutions**, the solutions based on the **Model-Driven Engineering**, and finally the **Interpreters** of source code defined in two categories: the **Virtual Machines** (VMs) and **Web-Based** solutions.

Table 2 provides a comparison between the previous survey papers to show how they categorize the approaches of the cross-platform mobile development solutions. They use the following categorization methods:

- **No Categorization**: these survey papers compare a set of cross-platform solutions to help the developer choose the solution that matches his/her needs to deliver the required Apps.
- **App Categorization**: these survey papers categorize the cross-platform solutions based on the type of cross-platform App.
- **Approach Categorization**: these survey papers categorize the cross-platform solutions based on their approaches.

This paper uses the **"Approach Categorization**" method to compare the cross-platform solutions based on the used approaches and presents an enhanced categorization compared to the other survey papers [6,9,11,15]. This paper includes the most recent approaches that were not included in the previous surveys like **Component-Based approach**, **Cloud-Based approach**, and **Merged approach**. Also, this paper proposes sub-categorization for several approaches. This paper attempts to provide a global view to the existing cross-platform mobile development solutions. The main aim of this paper is helping the researchers to know the existing cross-platform mobile development approaches and the open

Table 2 Comparison of survey papers.

Survey paper	Author	Year	Proposed categorization for the cross-platform mobile development solutions	Categorization method
[8]	Holzinger	2012	Briefly describes three cross-platform frameworks: JQuery Mobile, PhoneGap, and Titanium Mobile	No Categorization
[9]	Raj	2012	Categorizes the cross-platform solutions based on 4 approaches: (1) **Web** Approach (2) **Hybrid** Approach (3) **Interpreted** Approach (4) **Cross-Compiled** Approach	Approach Categorization
[10]	Smutny	2012	Categorizes the mobile web and native Apps to four different configurations: **Native Apps, Hybrid Apps, Dedicated Web App** tailored to a specific platform, and **Generic mobile App** which is a mobile website run on any platform	App Categorization
[11]	Ribeiro	2012	Categorizes the surveyed solutions based on four categories: (1) Runtime (2) Web-to-native wrapper (3) App Factory (4) Domain Specific Language	Approach Categorization
[12]	Palmieri	2012	Compares four cross-platform mobile development solutions: Rhodes, PhoneGap, DragonRad and MoSync	No Categorization
[13]	Xanthopoulos	2013	Classifies the cross-platform App types to: **Web**, **Hybrid**, **Interpreted,** and **Generated** Apps	App Categorization
[14]	Ohrt	2012	Defines two categories for the mobile Apps: **Integrated Apps** (include native or interpreted Apps) and **Nonintegrated Apps** (include Java Midlets, Flash Apps, or Web Apps)	App Categorization
[15]	Heitkötter	2013	Classifies the cross-platform approaches to: (1) Approaches use a runtime environment (like **Web Apps**, a **Hybrid** approach, or **Self-contained environment** approach) (2) Approaches generate platform-specific Apps from a common code base at compile time (like **Model-Driven solutions** and **Cross-Compiling)**	Approach Categorization
[6]	Perchat	2013	Classifies the cross-platform solutions as follows: (1) The **Cross-Compilers** (2) The solutions based on the **Model-Driven Engineering** (3) The **Interpreters** of source code have two categories: **Virtual Machines** and **Web-Based** solutions	Approach Categorization

research areas in this field. The next section introduces the necessary background of the mobile application development.

3. Mobile applications development

Mobile Apps development is a special case of the software development as the developers have to consider different aspects such as short development lifecycle, the mobile device capabilities, mobility, the mobile device specifications like screen sizes, the App user interface design and navigation, security and user privacy, and the Apps require marketing to gain popularity. Many Apps are available in the stores to help for a life based on mobile, or mLife. This includes mTourism, mEnvironment, mEducation, mGovernment, mHealth, mEntertainment, and suchlike.

The mobile App development lifecycle consists of the following: (1) analysis of the App idea, (2) the user interface design, (3) the App development using the tools and programming languages of the target platform, (4) the App testing on different devices, and finally (5) publishing the App on the target platform store. New functions or updates to the App are released in successive versions of this App in the target platform store. To develop the mobile App for many platforms, the above development lifecycle is repeated for each target platform except the first step of analyzing the requirements of the App.

If the developer wants to develop the same App for all platforms, the following laboratory equipments are needed: one Personal Computer (PC) and one Mac (used for iOS Apps development), the development tools of each platform and the proper setup, at least one smartphone device and one tablet device for each platform and sometimes multiple devices for the same platform like Android because it is supported by many smartphones vendors [6]. Then, it is required to create one developer account for each platform store to publish the developed Apps. The next subsections explain the following: the restrictions and challenges of the mobile Apps development, the types of mobile Apps, and the cross-platform mobile development.

3.1. Challenges of mobile applications development

The development of mobile Apps has a lot of restrictions and challenges such as:

(1) Limited resources of the mobile devices although they are more powerful than before [16]:
- Limited computing power of mobile devices.
- Limited storage space for mobile devices.
- Connectivity for mobile devices will be affected by the movement.

(2) Heterogeneity of mobile operating systems:
- The different operating environments for Apps (i.e. the App developed by iOS SDK can only run on iOS devices, and the Android Apps can only run on Android devices) [17].
- To develop the same App on different Operating Systems (OS's), the developer has to learn how to develop on these different OS's SDKs using their programming languages and APIs [18].

(3) Heterogeneity of devices may require different versions of the same application [17]:
- Different computing capabilities and hardware configurations of devices have to be considered when developing Apps.
- Different screen size of devices: the screen of most mobile phones is smaller than 4-inch, the tablet screen sizes are 7-inch or 10-inch, and the smart TV screen can be larger than 60-inch.
- Different input methods: touch screen, keyboard, TV remote control, and TV in smart TV.

(4) The user experience: the developers have to define a simple and a user friendly interface for the developed applications [6].

(5) Application maintenance: frequent updates of the mobile platform may affect some Apps making them unusable in the new version; hence, maintenance and updates to these Apps are required [6]. Also the version management is difficult because the users may not update to the new version of the App when released [17]. The maintenance or updates of the App developed for different platforms means that the developer repeats the same updates in all versions of different platforms [18].

(6) Cross-platform development: the development of the same App for different platforms means repeating the same work several times because each platform vendor provides the developers with different programming languages and development tools.

Although the development of mobile Apps has a lot of restrictions and challenges, the main challenge that is addressed in this paper is how to develop the mobile App once and run it on different mobile platforms to save the time and efforts of the developers. This paper will introduce different approaches and solutions to solve this challenge and ends with the open research areas.

3.2. Mobile applications types

The types of the mobile Apps are ***web App***, ***native App***, and ***hybrid App***. These types will be explained in the following sub-sections.

3.2.1. Web App

The mobile web Apps are developed using the web technologies such as HTML, HTML5, JavaScript, and CSS. These Apps do not require to be installed from the store as they are accessed through a URL entered in the mobile web browsers. Table 3 shows the pros and cons of the web App type of mobile applications.

Table 3 Pros and cons of the web App type of mobile applications.

Pros	Cons
• Easy to learn and develop using the web technologies • All the processing is done on the server and only the UI is sent to the mobile for rendering • The maintenance of the App is simple because the updates of the App and the data are done on the server • The same App is developed once and can run on different platforms using the mobile web browsers	• The web Apps are not available in the store • Internet connection is needed to run the App • The web Apps cannot access the mobile device software and hardware such as camera, and GPS sensors [9] • Less performance because the interpreter language of HTML and JavaScript is parsed and implemented through web browsers [19]

Table 4 Pros and cons of the native App.

Pros	Cons
• Have full APIs to access all the mobile device features like camera, sensors, network access, GPS, file storage, database, SMS, and email • Higher performance than the web Apps • Native look and feel of the user interface	• Native Apps are more difficult to develop and require a high level of experience [13] • They need to be developed separately for each platform, hence increasing the development time, cost, and efforts [20] • Restrictions and costs associated with developing and deploying to certain platforms (i.e. Apple developer license and Apple's approval to distribute Apps to the iTunes Apps store) [10]

3.2.2. Native App

The mobile native Apps are developed using the tools and programming languages provided for a certain mobile platform. These Apps run only on mobiles with the target platform. The native App can be downloaded and installed from the store. Table 4 shows the pros and cons of the native App.

3.2.3. Hybrid App

The mobile hybrid App combines the web App and the native App. It is developed using the web technologies like the web App but it is rendered inside the native App using a web view control. The device capabilities are exposed to the hybrid App through an abstraction layer (JavaScript APIs) [9]. This App can be downloaded and installed from the store. Table 5 shows pros and cons of the hybrid App.

3.3. Cross-platform mobile applications development

The cross-platform solutions help the developers to write the source code once and run the produced mobile application on different platforms. In [13], Xanthopoulos and Xinogalos

Table 5 Pros and cons of the hybrid App.

Pros	Cons
• Can be used for both server backed and standalone Apps [9] • The application size is small if all the data are stored on the server • The App can access the device features • The user interface can be reused across different platforms [9]	• Less performance than the native App • The user interface will lack the look and feel of the native App. To achieve the native look and feel, the platform-specific styling might be required [9]

Table 6 Pros and cons of the cross-platform mobile development.

Pros	Cons
• The App is developed once and is available to more users who use different platforms • Ease of development as the App is written once and deployed many times on different platforms • Reduction of the development time and efforts	• The existing cross-platform solutions are still under research and most of the commercial ones are based on the web technologies such as HTML5 • Lack of last features introduced by OS's, because for each update done in OS's supported by the tool the vendor should update its own tool as well [12]

classify the cross-platform App types into (1) **Web App**, (2) **Hybrid App**, (3) **Interpreted App,** and (4) **Generated App**. The interpreted App and generated App types are subtypes of native Apps. There are many cross-platform mobile development tools and the target users of these tools differ according to the goals of each tool, like some tools target students for learning mobile development, doctors to customize applications for patients, non-technical people for producing simple applications, and professional developers for producing advanced applications. All these tools use different approaches that will be explained in more details in the next section. Table 6 shows the pros and cons of the cross-platform mobile development.

4. Cross-platform mobile development approaches

This paper proposes to categorize the cross-platform mobile development approaches into six main approaches: Compilation, Component-Based, Interpretation, Modeling, Cloud-Based, and Merged. Also, sub-categorization is proposed for the following approaches: Compilation, Interpretation, and Modeling. Table 7 explains the pros and cons of the different approaches and sub-approaches. Fig. 2 shows the six main approaches and their sub-approaches along with the cross-platform solutions for each approach/sub-approach. This categorization introduces an enhancement to the survey papers [6,9,11,15] as follows:

- Some approaches are not included in these survey papers such as Component-Based approach, Cloud-Based approach, and Merged approach.
- Some sub-approaches are not included in these survey papers such as Trans-Compiler sub-approach and MD-UID sub-approach.

Section 5 includes a comparison of the different cross-platform mobile development solutions described in this paper. The comparison includes the solution characteristics. For example, the system model or architecture, the target platforms, the type of the output mobile App (web, native, or hybrid), the product name and URL if exists, and the availability of the solution.

This section provides the following for each approach:

- Description of the main idea of the approach and its sub-approaches.
- Explanation of a sample of the cross-platform mobile development solutions (per approach/sub-approach) and their limitations.

4.1. Compilation approach

Compilation is one of the cross-platform mobile development approaches. It consists of two sub-approaches: cross-compiler and trans-compiler. The compiler is a program that transforms the source code written in the source language (high-level programming language) into the target language (lower-level language like the assembly language or machine code). The common use of the compiler is to transform the source code written in high-level programming language to an executable program. The compiler is called cross-compiler when the compiler runs on a computer with OS different from the one on which the compiled program will run. It is called trans-compiler when it transforms one high-level programming language to another high-level programming language.

4.1.1. Cross-compiler

The cross-compilers generate programs for multiple platforms. The cross-compiler runs on a platform other than the platforms on which the compiled programs will run. The next subsections explain in more details the following solutions: XMLVM and MoSync.

4.1.1.1. XMLVM [44]. XMLVM framework is a bytecode level cross-compiler. XMLVM cross-compiles an Android App to other platforms like iOS or Windows Phone 7 (WP7) [45]. Every Android App runs in a separate instance of the Dalvik VM, but the generated Apps from the XMLVM do not require the Dalvik virtual machine to run on the target platform. The byte-code transformation was chosen based on the following reasons: bytecodes are much easier to parse than Java source code, some high-level language features such as generics are already reduced to low-level bytecode instructions, and the Java compiler does extensive optimizations to produce efficient bytecodes.

Android App is cross-compiled to WP7 App as shown in Fig. 3. The steps of cross-compilation are as follows:

Table 7 Pros and cons of the cross-platform mobile development approaches.

Approach		Pros	Cons	Solutions
Compilation	• Cross-Compiler	• Reuse of the existing source code by cross-compilation to another application run on different platform • The produced applications are native, hence get the advantages of the native App	• The mapping between the source language and the target language is very difficult to achieve, so the cross-compiler supports a few platforms and focuses only on the common elements of these platforms. [6]	• MoSync [21] • Corona [22] • Neomades [23] • XMLVM [24]
	• Trans-Compiler	• Used to reuse the legacy applications by translating the legacy code to use the next version of the same programming language • Reuse of the existing source code by trans-compilation to another application run on different platform • The produced Apps are native, hence get the advantages of the native App	• Focuses only on the common APIs in both the source and the target programming languages • Needs regular updates to reflect the changes in the APIs of the source or the target languages	• [25] • J2ObjC [26] • JUniversal [27]
Component-Based		• Simplifies the support of new platforms by implementing the set of components with the defined interfaces for the new platform	• Focuses on the common functions among all supported platforms • The developer has to learn how to use the defined component interfaces	• [18] • [20]
Interpretation	• Web-Based	• Easy to learn and use as it depends on the web technologies	• The user interface of the web-based Apps does not have the native look and feel • Less performance of the produced applications than the native apps	• PhoneGap [28] • Rhomobile [29] • xFace [30]
	• Virtual Machine	• Smaller size of Apps and faster downloading times from the store because all the libraries and methods needed for the App to run are stored in the VM	• Slow execution of the application on the VM hence the VM is not used with Apps that need short response time • The VM needs to be downloaded from the App store which is not possible for the Apple's platform (iOS)	• MobDSL [31]
	▪ Runtime	• The source code is written once for the target platforms	• At runtime, the loading performance is lower, as interpreting the source code on the device needs to be done every time the application runs [11]	• Titanium [32] • Xamarin [33]
Modeling	• MD-UID	• Saves the development time by generating the UI code [34] • Useful in prototyping as it allows a rapid UI development to evaluate the usability of the Apps in many devices and platforms [34]	• Needs to focus on the similarity of user interface in different platforms [34] • Difficulty of maintenance of the generated UI for the different platforms. A possible solution is to allow a reverse engineering from the code to the model and keep changes when regenerating the UI from the updated model [34]	• XMobile [34]
	• MDD	• The language used for modeling is an effective tool to define requirements • Helps the developers to focus on the functions of the App instead of the technical implementation issues	• Does not support reuse of existing native source code [25]	• JSAF [35] • MD2 [36,37] • UsiXML [38] • Jelly [39] • MobiA modeler [40] • AppliDE [41]

(*continued on next page*)

Table 7 *(continued)*

Approach	Pros	Cons	Solutions
Cloud-Based	• Application processing is delegated to the cloud	• Requires high-speed network environment • The mobile device needs internet connection to run the application	• [42] • [17]
Merged	• Benefits from the strengths of each approach • Provides more features/facilities to the developer	• Requires a lot of efforts in development	• [6] • ICPMD [43]

(1) Android Apps written in Java are compiled normally into bytecode via a regular Java compiler, and fed into XMLVM tool.
(2) The bytecode is then transformed into an XML representation by converting the stack-based bytecode instructions to register-based instructions introduced by Dalvik.
(3) XSL stylesheets then transform the generated XML representation into code for the target language (i.e. C#) by mimicking the register machine in the target language.
(4) The generated project from XMLVM contains all cross-compiled source files and other resources (i.e. images) but no binaries.
(5) The Visual Studio is used to compile the generated App.

XMLVM relies on the Android API for application development. Instead of creating a new API for various functionalities, XMLVM makes use of the Android API which is mapped to the target platform. Fig. 4 shows the classic Android application, and the corresponding Android application used in the XMLVM layer model. While a classic Android application makes direct use of the underlying native platform, the mapping of an Android application to a different platform is divided into several different layers as follows:

- **The Android Compatibility Library** (ACL) (the highest level of the XMLVM layer model) offers the same API as the Android platform, but supports multiple underlying platforms through the use of Common Device API. The ACL contains the implementation of all parts which can be solved in a platform independent manner which includes APIs like layout manager, XML parsing, or the Android App lifecycle. If a part needs access to native functionality, it uses Common Device API to access it.
- **Native Adapter Libraries** (the middle level of the XMLVM layer model) implement the Common Device API and adapt differences between the Android's API and the API of the underlying platform. The Common Device API is exposing all platform dependent native functionality needed by the ACL. Examples of exposed native functionality are UI widgets (i.e. buttons or text fields) or sensor API. The adapter library hides platform specific API from the ACL and clearly separates the wrapper libraries from any Android related dependencies.
- **Wrapper libraries** (the lowest level of the XMLVM layer model) are representing the native APIs of the supported platforms in Java because the supported platforms use different programming languages than the Android platform. For example, the C# APIs of the WP7 platform (or the Objective-C APIs for the iOS platform) is represented as Java APIs, allowing Java programs to interact with native functionality of the target platform.

The limitations of this solution include the following:

- This tool does not offer the complete API mapping for the supported platforms because the APIs of the Android, iOS and WP7 operating systems are complex libraries that consist of thousands of methods.
- This tool cannot map some UI idioms (i.e. Android's hardware buttons).

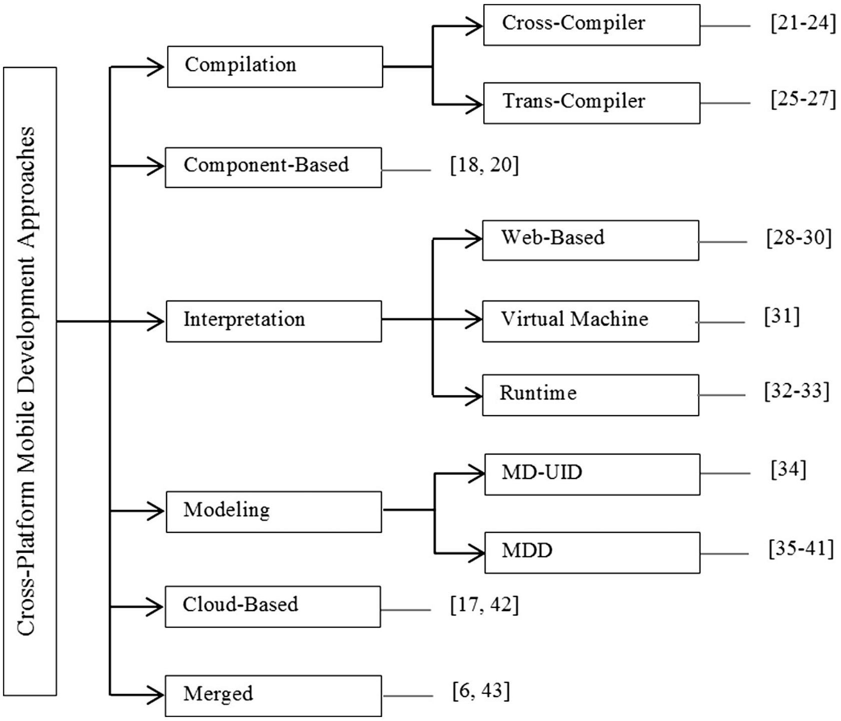


Figure 2 Cross-platform mobile development approaches.

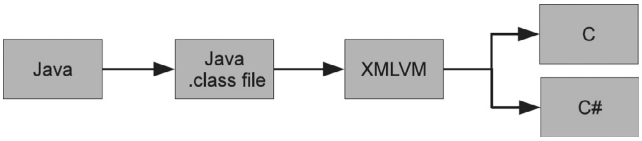


Figure 3 XMLVM framework cross-compiles an Android App to C# code for WP7 [44].

- This tool cannot cross-compile some functions because they are offered by the Android API and not available in iOS or WP7 (i.e. background services).

4.1.1.2. MoSync [21]. MoSync framework supports the development of cross-platform Apps using standard C/C++, web-based languages like HTML/HTML5/CSS/JavaScript, or a combination of both to create hybrid Apps. It provides an Eclipse-based IDE. This tool has many components tightly coupled together, including libraries, runtimes, device profile database and compilers [12]. Applications in MoSync are built to target a device profile by using the GNU Compiler Collection (GCC) and pipetool. After writing the App, the IDE uses the GCC to compile the code to MoSync intermediate language which is fed to the pipetool to link it with the App resources and MoSync libraries. To create the executable packages for each target device, the IDE combines the resulting bytecode or Java source code with the target platform runtime. MoSync runtimes are libraries and programs that execute the MoSync code on the target device. MoSync is an open source software and it is extensible in a way similar to Eclipse (i.e. adding plug-ins or external libraries) [46].

MoSync has two architectures: one for C/C++ language and the other for the web-based languages. Both architectures consist almost of the same components. Fig. 5 shows the architecture components, and these components are [47]:

(1) **Service layer module**: it supports many functions like file I/O, networking, and memory management.

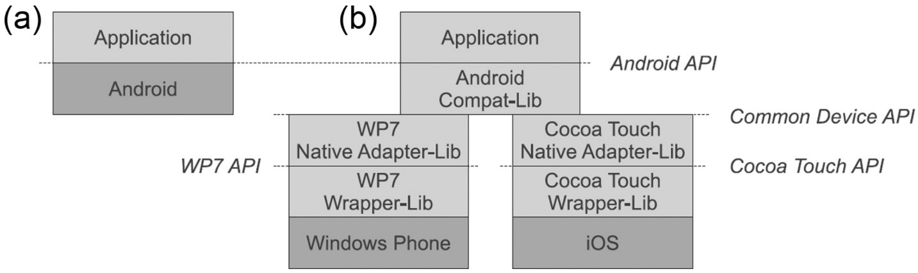


Figure 4 (a) Classic Android App (b) Android App in the XMLVM layer model [44].

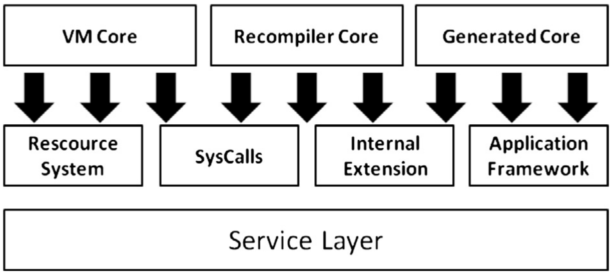


Figure 5 MoSync runtime architecture [12].

(2) **Application framework module** (holds the runtime entry point): it manages platform specific events like event management, initialization, and destruction.
(3) **Syscalls module**: it provides implementation for the mobile features which are essential for all platforms such as camera, contacts, images, audio, and networking. It interprets the MoSync resource files and provides event management, initialization, and destructions.
(4) **Resource System module**: it manages resources like images, sound, and data blobs. It manages dynamic creation and destruction of resources.
(5) **Internal Extensions module**: the platform-specific features (are not available in all platforms) are implemented as functions that are called through a single Syscall. This Syscall returns a specific error value if the called function is not available in the platform.
(6) **Core module**: it executes the MoSync programs by interoperating with Syscall and Resource System. There are three different types of core which all share the same common interface. These types are
    - **Virtual Machine Core**: this core is a virtual machine that loads, interprets, and executes MoSync bytecode directly. The execution is implemented in a single small function that allows efficient Just-In-Time (JIT) optimization. This core is used for Java Mobile Edition (Java ME).
    - **Recompiler Core**: this core loads MoSync bytecode and recompiles it into native code for the specified platform or typically Advanced RISC Machine (ARM) code. Then the generated code is executed. This core is used for Symbian and Windows Mobile.
    - **Generated Core**: this core neither interprets nor recompiles MoSync bytecode. It contains native code generated ahead of time from MoSync intermediate language. This core is used for iOS.

VM core is best for debugging and its dynamic approach allows to load new code at runtime. This property is very useful for many Apps. Recompiler core is more efficient but less debugging support and its dynamic behavior allows fast recompilation of some code. Generated core has zero overhead for low end devices which are not able to load code at runtime [47].

The limitations of this solution include the following:

- The developer needs to know the C/C+ + language or web technologies to use this tool.
- The MoSync IDE is characterized by a long start-up time and lacks a complete integration with other tools (e.g. a user interface designer) [48].
- The debugger is poor for the C+ + implementation, while it gives reasonable functionalities for the JavaScript and web-based development [48].
- The MoSync is not updated with the recent versions of the supported platforms. For example, iOS is supported till version 4.3, Android till version 4, Blackberry is in beta till version 6 and Windows Phone till version 7.5 [48].
- Some functions are not available through the APIs (e.g. Bluetooth for iOS and Google Maps for Android) [48].

4.1.2. Trans-compiler

The trans-compiler is a source-to-source compiler that takes as input a source code of a high-level programming language and the output is a source code of another high-level programming language. The next subsections explain in more details the following solutions: J2ObjC and an unnamed solution that was mentioned in a paper with title "Towards Platform Independence of Mobile Applications Metamorphosing Android Applications for the Web" [25].

4.1.2.1. J2ObjC [26]. This tool translates Java code to Objective-C code for the iOS platform (iPhone and iPad). It supports the transformation of general business logic as the transformation of the UI-specific code is not supported. The goal is to write the application without user interface in Java (the application logic) and then use this tool to be converted to iOS application (without user interface). J2ObjC supports most Java language and runtime features required by the client-side application developers, including exceptions, inner and anonymous classes, generic types, threads, and reflection. In addition, the JUnit test translation and execution is supported. The J2ObjC is an open-source command-line tool from Google, which is currently between alpha and beta quality.

The limitations of this solution include the following:

- The transformed source code needs some refinement.
- There is no support to transform the user interface code.

4.1.2.2. Towards Platform Independence of Mobile Applications Metamorphosing Android Applications for the Web [25]. This tool reuses the source code of Android Apps by converting them to platform independent web Apps using the Google Web Toolkit (GWT). This conversion process is illustrated in Fig. 6. The Android has its own custom Java library and the target system GWT also has its own custom library (subset of the Java runtime library). The main task of the Android2GWTConverter is to handle the differences in these two Java libraries. After that, the GWT compiler translates the Java source for the client side to JavaScript code and for the server side to Java bytecode.

The Android2GWTConverter has to identify all statements in the source code which are not supported in the target system. The Android source code is parsed by the converter to produce Abstract Syntax Tree (AST) which is going to be modified based on the following tasks of the converter:

- **Handle unsupported library calls**: the unsupported library calls in the Android source code are modified to fit the GWT target library (For example, the method

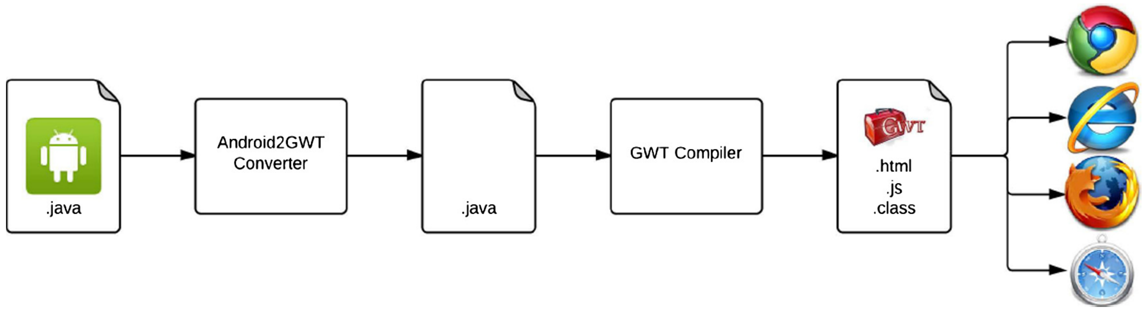


Figure 6 The steps of converting the Android App to the web App. [25].

System.nanoTime() is not available in the GWT custom library, so this missing method can be replaced by System.currentTimeMillis*1E6).

- **Resolve multithreading code**: the JavaScript is single threaded; hence the parts of the multithreading code have to be modified. The solution is to shift the work to the server and then execute it via a Remote Procedure Call (RPC), or to use the new HTML5 feature Web Workers. The first option has been chosen in this tool because the Web Workers feature is not supported yet by all browsers.
- **Provide Android wrapper classes**: the complexity of the converter is reduced due to the strength of the GWT's UI elements and other functions of the available libraries. Android wrapper classes provide the API calls which access internally GWT code or directly HTML5 features.
- **Manage Android resources**: the Android resources and their references from the code have to be modified. All the used resources are copied to the resource folder of the target project to be accessible on the GWT server.

The limitations of this solution include the following:

- The converter does not handle yet the UI layouts which are defined as XML files in the Android Apps.
- The converter ignores the features in the Android source system which are neither supported by GWT nor by HTML5.

4.2. Component-Based approach

Component-Based is one of the cross-platform mobile development approaches. The software component is a package or a module that contains a set of related functions or data. The system functions can be divided into several separate components considering that the related functions are grouped together in the same component. Each component has an interface that specifies the services that can be used by other components. Components communicate with each other via interfaces. Therefore, there is no need to know the inner implementation of the component to use it. The Component-Based approach divides the system functions into a set of separate components with defined interfaces. Each component has the same interface for all platforms, but different inner implementations based on each supported platform. The next subsections explain in more details the following solutions: an unnamed solution that was mentioned in a paper with title "Component-Based Mobile Web Application of Cross-Platform" [18] and another unnamed solution that was mentioned in a paper with title "Generic Framework for Mobile Application Development" [20].

4.2.1. Component-Based Mobile Web Application of Cross-Platform [18]

This tool simplifies the development of Apps for different platforms using the concept of software components by packaging the main features of mobile development into set of separate modules including storage management, networking, graphics, file system, and system service components. These components could be reused in the development and Apps developed based on these components can be migrated to other platforms easily.

Fig. 7 shows the framework of this tool. The framework consists of four layers as follows:

(1) **Application Layer**: this layer is divided into several modules which depends on the requirements of the App. These modules can be developed by different teams. The App is developed using the web technologies: HTML, XML, AJAX, and CSS.

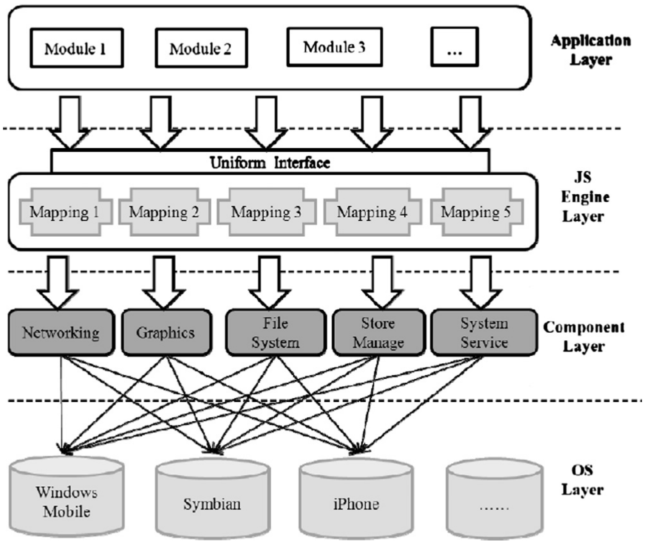


Figure 7 Framework of component-based hierarchical mobile application development [18].

(2) **JS Engine Layer**: this layer parses the JavaScript code of the application layer to map the methods and properties of script with matched interface of the component layer to implement specific function by these component interfaces. For example, when the interface of CreateNewFolder is called by JavaScript language in the application layer, JS engine will map this method to the matched interface of the component.

(3) **Component Layer**: this layer masks the differences of the underlying platforms by supporting the application layer with unified interfaces through the JS engine layer. The different components include
- **Networking**: includes network connectivity and the functions of "disconnect network", "select mode of connections", and "diagnosis of network fault".
- **Graphic**: deals with the output of graphics and images to convey information between system and drawing program.
- **File System**: includes the functions to create new file, delete file, modify file, and find file.
- **Store Management**: includes the memory allocation and destruction.
- **System Service**: includes the functions to call local services like dial phone, send a message, and open camera.

All the five components are implemented separately for each platform. An implementation for certain OS refers to the supported APIs of this OS. These components can be compiled to library (lib) or dynamic link library (dll) file. To develop a mobile App that runs on Symbian OS, just include the static or dynamic library files (libs or dlls) that have been compiled on Symbian OS without modifying the code of the application layer.

(4) **OS Layer**: this layer includes a set of different mobile platforms like Windows Mobile, Symbian, iOS, Palm, and Android.

The limitations of this solution include the following:

- It is still a proposal for a component-based hierarchical development framework.
- It focuses only on the common functions among the target platforms.

4.2.2. Generic Framework for Mobile Application Development [20]

This framework enables the mobile application developer to develop the App just once and deploy it on three mobile platforms: Android, BlackBerry and iPhone as shown in Fig. 8. The JavaScript engine is the common artifact that exists across the various mobile platforms; hence, this common artifact is utilized in constructing this framework. The application developer develops the App in HTML, JavaScript and CSS. The developer uses the framework features/components in the business logic of the App by utilizing the available APIs in the framework which connects to the respective framework API interfaces, including Android interface 2.0, BlackBerry interface 3.0, and iPhone interface 4.0. These API interfaces call the device specific native APIs of Android 2.1, BlackBerry 3.1, and iPhone 4.1 respectively. The result is returned in the reverse sequence to the business logic, hence making it possible to access the device features across the three platforms with the same business logic.

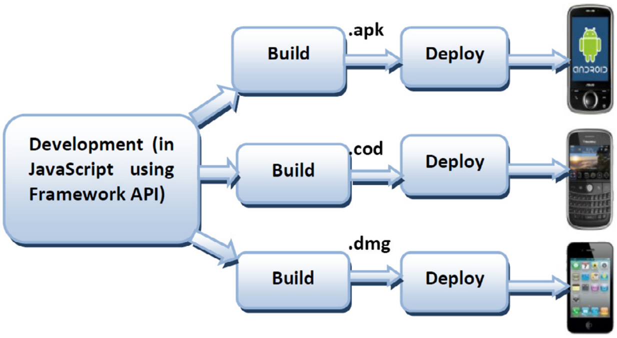


Figure 8 Application development using the framework [20].

The framework provides the following features:

- **Graph**: the framework provides user interface components such as various types of graphs like bar, line, pie, stacked side bar, side bar, and mini pie.
- **Accelerometer**: the accelerometer sensor in mobiles is used to measure vertical and horizontal positioning of the phone. The framework allows to register, unregister the accelerometer sensor of the phone, and get the acceleration of the device whenever it changes.
- **Camera**: The framework provides functions to capture a picture from the mobile camera.
- **Contacts**: The framework allows to add a contact along with his/her information to the address book of the device.
- **Location (Global Positioning System, GPS)**: the framework allows to register, unregister the GPS of the device, and get the latitude and longitude of the device.
- **Mail**: the framework allows to invoke the native mail client of the device from the developed App.
- **Orientation**: the Orientation sensor measures the vertical and horizontal positioning of the device and returns azimuth, pitch and roll. The framework allows to register, unregister the orientation sensor of the phone, and get orientation of the device whenever it changes.
- **SMS**: the framework allows to include the SMS feature in the App.
- **File Input/Output**: the framework allows to create, read, write, copy, rename and delete the files and directories on device storage and SD card. It also allows to retrieve information like: last modified time and size of the file.
- **SQLite**: SQLite is a software library that implements a local (serverless) transactional SQL database engine. SQLite stores the entire database (tables, definitions, and data) as a single cross-platform file on both device storage and the SD card. The framework provides functions to open, create, query and delete the database.
- **Device Information**: the framework allows to get information about the device like operating system, version of the OS, Universally Unique Identifier (UUID) and International Mobile Equipment Identity (IMEI).

The limitations of this solution include the following:

- It is still a proposal for a component-based hierarchical development framework.
- It focuses on the common functions among the target platforms.

4.3. Interpretation approach

Interpretation is one of the cross-platform mobile development approaches. It consists of three sub-approaches: Virtual Machine (VM), Web-Based, and Runtime Interpretation. The interpreter translates a source code to executable instructions in real time with a dedicated engine. The developers develop the mobile application once and the interpreter manages their execution on many platforms.

4.3.1. Virtual machine

The most common virtual machine is the Java Virtual Machine (JVM) which is a virtual computer simulating various computer functions. JVM has its own complete hardware architecture such as CPU, stack, and register [18]. In addition, it has a corresponding instruction system [18]. The main idea of this approach is to develop the mobile App with a platform-independent language. Then, the developed App runs on the dedicated virtual machine which is installed on different platforms. The next subsection explains in more details the MobDSL language.

4.3.1.1. MobDSL [31]. The mobile Apps can be developed using a new defined language dedicated to the mobile domain (MobDSL) and these applications can run on a VM that support multi-platforms. A domain analysis was done to define the set of features to be supported in the MobDSL language. These features include the following:

- **Screen Size**: mobiles support only a limited size display which leads to a relatively small number of GUI features; therefore there is more scope for building these features into a common language. The resolution of mobile screen varies by hardware vendor. The UI design on the tablet is different than the phone which requires developers to create Apps with interfaces to suit each device type.
- **Layout Control**: Android uses XML files to control layout and support different layout styles including Linear, Relative and Absolute (deprecated). iOS supports programmatic layout and prebuilt XML type interfaces using Interface Builder (help the developer to create UIs easily but these layouts are less dynamic than the programmatic ones).
- **GUI Element Containership**: both iOS and Android platforms use a form of GUI element containership. In iOS development, the emphasis is on the application Window and its Views with Subviews. These elements are used to create any user interface (from simple to complex). A similar model is used in Android, except with Views and ViewGroups. Interface control on both platforms has similarities and differences. On iOS, Views are controlled by the View Controllers (implementation of the event handlers). On the other hand, Android development uses Intents and Activities.
- **Event Driven Applications**: the target Apps are event driven. In implementation, registering methods for event handlers is done dynamically, not statically. This method means there is a lack of checking at compile time to prevent an application crashing.
- **Hardware Features**: mobile devices have many features like camera, sensors, microphones, accelerometers, and GPS. These features tend to be fairly standard in their behavior if they are supported by the platform. The use of these features is done using the platform-specific framework.
- **Concurrency**: the use of concurrency in mobile Apps is paramount. This is carried out by using threads, for example: a UI thread starts with the execution of an iOS or Android App. This thread is used for the UI elements of the App, so heavy or concurrent tasks should run in its own thread. This can help to avoid the UI halts and a 'laggy' experience for the user.
- **Object-Orientation**: mobile Apps are developed using Object-Oriented (OO) languages. In iOS, the main language is Objective-C. It supports C+ + and the non-OO C as well. In Android, Java is used but with different libraries and uses the Dalvik VM instead of the Java VM because its characteristics support mobile devices more.
- **Transitional Behavior**: state machine transitional behavior is very common in mobile Apps, and can be found on the Android platform. Each Activity can be viewed as a state machine that stores the state and actions done by the user, and then causes transitions between different views or activities.

The MobDSL language is not designed for all types of mobile Apps. The language design focuses on supporting Apps that do not require extensive user-interaction and are driven by data stored in an external server. The basic calculus for the MobDSL language (based on the λ-calculus) is shown in Fig. 9.

Fig. 10 shows the architecture that uses the MobDSL on multiple platforms. It consists of three tiers:

(1) The application, written and compiled using the DSL.
(2) The DSL specific engine and implemented libraries.
(3) The running platform: Android or iOS. For each of the target platforms, the virtual machine will consist of two major parts: the platform libraries (MobLib) which will contain the specific platform API calls, and the engine (MobEng) that will run the compiled code and make the appropriate platform calls using the bundled platform library set.

The limitations of this solution include the following:

$$
\begin{array}{lcll}
E & ::= & & \text{expressions} \\
 & & V & \text{variables} \\
 & | & \textbf{let } V = E \textit{ in } E & \text{local defs} \\
 & | & \textbf{letrec } V = E \textit{ in } E & \text{recursive defs} \\
 & | & \textbf{fun}(V...V).E \mid E(E,...E) & \text{funs, apps} \\
 & | & \{V = E; \ldots; V = E\} \mid E.V & \text{records, ref} \\
 & | & [E, \ldots, E] & \text{lists} \\
 & | & \textbf{widget}(V)EE & \text{widgets} \\
 & | & \textbf{if } E \textbf{ then } E \textbf{ else } E & \text{choice}
\end{array}
$$

Figure 9 Calculus definition of the MobDSL language [31].

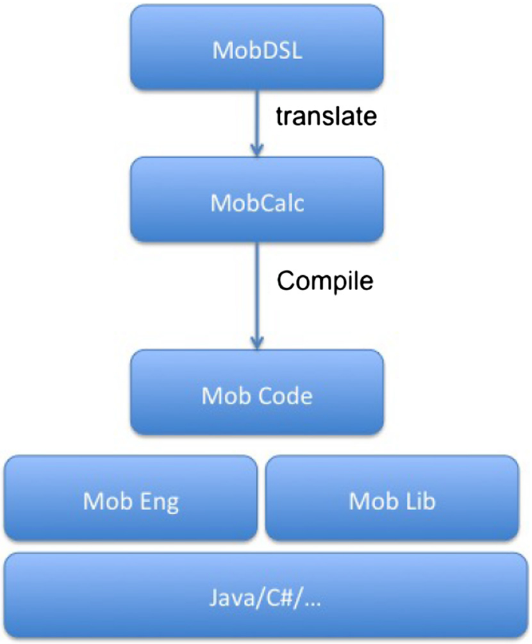


Figure 10 System architecture [31].

- The calculus language has been prototyped as an interpreter in Java, but the mobile VM for platforms like Android and iOS is not yet implemented [31].

4.3.2. Web-Based

The Web-Based tools mainly use the web technologies like HTML, HTML5, Javascript, and CSS to develop mobile Apps that can run on different platforms. To allow access to the device features like Camera, and Sensors, wrappers are implemented to access these native APIs. The next subsections explain in more details the following solutions: PhoneGap [12] and xFace [30].

4.3.2.1. PhoneGap [12]. PhoneGap is a “wrapper” that allows to load web Apps into native Apps and allows them to access the device functionalities. PhoneGap Apps are hybrid, which means that they are not purely native or web-based. The meaning of “not purely native” comes from the web UI which is rendered via webview instead of the native language of the OS, whereas “not purely web-based” comes from the lack support of HTML in some functions. Besides, PhoneGap also offers the possibility to extend the tool by developing custom plug-ins. The developers have to write a single source code for any mobile OS supported by the tool. PhoneGap does not provide a unique IDE to develop Apps on all mobile platforms, so the developers have to write the source code with an IDE and port it on other IDEs (e.g. Eclipse for Android or XCode for iOS). As an alternative, it provides a service called PhoneGap Build that allows developers to compile their PhoneGap Apps in the cloud. The developers upload their Apps written in HTML, CSS, and JavaScript and get back app-store ready application binaries. With this service there is no need to install mobile platform SDKs to build native Apps [11]. PhoneGap supports the most common mobile platforms: Android, iOS, Blackberry and Windows Phone. The most complete APIs support is provided for Android and iOS [48].

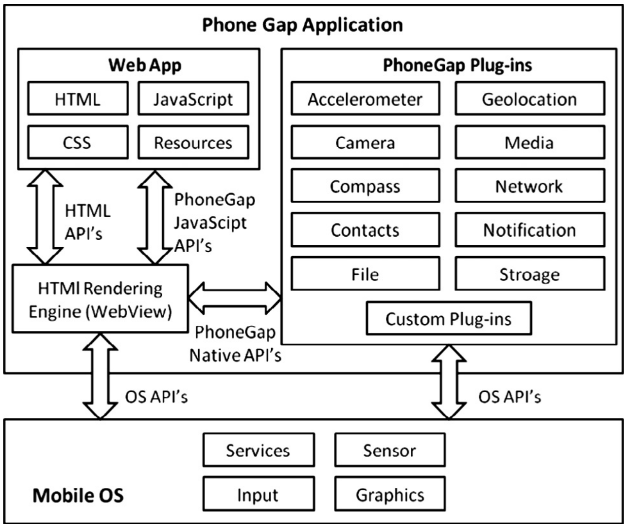


Figure 11 PhoneGap architecture and interfacing among components [12].

Fig. 11 shows the PhoneGap architecture which consists of the following three layers:

(1) **Web Application**: represents the application source code.
(2) **PhoneGap**: is composed by JavaScript and native API’s. This layer is responsible for the interfacing between web App and PhoneGap layers. Also, it takes care of the interfacing between JavaScript API’s that are used by the App with native API’s that are used by mobile OS’s. The functionality of this layer is to maintain the relationship between JavaScript API’s and native API’s of each mobile OS. PhoneGap provides JavaScript API’s to developers that allow the access to advanced device functionality such as Accelerometer, Barcode, Bluetooth, Calendar, Camera, Compass, Connection, Contacts, File, GPS, Menu, and Near Field Communication (NFC).
(3) Mobile OS and native API’s.

The limitations of this solution include:

- It does not provide the possibility to have native user interface [48].
- The performance of the final application is far from the native application [48].

4.3.2.2. xFace [30]. xFace is a platform for developers to write widgets (light-weight mobile web Apps) once and run it on multiple platforms using the web technologies such as HTML, CSS, and JavaScript. xFace has the following characteristics [30]:

- **Open standard:** xFace drafts out an open standard for lightweight mobile Apps. This standard is based on W3C’s Web Design and Apps standards (XHTML, CSS, AJAX, etc.) and Joint Innovation Lab (JIL) Widget API standard.

- **Cross-platform:** xFace supports different platforms such as Windows mobile 5.0/6.0/6.5, Symbian S60 v3/v5, MTK 6225/6235, Brew 3.1.5, Android 2.1, and Intel Moblin 2.0.
- **Lightweight:** xFace is a lightweight engine because it cuts off some tags or attributes in W3C's standards. xFace supports only 27 tags of HTML while the W3C XHTML1.0 standard supports 74 tags. The unsupported tags are not important because they can be replaced by other tags and JavaScript functions, or they have no use in mobile Apps.
- **Phone services API:** xFace allows the developer to access the mobile phone features by using JavaScript APIs.
- **Intellectual property right (IPR) protection:** the web languages are interpretive execution languages. The web Apps are not compiled before they are distributed to mobile phones. Therefore, people can easily get the source codes. To protect the developers' IPR, xFace develops a tool to encrypt the source codes before distribution. After distribution of the source codes, the xFace engine decrypts them and then executes the App.
- **Emulator:** the xFace emulator is a tool on PC platform for developers to test and preview their web Apps just as they would be run on the actual hardware of the phone.

Fig. 12 shows the architecture of the xFace. The architecture consists of three components as follows:

(1) **Porting Layer:** it is an adapter between xFace core and mobile platforms. It abstracts the capabilities of the mobile phone into several categories of interfaces such as networks, file system, GUI, timer, multimedia, and memory management.

(2) **xFace Core:** it is the kernel of xFace and performs most xFace's functions in the following modules
   - **XHTML Parser:** parses the ".html" file into an XML Document Object Model (DOM) tree, and then transforms it into an XHTML DOM tree with its specific attributes.
   - **CSS Parser:** parses the ".css" file or a style element in html file into a Style Sheet structure, and then applies the styles in the structure to each element in the DOM tree.
   - **JavaScript Engine:** xFace uses the open source JavaScript engine "SpiderMonkey" to execute the JavaScript codes in ".js" files.
   - **Render:** it layouts every element in the tree based on their attributes and styles, puts them into the right place, displays them on the screen, deals with the images and animations, and responds to the user's operations. As it is the direct interface to users, a lot of optimizations is done to enhance the performance of this module to offer a better user experience.
   - **Resource Manager:** it provides a uniform interface of all the resources xFace core could access. Generally, the resources come from the local file system and the http/https web protocol.
   - **AJAX:** it allows retrieving data from the server asynchronously. This is done in the background without affecting the behavior and display of the current page. xFace implements the W3C XMLHttpRequest level2 standard which enhances the XMLHttpRequest object with new features, such as cross-origin requests, progress events, and the handling of byte streams for both sending and receiving.

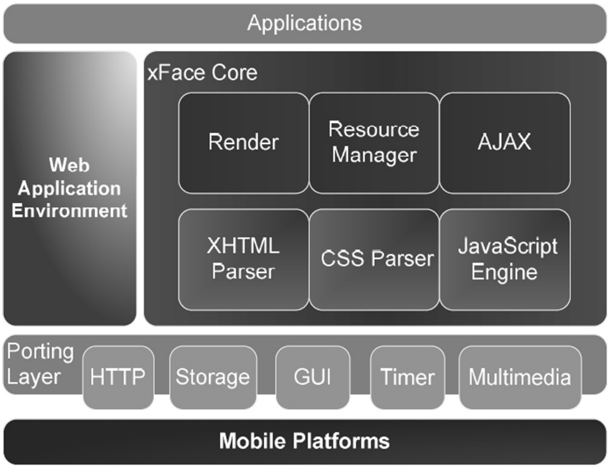


Figure 12 The architecture of xFace [30].

(3) **Web Application Environment:** it is the runtime environment for users to install, run, update or uninstall a mobile web App package after downloading it from the store. It is also connected to a billing system because some of the web Apps may not be free. xFace App can be compressed into a package then it could be installed on the phone and run locally.

To port xFace onto a new platform, there is no need to modify any line of xFace core, just write an implementation of Porting Layer interfaces.The limitations of this solution include the following:

- It does not support the native mobile applications. It is limited to mobile web applications only.

4.3.3. *Runtime Interpretation*

Runtime is an execution environment and a layer that makes the mobile App runs in the native platform. This approach translates the code to bytecode and then, at runtime, that bytecode is executed by a virtual machine supported by the mobile device. Rhodes and Titanium belong to this category [11]. The Titanium solution will be explained in more details in the following subsection.

4.3.3.1. Titanium. This tool allows the developers to write mobile Apps using a scripting language (JavaScript) as a programming language to implement cross-platform Apps. The scripting languages are augmented with mobile-specific APIs such as GUI development and location-awareness. Using Titanium, the developer's code is combined with an application skeleton for the target platform, and the output App runs the developer's code using a platform-specific engine (WebKit for iOS devices, Rhino for Android systems). Whenever the developer's code invokes a mobile-specific API, these calls are transformed into calls to APIs developed and provided by Titanium vendor [5].

Titanium does not use HTML to create the user interface. Instead, the UI is implemented completely programmatically. Developers use JavaScript to build the interface and to implement logic and data, extensively using the Titanium API. The code is then packaged with Titanium's engine. At runtime, this engine interprets the JavaScript code and creates the user interface. However, their look-and-feel resembles the typical platform appearance more closely; the UI is made up of native elements [15].

In contrast to PhoneGap, Titanium UIs are using real native UI components rather than web views. Like PhoneGap, Titanium requires the SDKs of the target platforms to be installed and set up within the SDK. It uses the Apache License (version 2.0), but the software is proprietary. The basic version of Titanium is free. There is also a Professional Edition and an Enterprise Edition. These offer additional support, some additional modules and early access to new features [8]. The platforms supported by this framework are iOS from 5.01, Android from 2.3.32, Blackberry, Tizen and Web Mobile, but the APIs almost cover Android and iOS [48].

Titanium links JavaScript to native libraries, compiles it to bytecode and then the platform SDK (Android or iOS) builds the package for the target platform. Also, the output App contains a JavaScript interpreter runtime and a Webkit rendering engine [11].

The limitations of this solution include the following:

- Although Titanium supports different platforms, the negative aspect of this framework relies on the support essentially limited on iOS and Android [48].
- At runtime, the loading performance is lower, as interpreting the source code on the device needs to be done every time the application runs [11].

4.4. Modeling Approach

Modeling is one of the cross-platform mobile development approaches. It consists of two sub-approaches: Model-Based User Interface Development (MB-UID) and Model-Driven Development (MDD). The developers use abstract models to describe the functions and/or the user interface of the applications. Then these models are transformed to source code for different target platforms.

4.4.1. Model-Based User Interface Development (MB-UID)

MB-UID approach is used to generate the UI automatically by formally describing tasks, data, and users for an App, and then use these formal models to guide the design generation. It separates between the interface description and the App logic. There are two strategies for the UI code generation [34]:

- **UI generation at execution time** of the App: it is used for adaptation of web systems based on request/response. The constraint is that the App must remain connected to the server all the time.
- **UI generation at development time** before the execution of the App: the developer can verify the generated concrete interface, and adds specific functionalities to each generated version. Besides, it allows the construction of Apps with different connectivity modes (i.e. always connected, disconnected with later synchronization).

The next subsection explains in more details the XMobile solution.

4.4.1.1. XMobile [34]. This tool semi-automatically generates, at development time, the user interface for providing fast development of multi-platform and adaptive Apps using the MB-UID approach. XMobile is a framework that generates device-independent user interfaces. It can be used to describe the interface in a more detailed way to generate richer interfaces. It does not limit the types of Apps that can be built with the environment.

The XMobile environment as shown in Fig. 13 is composed of the XFormUI framework and the Use Interface Generator (UIG) tool. The framework XFormUI is a toolkit for creating graphic interfaces that are independent of a particular device and from a certain platform. The developer describes the navigation graph associated with the App. Then, the UIG tool uses this initial description to generate the final code for the different supported platforms.

The developer uses the XMobile to:

(1) Describe the App forms using the following standards: CSS to define the style (i.e. colors and layout), XForms to define the components, and XML Schema to define the constraints to the data input fields of the form.
(2) Create the manifest.xml file using an Eclipse plug-in. This file defines the navigation among the interface elements and the mapping rules. They are then converted to an executable code of a specific platform. The Eclipse plug-in uses the UIG tool to generate the code of the interface. This generated code uses the framework XFormUI to compose the form interface. The XFormUI was designed to have components with the same behavior and name as the XForms components.

This approach splits the user interface description into five levels: Abstract Interface, Presentation, Data and Validation, Dialog and Concrete Interface. The developer describes the first four models and the UIG tool generates the last model. Fig. 14 shows the XMobile user interface models.

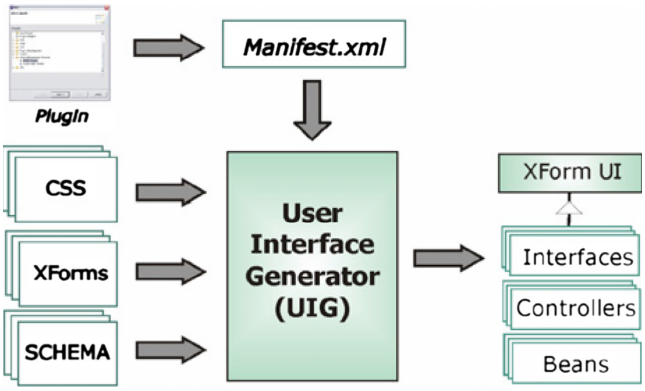


Figure 13 XMobile environment [34].

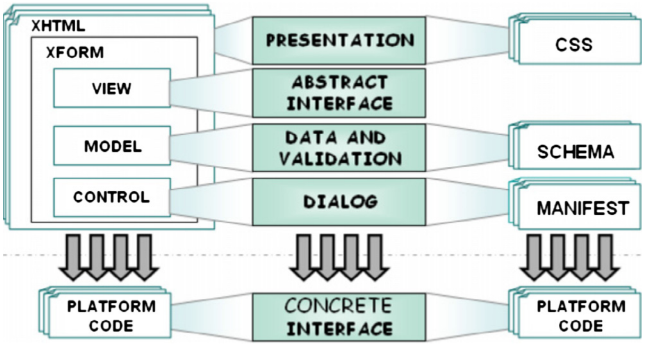


Figure 14 XMobile user interface models [34].

The limitations of this solution include the following:

- It focuses on generating the user interface only and does not support the source code generation.

4.4.2. Model-Driven Development (MDD)

The main concept of MDD is to generate platform specific versions of the App out of a platform independent model. The abstract model of the App is described by the Unified Modeling Language (UML) or DSL. The next subsections explains in more details the following solutions: *JavaScript Application Framework* (JSAF) [35] and MD2.

4.4.2.1. JavaScript Application Framework (JSAF) [35]. This tool allows to develop mobile web Apps using web technologies such as JavaScript, HTML, HTML5, and CSS3. This framework is defined following Platform Independent Model (PIM) in MDD to achieve the support of developing a single mobile App that runs on various mobile platforms. The JSAF App is rendered on a browser which will load JSAF framework prior to the application. When an App is launched, a new DOM page is created and the App's base document (index.html) is loaded and this document must request loading of "JSAF.js" as follows:

<script type=”text/javascript” src=”/JSAF/framework/scripts/JSAF.js”>

After the loading, the execution flow of framework takes control. It initializes and loads all the framework files and application sources.

There is a wide range of widget APIs available in JSAF which can be used directly in the developer's application once the JSAF is loaded in the browser of the mobile; the widgets are also loaded into the browser. The widget has to be instantiated to be displayed on the mobile screen.

Fig. 15 shows the layered view of the architecture. This framework is based on the MVC application model; the JSAF Apps are structured and packaged in MVC model as follows:

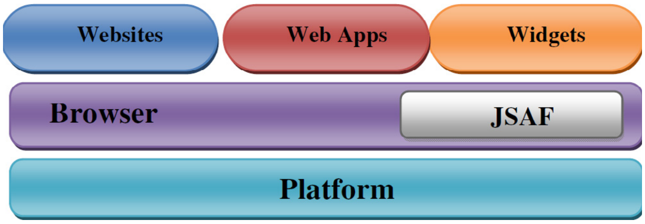


Figure 15 Layered view of the architecture [35].

- **View** consists of UI descriptions which are in HTML and CSS.
- **Controller** is responsible for initialization of setup, initialization of model and handling user actions. Controllers are called Assistants in the App.
- **Model** consists of the application data.

The JSAF application model is based on pages and views. A page corresponds to one window and a view corresponds to the content displayed inside the page. A page can have multiple views to be displayed. An application can have multiple pages in it. At a time, a page can have only one 'active' view. An application can have only one 'active' page at a time. An application can have multiple entry points, and multiple pages can be useful to provide multiple entry points to an application. JSAF provides different features for the App development as follows:

(1) **Single Page Multiple View**: a single page in which multiple views are displayed one over the other just like a stack. Each view has a separate View Assistant and each page has a separate Page Assistant.
(2) **Multiple Page Multiple View**: multiple single pages and each page has multiple views. There are multiple Page Assistants for each page and multiple View Assistants for each view.
(3) **Cross Application Communication**: it supports linking and establishing a communication between two applications, the present application App1 is in a page with view1 can access a different application App2 with different view view2 directly. This reduces the complexity for the user who has to access these Apps separately.

The limitations of this solution include the following:

- Does not support native mobile applications.

4.4.2.2. MD2 [36,37]. MD2 framework is based on a new DSL which is tailored to the domain of mobile Apps. This tool allows developing Apps by describing the application model using the new defined DSL, and then a set of transformation steps are done to generate native and platform-specific source code. The MD2 new language and the Apps created with the framework follow the MVC pattern. The new DSL of MD2 considers the following needs of developers:

- Define data types, and access the Create, Retrieve, Update, and Delete operations (CRUD) for these types, locally on the device as well as on a server.
- Implement the UI with different layouts, especially UIs with tabular views (tabs), and with a variety of typical UI components.
- Control the sequence of UI views.
- Define data bindings and input validation.
- React to events and state changes and use device features such as GPS.

Fig. 16 shows the workflow and architecture of MD2. Three phases for developing Apps with MD2 are as follows:

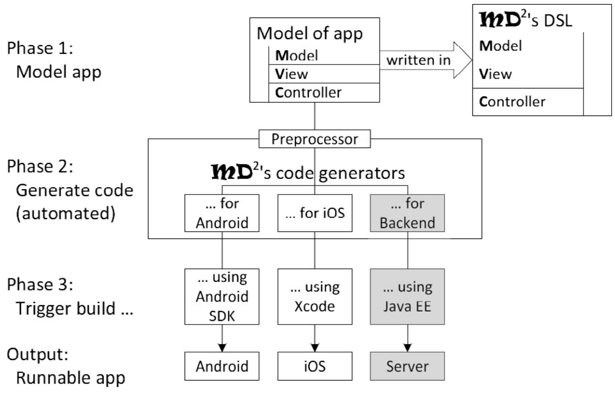


Figure 16 Workflow and architecture of MD2 [37].

(1) The developer describes the App in a textual model using the new defined DSL for MD2. The abstract syntax and the textual notation of the language have been defined and implemented using Xtext.
(2) A code generator for each of the supported platforms transforms the model into source code (Java for Android and Objective-C for iOS) that is suitable to the corresponding platform. It generates also the necessary structural elements such as project files. The code generator creates some XML files (for example, the GUI in Android and the data model in iOS). Moreover, the project files and settings for Eclipse (Android) and Xcode (iOS) are generated. The generated Apps are bundled with (static) libraries for common features. In addition to Apps, a further code generator creates a server backend based on the data model of the application. The backend can be run on a Java Enterprise Edition (JavaEE) application server.
(3) The developer compiles the generated source code using the corresponding development environments such as the Android Developer Tools for Android or Xcode for iOS. The generated Apps have a native look and feel, can access the device hardware functions, and communicate with remote servers.

The limitations of this solution include the following:

- It is still a prototype framework.
- It focuses on one category of mobile applications: the data-driven business mobile applications.
- It focuses on generating mobile applications that do not support the reuse of existing source code.
- It focuses on code generation for tablets.

4.5. Cloud-Based approach

Cloud-Based is one of the cross-platform mobile development approaches. In this approach, the application processing is done in a cloud server instead of running the application locally. Therefore, it uses most of the cloud features, including flexibility, virtualization, security, and dynamic management. The client side can use the mobile thin device because only basic processing is required at the terminal. In the cloud environment, thin client devices are lightweight and potentially energy efficient [42]. The next subsections explain in more details the following solutions: a solution that was mentioned in a paper with title "*Multi-Platform Mobile Thin Client Architecture in Cloud Environment*" and another solution that was mentioned in a paper with title "*A Cross-platform Application Development Environment Supported by Cloud Service*".

4.5.1. Multi-Platform Mobile Thin Client Architecture in Cloud Environment [42]

This solution implements multi-platform mobile thin-client architecture in the cloud environment. The thin-client mobile App runs on the user's mobile device and the server application runs on the cloud. The client and server communicate across a wireless connection between the mobile and the server. The client sends input data across the network to the server, and the server returns the display updates. The existing thin-client systems are divided into two categories (based on how the display information is represented):

(1) **The first category** utilizes high-level commands to represent the screen updates (i.e. the Citrix Metaframe and Microsoft Remote Desktop Protocol (RDP)). The graphics commands are transferred from the server to the client, which is responsible for handling the updates. The interpretation of high-level commands mainly depends on the OS's so it is hard to develop the servers and clients on different OS's with different rendering mechanisms.
(2) **The second category** utilizes low-level approaches to represent the screen of remote servers (i.e. Virtual Network Computing (VNC) and Sun Ray systems). They process updates to the display on the server and transfer only compressed pixel data that represent the new display to the client. The most important is that they are platform-independent.

Fig. 17 shows the architecture of the multi-platform thin client system, leveraged with the cloud environment and VNC protocol. The architecture consists of five components as follows:

(1) **Mobile Terminal**: a mobile thin client is connected wirelessly to the cloud infrastructure where the multi-platform servers are deployed.
(2) **Authentication and Management System (AMS):**
   - At first, the mobile terminal sends a request to AMS, the request includes:
     – The user's information, like terminal number and password.
     – Terminal information, like the type of terminal device.
     – Application information, like name of the App, platform of the App, the App authorization number, and item number of the App.
   - If this is the first time to login in, the AMS uses the request information to deploy the virtual machine. If not, the AMS uses the information stored in the Personal Storage Cloud.
   - When the connection is linked, the VNC server deployed in the Mobile Emulator will send

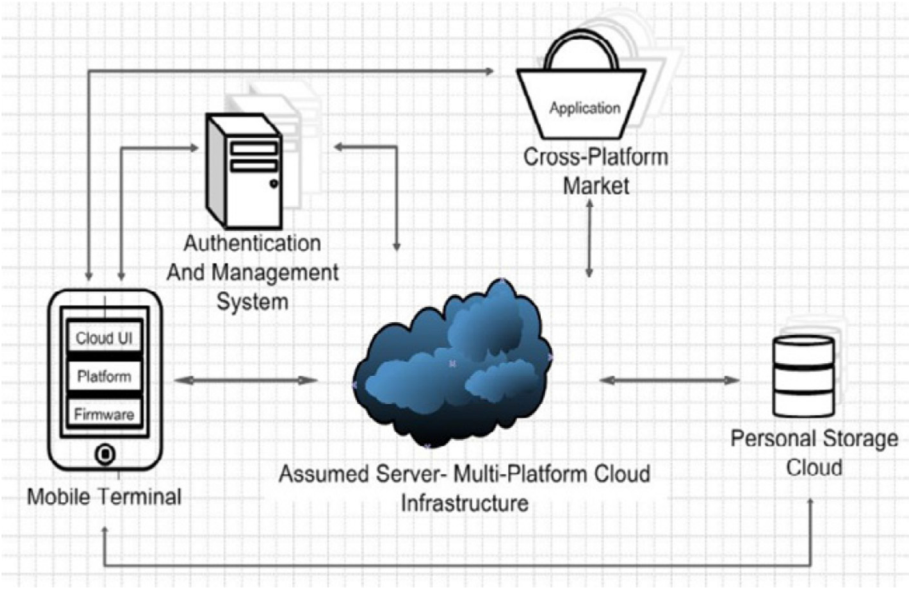


Figure 17 The architecture of multi-platform thin-client system [42].

continuously the frame buffer of display updates to the mobile terminal.

(3) **Cross-Platform Market**: provides Apps that belong to different platforms to sell. When a user buys an App, the application is already installed in the Cloud Infrastructure (CI), the CI just adds the user item number to the authorization list of the App.
(4) **Personal Storage Cloud**: stores the application data and the users' data.
(5) **Multi-Platform Cloud Infrastructure**: different mobile emulators are deployed in the cloud including Android emulator, iPhone emulator, Windows Mobile 7 emulator, and others. Therefore, all kinds of Apps can run on the Cloud Infrastructure. In every virtual machine, a VNC server is deployed to capture the updates of Emulator and then sends them to the mobile terminal.

The limitations of this solution include the following:

- The remote computing or virtual desktop was tested in LAN or high-speed network environment; hence, this solution needs improvements to make it better in lower speed network environments.

4.5.2. A Cross-platform Application Development Environment Supported by Cloud Service [17]

The main target of this solution is to provide a cross-platform application development environment which is supported by cloud service. This development environment can generate the following: (1) applications for different platforms which run on different devices, and (2) service applications which run on the cloud and offer specific services according to the terminal device.

Fig. 18 shows the system architecture which is divided into three parts:

(1) **Mobile Terminal Device Application Development Environment (MTDADE):** provides two functions:
  - Support for developing cloud service application which has the ability to integrate and aggregate data in the cloud. The service application runs at the cloud server and provides version management, service update, service publication to the mobile terminal device. The cloud service supports many platforms such as Android, embedded Linux, Windows Phone7, and iOS. Also, it can provide service to various terminal devices such as smartphone, pad, and Internet TV.
  - Support for developing mobile applications, such as iOS and Android Apps, and publishing the developed Apps in the App market. The mobile terminal device can get the corresponding Apps from the market and achieve service provided by cloud.

(2) **App Market and Cloud Server**: the cloud system which is modified to adapt wireless mobile terminal device combines two layers as follows:
  - **Service Layer**: manages the service applications in the cloud. In addition, it manages the data in the cloud and provides data to the clients. This layer receives the service applications developed by the MTDADE and provides service publication, version management, data integration, data searching, and service composition.
  - **Client Interface Layer**: includes the following modules: Terminal service and User management, Message management, and Sensor database and network API.

(3) **Mobile Terminal Device**: There are two methods to run an application on the mobile terminal device. One way is to run the application over the web browser which needs the support of web protocols, script parsing, and

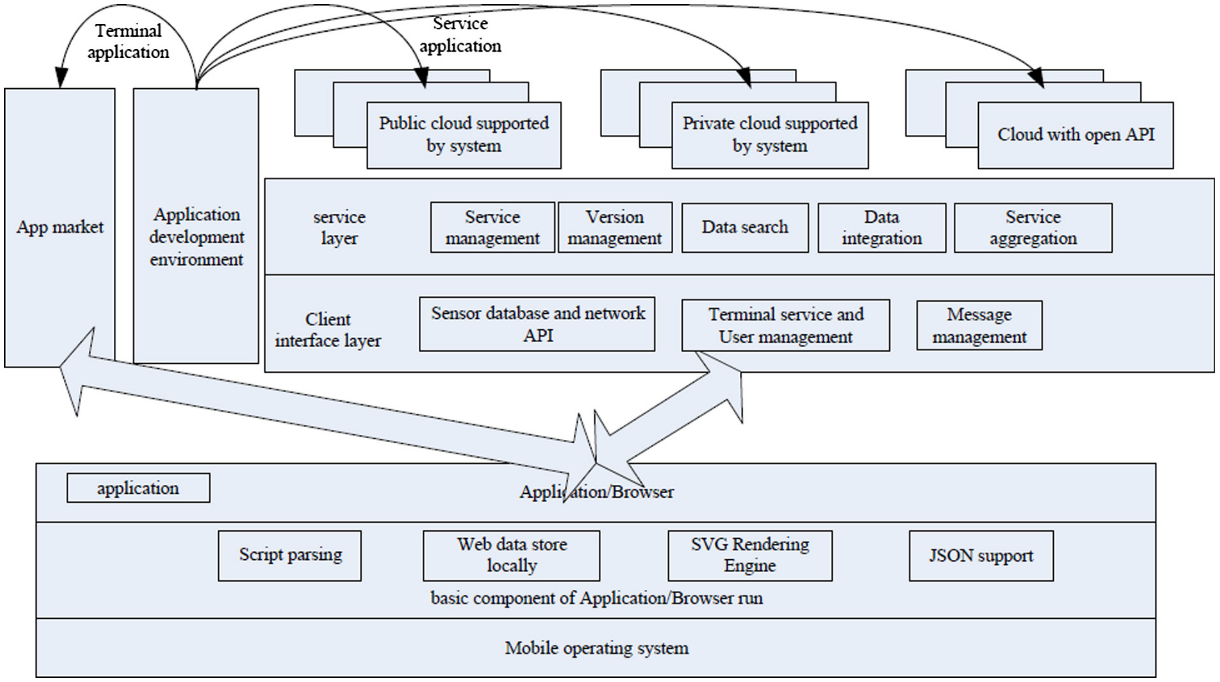


Figure 18 The system architecture [17].

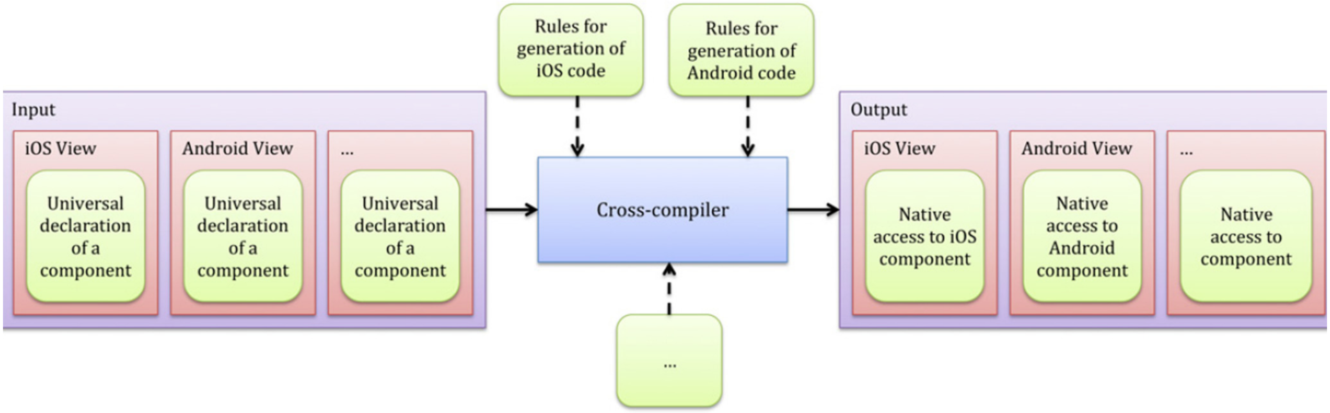

Rules for generation of iOS code
Rules for generation of Android code
Input
iOS View
Universal declaration of a component
Android View
Universal declaration of a component
...
Universal declaration of a component
Cross-compiler
...
Output
iOS View
Native access to iOS component
Android View
Native access to Android component
...
Native access to component

Figure 19 Translation of the declaration of a component to native instructions that call this component on any platform [6].

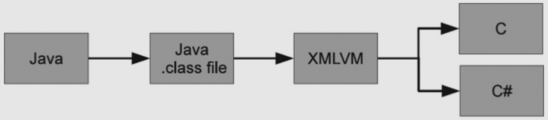
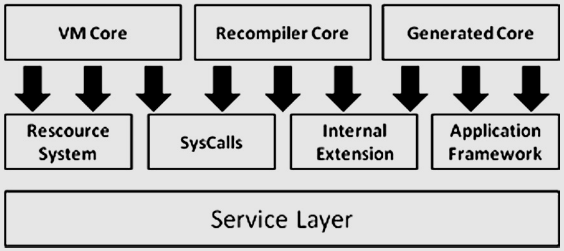
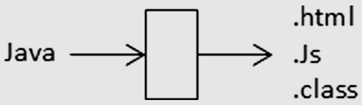
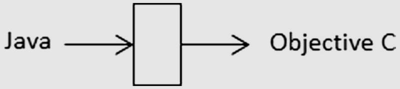
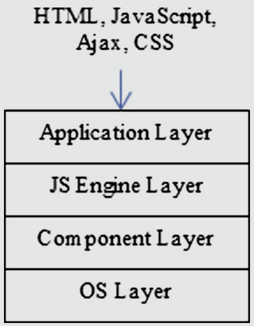
browser display. The other way is to run the application supported by the terminal platform directly which needs to get the corresponding platform App from the cloud. The mobile terminal device consists of two layers as follows:

- The bottom layer includes the following modules: Script parsing, Scalable Vector Graphics (SVG) Rendering Engine, XML parsing, and Web data store locally technology. This layer provides the facilities needed to make the application runs over the browser.
- The upper layer is application/browser which communicates to the cloud by the client interface layer in the cloud system.

The communication process of this architecture can be described as follows:

- The MTDADE allows developing cloud service applications and terminal applications. It publishes the developed terminal Apps in the App market and sends the service applications to the cloud.
- The Service management module in the cloud service management receives a message from the MTDADE and then it allocates appropriate resources for the new service application, runs the new service application and registers the new service application into the system.
- The mobile terminal device downloads the mobile terminal application from the App store.

Table 8 Comparison of the different cross-platform solutions based on the system characteristics.

Approach		Paper title/ Solution name	Model/Architecture	Supported platforms	App type	Product name	Product URL	Availability
Compilation	Cross-compiler	Cross-compiling Android applications to the iPhone (2010) [49]	 [44]	Android, iOS	Native App	XMLVM [24,44,45,49]	http://xmlvm.org	GNU Lesser General Public License, version 2.1
		Cross-Compiling Android Applications to Windows Phone 7 (2012) [44]		Android, WP7	Native App			
		MoSync [12,21]	 VM Core Recompiler Core Generated Core Rescource System SysCalls Internal Extension Application Framework Service Layer [12]	Android, iOS, BlackBerry, JavaME, Symbian, Windows Phone, Moblin, MeeGo, Windows Mobile	Native and Web Apps	MoSync	http://www.mosync.com	Open Source, GNU General Public License 2 (GPL2) & Commercial
	Trans-compiler	Towards Platform Independence of Mobile Applications (2013) [25]		Many	Web App	X	X	X
		J2ObjC [26]		Android and iOS	Native App	J2ObjC	https://github.com/google/j2objc	Open Source, command-line tool
Component-Based		Component-based mobile web application of cross-platform (2010) [18]	 HTML, JavaScript, Ajax, CSS Application Layer JS Engine Layer Component Layer OS Layer	Windows Mobile, Symbian, iOS, Palm, Android	Web App	X	X	X

(continued on next page)

Table 8 (*continued*)

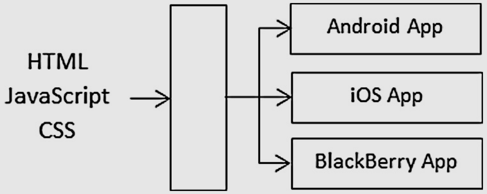
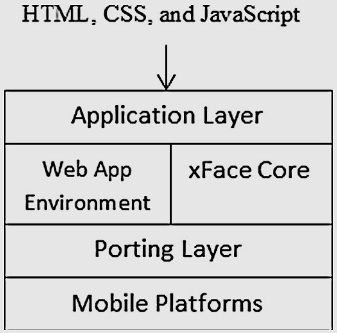
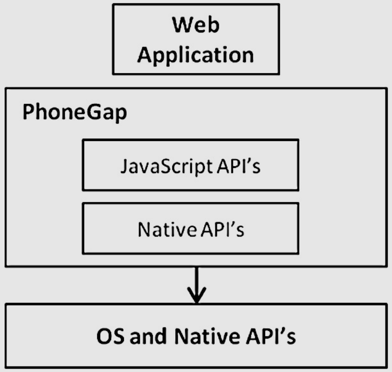
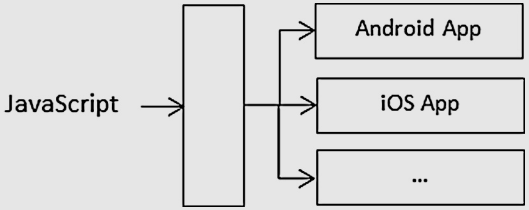
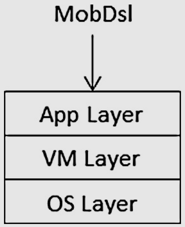
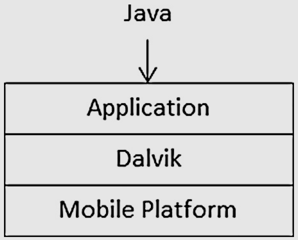
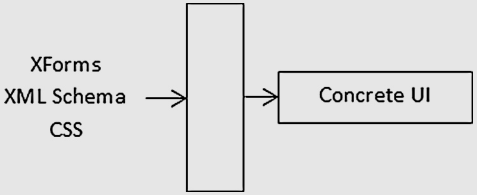
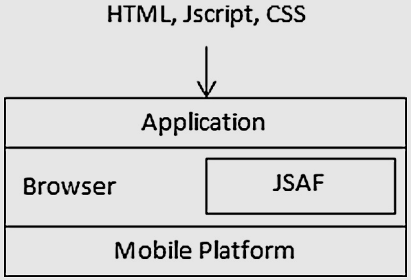
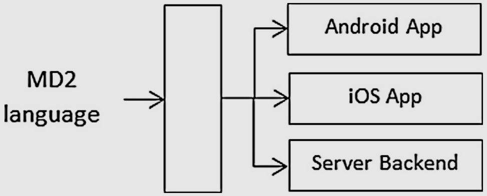
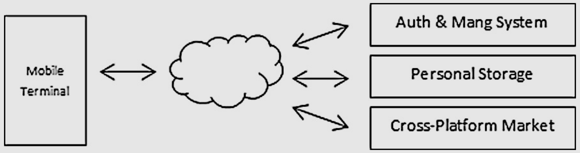
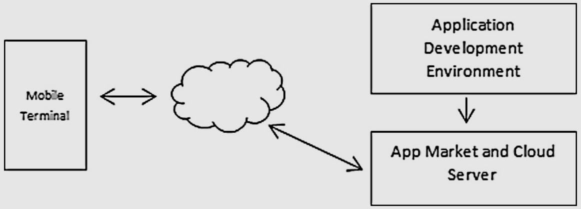
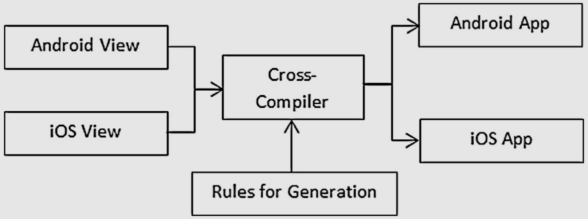
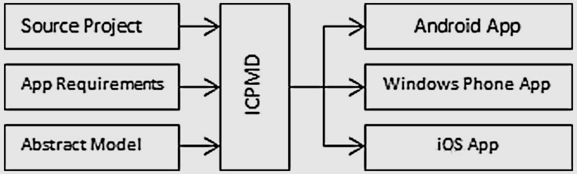
Approach		Paper title/ Solution name	Model/Architecture	Supported platforms	App type	Product name	Product URL	Availability
Interpretation	Web-Based	Generic Framework for Mobile Application Development (2011) [20]	 HTML JavaScript CSS Android App iOS App BlackBerry App	Android, BlackBerry and iPhone	Hybrid App	X	X	X
		xFace – A Lightweight Web Application Engine on Multiple Mobile Platforms (2010) [30]	 HTML, CSS, and JavaScript Application Layer Web App Environment xFace Core Porting Layer Mobile Platforms	Android, iOS, Windows Phone, etc.	Web App	xFace	https://github.com/polyvi/openxface	command line interface tool, Apache version 2.0
		PhoneGap [12,28]	 Web Application PhoneGap JavaScript API's Native API's OS and Native API's [12]	Android, Bada, iOS, BlackBerr, Symbian, webOS, Windows Phone	Hybrid App	PhoneGap	http://phonegap.com/	Open Source under Apache 2.0 license
	Runtime Interpretation	Titanium [32]		iOS, Android, Windows, Blackberry	Native, Web, and Hybrid App	Titanium	http://www.appcelerator.com/titanium/	Open source under Apache 2.0 license (Free & Commercial versions)

Table 8 *(continued)*

Approach		Paper title/ Solution name	Model/Architecture	Supported platforms	App type	Product name	Product URL	Availability
Modeling	Virtual Machine	MobDSL: A Domain Specific Language for multiple mobile platform deployment (2010) [31]		Android and iPhone (Language prototype)	X	X	X	X
		Portability of Dalvik in iOS (2012) [50]		iOS	Native App	X	X	X
	MD-UID	XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices (2008) [34]		Many	X	X	X	X
	MDD	JavaScript Application Framework for Mobile Devices (2012) [35]	 HTML, Jscript, CSS Application Browser JSAF Mobile Platform	Many	Web App	X	X	X

(continued on next page)

Table 8 (*continued*)

Approach	Paper title/ Solution name	Model/Architecture	Supported platforms	App type	Product name	Product URL	Availability
	MD2 (2013) [36,37]		Android and iOS	Native App	MD^2 (MD2)	http://wwu-pi.github.io/md2-web/	open source under Apache License 2.0
Cloud-Based	Multi-Platform Mobile Thin Client Architecture in Cloud (2011) [42]	 Mobile Terminal Auth & Mang System Personal Storage Cross-Platform Market	Android, iOS, and Windows Phone 7	Native App	X	X	X
	A Cross-platform Application Development Environment Supported by Cloud (2012) [17]	 Mobile Terminal Application Development Environment App Market and Cloud Server	Android, embedded linux, iOS, and windows phone7	Cloud Service App and Native App	X	X	X
Merged	Component Based Framework to Create Mobile Cross-platform Applications (2013) [6]	 Android View iOS View Cross-Compiler Rules for Generation Android App iOS App	Android and iOS	Native App	X	X	X
	ICPMD: Integrated Cross-Platform Mobile Development Solution (2014) [43]	 Source Project App Requirements Abstract Model ICPMD Android App Windows Phone App iOS App	Android and Windows Phone 8	Native App	ICPMD	http://www.icpmdsolution.com/	X

- The mobile terminal device runs the App and sends a request to the Terminal service and User management module in the cloud.
- The Terminal service and User management module receives the request and recognizes the device type and the App type by their identifiers and then sends this information to the Service management and Version management modules.
- The Service management module uses the Data searching, Data integration, and Service aggregation modules in the cloud to process the request sent from the terminal and returns the result to that terminal.
- The Version management module pushes a message to the terminal device via the Message management module.

The limitations of this solution include the following:

- It is still a prototype of the application development environment.

4.6. Merged approach

The main idea of this approach is to merge multiple approaches together to benefit from the advantages of these approaches and minimize the drawbacks of each individual approach. The next subsections explains the following solutions: an unnamed solution that was mentioned in a paper with title "*Component Based Framework to Create Mobile Cross-platform Applications*" and ICPMD [43].

4.6.1. Component Based Framework to Create Mobile Cross-platform Applications [6]

This tool merges between the Component-Based approach, the Cross-Compilation approach, and a new tailored language. A set of components were defined to access any hardware native functions like camera, GPS and any native software like buttons, text fields, and map. Different implementations for these components can be done for any targeted platform and each component has a common interface (independent of any platform). This tool allows the developer to develop Apps based on native code and a new defined language (similar to Java annotations). The new language consists of a layer added to the App, giving access to the set of components and their methods. The developer implements the minimal structure of the App (i.e. the user interface and navigation) in native code and specify how and where to integrate the components using the new defined language. Then, the associated tool manages the code integration inside the native code. This solution needs several steps to be fully achieved as follows:

(1) **Component layer**: each component consists of one public interface and each targeted platform will have a particular native implementation. The different implementations are hidden to the users.
(2) **Universal language layer**: to access a component public interface, a declarative language was implemented that users could use (it is a link between component methods and Apps). This language allows users to declare, instantiate a component, and use its methods from its public interface. Annotations can be written in any place of source code and linked with any methods or variables.
(3) **Component integration**: the components integration in a native application.

Fig. 19 shows how a component declaration is translated to native instructions that call this component on any platform. There are two solutions to integrate a component into a native application: either at compilation time or at execution time. This solution implements a cross-compiler that manages the native source code and the new defined language instructions. The integration of components in the source code at compilation time allows having an efficient App. Besides, the components are developed for the target platform with native instructions so the translation between the defined language and native instructions is simple. The rules to link a component is straightforward; the cross-compiler will only have to instantiate and produce the native function calls with the parameters defined by the users in the new language. So, the generated Apps will be static. To support dynamic Apps (context-aware Apps), two types of components are needed: static and context-aware. The static components will be integrated at the compilation time, whereas the others will be loaded at the execution time by a module dedicated to the context monitoring. The components loaded by this module differ according to the context (the platform itself is a context element). To support a new operating system, two steps are needed: (1) implementing components for the new platform, and (2) adding a simple file of rules for the new platform in the cross-compiler.

The limitations of this solution include the following:

- The user interface should be defined manually for each supported platform.
- This solution focuses on the common functions among target platforms and does not consider the platform-specific functions.

4.6.2. Integrated Cross-Platform Mobile Development (ICPMD) solution [43]

The ICPMD solution provides three usage scenarios to the developer based on his inputs as shown in Fig. 20: (1) **the first scenario**: the developer has a source project (e.g. WP8 project) and wants to produce it to other target platforms (e.g. iOS and Android), (2) **the second scenario:** the developer has a set of requirements and wants to produce a mobile App for the target platforms, and (3) **the third scenario**: the developer has the Abstract model project and wants to update then save it or produce a mobile App for the target platforms.

The limitations of this solution include the following:

- The proposed solution is currently in its first phase and a lot of development and maintenance are needed to be fully functioning.

5. Comparison of the cross-platform mobile development solutions

The previous section explains in details the different cross-platform mobile development approaches and sample solutions for each approach. To summarize the different cross-platform mobile development solutions, a comparison of the system characteristics of the different solutions is shown in Table 8. The comparison will include the solution characteristics like the system model or architecture, the target platforms, the type of the output App (Web, Native, or Hybrid), the product name and product URL if exists, and the availability of the solution.

6. Open research areas

Although there are many cross-platform mobile development solutions, they have many limitations and they do not tackle or solve all the challenges of mobile development. Therefore, more research and development are required. This section identifies some of the most promising areas for cross-platform mobile development research, including the following:

(1) **Different Mobile Platforms Support**: the main aim of any cross-platform mobile development solution is to support the different platforms and to simplify the supporting of a new platform. Most of the existing cross-platform solutions focus mainly on the common features of the different platforms and ignore the platform-specific differences. These observations raise some research issues, including the following:
- The need to provide new alternatives to simplify the support of different mobile platforms.
- The need to support the different versions of the same mobile platform (i.e. support Windows Phone 7/7.5/8/8.1).
- The need to provide solutions that support the common features in the different platforms along with the specific features of each platform.
- Each platform vendor regularly produces new versions of the supported platform. The new versions may include changes in the development tools, programming languages, or APIs of the mobile platform. Therefore, there is a need to introduce new cross-platform solutions that adapt properly to the changes done by the platform vendors.

(2) **Native Programming Languages Support**: many cross-platform mobile development solutions are based on tailored DSLs or specific programming languages (for example, Titanium is based on Javascript). Therefore, the developer has to learn a new language to use any of these cross-platform solutions. This observation raises some research issues, including the following:
- The need to provide solutions that help the developer to use his favorite programming languages to build the mobile Apps (e.g. Java for Android, Objective C for iOS, and C# for Windows Phone).
- The need to provide facilities that map between the different native programming languages and also consider the APIs differences of these languages.

(3) **User Interface Support:** many of the existing cross-platform mobile development solutions focus to generate the source code of the App and do not support the UI generation. Also, the cross-platform solutions that are based on the web technologies have a main limitation: the generated Apps lack the native look and feel. Therefore, there is a need to introduce solutions that:

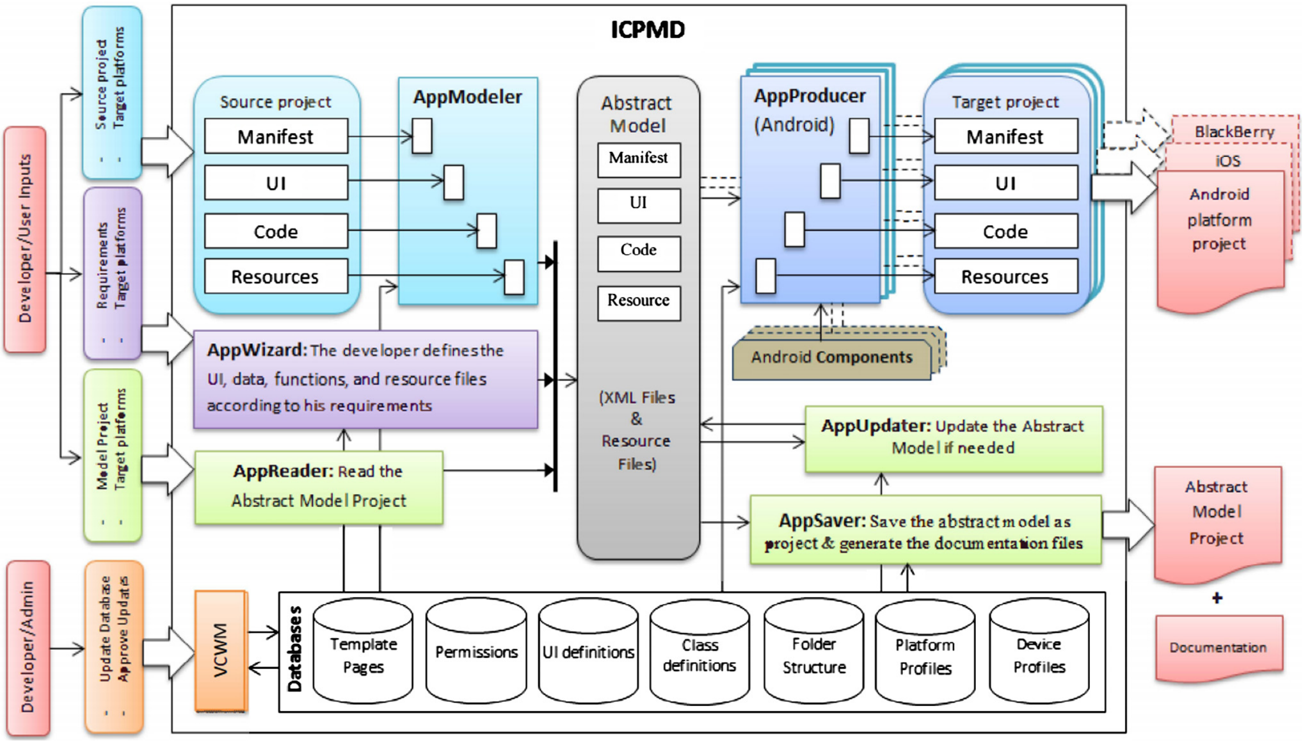

ICPMD
Developer/User Inputs
Source project - Target platforms
Requirements - Target platforms
Model Project - Target platforms
Developer/Admin
Update Database - Approve Updates
Source project
Manifest
UI
Code
Resources
AppModeler
AppWizard: The developer defines the UI, data, functions, and resource files according to his requirements
AppReader: Read the Abstract Model Project
Abstract Model
Manifest
UI
Code
Resource
(XML Files & Resource Files)
AppProducer (Android)
Android Components
Target project
Manifest
UI
Code
Resources
BlackBerry
iOS
Android platform project
AppUpdater: Update the Abstract Model if needed
AppSaver: Save the abstract model as project & generate the documentation files
Abstract Model Project
+
Documentation
VCWM
Databases
Template Pages
Permissions
UI definitions
Class definitions
Folder Structure
Platform Profiles
Device Profiles

Figure 20 The architecture of the ICPMD solution [43].

- Maintain the native look and feel of the generated Apps user interface.
- Consider the different screen sizes of smartphone devices and tablets (i.e. 4-inch, 7-inch, 10-inch, and the smart TV screen can be larger than 60-inch).
- Consider the user interface styling and themes.

(4) **Source Code Reuse**: most of the existing cross-platform mobile development solutions do not support the code reuse of the existing legacy Apps. Therefore, the developer has to rewrite these Apps in order to upgrade them to a higher version of the same platform, or port them to other mobile platforms. This observation raises the following research issue:

- The need to provide approaches to reuse the source code of the existing legacy native Apps. These approaches support the developer to convert a legacy App to the newer version of the same platform or convert it to the equivalent App for another platform.

(5) **Generating Full Mobile Applications**: there is a need to introduce solutions that consider generating full mobile applications instead of focusing on source code or user interface transformations only.

7. Conclusion

The cross-platform mobile development solutions extend the software development lifecycle by writing the mobile application once and run it on different platforms to save the time and efforts of the developers. Many of these cross-platform solutions are still under research and development, including MD2 and XMLVM. These cross-platform solutions are based on different approaches. This paper surveys the cross-platform approaches, including **Cross-Compilation** approach, **Virtual Machine** approach, **Model-Driven Development** approach, and **Web-Based** approach. Also, this paper proposes sub-categorization for several approaches and includes the most recent approaches: **Component-Based** approach**, Cloud-Based** approach, and **Merged** approach.

This paper helps the researchers to know the existing cross-platform mobile development approaches and the open research areas in this field. This paper provides detailed descriptions for each approach, including the following: (1) describe the main idea of the approach and its sub-approaches, (2) define the pros and cons of the approach/sub-approach, and (3) introduce a sample of the cross-platform mobile development solutions (per approach/sub-approach) and their limitations.

The mobile application types include web App, native App, and hybrid App. The native and hybrid Apps are more widely used than the web Apps because they can be downloaded from the App stores. Titanium and Xamarin are commercial solutions that are widely used to produce native Apps. These solutions are based on the **Runtime Interpretation** approach. PhoneGap is a solution that is widely used to produce hybrid Apps. This solution is based on the **Web-Based** approach. However, till now the final solution that solves the entire cross-platform mobile development problem does not exist. The new trend that will probably play a major role in the future is the **Merged** approach which merges multiple approaches together to benefit from the advantages of these approaches and minimize the drawbacks of each individual approach. This paper ends with some of the most promising areas in the cross-platform mobile development research, including the following: (1) different mobile platforms support, (2) native programming languages support, (3) user interface support, (4) source code reuse, and (5) generating full mobile applications.

References

[1] Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014 <http://www.gartner.com/newsroom/id/2944819> [accessed 24.05.15].

[2] Akinkuolie BB, Chia-Feng L, Shyan-Ming Y. A cross-platform mobile learning system using QT SDK framework. In: 2011 Fifth International Conference on Genetic and Evolutionary Computing (ICGEC); 2011. p. 163–7.

[3] ElYamany HF, Yousef AH. A mobile-quiz application in Egypt. In: 4th IEEE international E learning conference, Bahrain; 2013.

[4] Wu E-K, Yen SS, Hsiao WT, Tsai CH, Chen YJ, Lee WC, et al. Cross-platform mobile personal health assistant APP development for health check. In: Huang Y-M, Chao H-C, Deng D-J, Park JJ, editors. Advanced technologies, embedded and multimedia for human-centric computing, vol. 260. Netherlands: Springer; 2014. p. 1257–68.

[5] El-Kassas W, Solyman A, Farouk M. mTourism multilingual integrated solution: a case study "EgyptTravel". In: eChallenges e-2014, 2014 conference; 2014. p. 1–9.

[6] Perchat J, Desertot M, Lecomte S. Component based framework to create mobile cross-platform applications. Procedia Comput Sci 2013;19:1004–11.

[7] Corral L, Janes A, Remencius T. Potential advantages and disadvantages of multiplatform development frameworks – a vision on mobile environments. Procedia Comput Sci 2013;10:1202–7.

[8] Holzinger A, Treitler P, Slany W. Making apps useable on multiple different mobile platforms: on interoperability for business application development on smartphones. In: Quirchmayr G, Basl J, You I, Xu L, Weippl E, editors. Multidisciplinary research and practice for information systems, vol. 7465. Berlin Heidelberg: Springer; 2012. p. 176–89.

[9] Raj R, Tolety SB. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: 2012 Annual IEEE India Conference (INDICON); 2012. p. 625–9.

[10] Smutny P. Mobile development tools and cross-platform solutions. In: 2012 13th International Carpathian Control Conference (ICCC); 2012. p. 653–6.

[11] Ribeiro A, da Silva AR. Survey on cross-platforms and languages for mobile apps. In: 2012 eighth international conference on the Quality of Information and Communications Technology (QUATIC); 2012. p. 255–60.

[12] Palmieri M, Singh I, Cicchetti A. Comparison of cross-platform mobile development tools. In: 2012 16th International Conference on Intelligence in Next Generation Networks (ICIN); 2012. p. 179–86.

[13] Xanthopoulos S, Xinogalos S. A comparative analysis of cross-platform development approaches for mobile applications. In: Presented at the proceedings of the 6th Balkan conference in informatics, Thessaloniki, Greece; 2013.

[14] Ohrt J, Turau V. Cross-platform development tools for smartphone applications. Computer 2012;45:72–9.

[15] Heitkötter H, Hanschke S, Majchrzak T. Evaluating cross-platform development approaches for mobile applications. In:

Cordeiro J, Krempels K-H, editors. Web information systems and technologies, vol. 140. Berlin Heidelberg: Springer. p. 120–38.

[16] Yung-Wei K, Chia-Feng L, Kuei-An Y, Shyan-Ming Y. A cross-platform runtime environment for mobile widget-based application. In: 2011 International conference on cyber-enabled distributed computing and knowledge discovery (CyberC); 2011. p. 68–71.

[17] Baixing Q, Tian-zhou C, Hongjun D, Bin P, Minghui W. A cross-platform application development environment supported by cloud service. In: 2012 IEEE 14th international conference on high performance computing and communication & 2012 IEEE 9th international conference on embedded software and systems (HPCC-ICESS); 2012. p. 1421–7.

[18] Biao P, Kun X, Lei L. Component-based mobile web application of cross-platform. In: 2010 IEEE 10th international Conference on Computer and Information Technology (CIT); 2010. p. 2072–7.

[19] Young-Hyun C, Kyung-Bae Y, Dea-Woo P. A study on the development of one source multi use cross-platform based on zero coding. In: 2013 International conference on Information Science and Applications (ICISA); 2013. p. 1–3.

[20] Sambasivan D, John N, Udayakumar S, Gupta R. Generic framework for mobile application development. In: 2011 Second Asian Himalayas International Conference on Internet (AH-ICI); 2011. p. 1–5.

[21] *MoSync* <http://www.mosync.com> [accessed 24.05.15].

[22] Corona <https://coronalabs.com/> [accessed 24.05.15].

[23] Neomades <http://neomades.com/en/> [accessed 24.05.15].

[24] XMLVM <http://xmlvm.org> [accessed 24.05.15].

[25] Klima P, Selinger S. Towards platform independence of mobile applications. In: Moreno-Díaz R, Pichler F, Quesada-Arencibia A, editors. Computer aided systems theory – EUROCAST 2013, vol. 8112. Berlin Heidelberg: Springer; 2013. p. 442–9.

[26] J2ObjC <https://github.com/google/j2objc> [accessed 24.05.15].

[27] JUniversal <http://juniversal.org/> [accessed 24.05.15].

[28] PhoneGap. Available: http://phonegap.com/ [Last Visited: 24/5/2015].

[29] Rhomobile <http://rhomobile.com/> [Accessed 24.05.15].

[30] Fan J, Zhigang F, Lei L. xFace. A lightweight web application engine on multiple mobile platforms. In: 2010 IEEE 10th international conference on Computer and Information Technology (CIT); 2010. p. 2055–60.

[31] Kramer D, Clark T, Oussena S. MobDSL: a domain specific language for multiple mobile platform deployment. In: 2010 IEEE international conference on Networked Embedded Systems for Enterprise Applications (NESEA); 2010. p. 1–7.

[32] Titanium <http://www.appcelerator.com/titanium/> [accessed 24.05.15].

[33] Xamarin <http://xamarin.com/> [accessed 24.05.15].

[34] Viana W, Andrade RMC. XMobile: a MB-UID environment for semi-automatic generation of adaptive applications for mobile devices. J Syst Softw 2008;81:382–94.

[35] Raghu R, Shobha KR. JavaScript application framework for mobile devices. In: Krishna PV, Babu MR, Ariwa E, editors. Global trends in computing and communication systems, vol. 269. Springer: Berlin Heidelberg; 2012. p. 291–9.

[36] Heitkötter H, Majchrzak TA, Kuchen H. Cross-platform model-driven development of mobile applications with md^2. In: Presented at the proceedings of the 28th annual ACM symposium on applied computing, Coimbra, Portugal; 2013.

[37] Heitkötter H, Majchrzak T. Cross-platform development of business apps with MD2. In: Brocke J, Hekkala R, Ram S, Rossi M, editors. Design science at the intersection of physical and virtual design, vol. 7939. Berlin Heidelberg: Springer; 2013. p. 405–11.

[38] Martinez-Ruiz FJ, Vanderdonckt J, Arteaga JM. Context-aware generation of user interface containers for mobile devices. In: 2008. ENC '08. Mexican international conference on computer science; 2008. p. 63–72.

[39] Meskens J, Luyten K, Coninx K. Jelly: a multi-device design environment for managing consistency across devices. In: Presented at the proceedings of the international conference on advanced visual interfaces, Roma, Italy; 2010.

[40] Balagtas-Fernandez F, Tafelmayer M, Hussmann H. Mobia Modeler: easing the creation process of mobile applications for non-technical users. In: Presented at the proceedings of the 15th international conference on intelligent user interfaces, Hong Kong, China; 2010.

[41] Quinton C, Mosser S, Parra C, et al. Using multiple feature models to design applications for mobile phones. In: Presented at the proceedings of the 15th international software product line conference, vol. 2, Munich, Germany; 2011.

[42] Tang W, Lee J-h, Song B, Islam M, Na S, Huh E-N. Multi-platform mobile thin client architecture in cloud environment. Procedia Environ Sci 2011;11(Part A):499–504.

[43] El-Kassas WS, Abdullah BA, Yousef AH, Wahba A. ICPMD: integrated cross-platform mobile development solution. In: 2014 9th International Conference on Computer Engineering & Systems (ICCES); 2014. p. 307–17.

[44] Antebi O, Neubrand M, Puder A. Cross-compiling android applications to Windows Phone 7. In: Zhang J, Wilkiewicz J, Nahapetian A, editors. Mobile computing, applications, and services, vol. 95. Berlin Heidelberg: Springer; 2012. p. 283–302.

[45] Puder A, Antebi O. Cross-compiling android applications to iOS and Windows Phone 7. Mob Netw Appl 2013;18:3–21.

[46] The MoSync Toolchain <http://www.mosync.com/docs/sdk/tools/guides/architecture/toolchain/index.html> [accessed 24.05.15].

[47] MoSync Runtime Architecture <http://www.mosync.com/docs/sdk/tools/guides/architecture/runtime-architecture/index.html> [accessed 24.05.15].

[48] Ciman M, Gaggi O, Gonzo N. Cross-platform mobile development: a study on apps with animations. In: Presented at the Proceedings of the 29th annual ACM symposium on applied computing, Gyeongju, Republic of Korea; 2014.

[49] Puder A. Cross-compiling Android applications to the iPhone. In: Presented at the proceedings of the 8th international conference on the principles and practice of programming in Java, Vienna, Austria; 2010.

[50] Gao S, Zheng T. Portability of Dalvik in iOS. In: 2012 international conference on Computer Science & Service System (CSSS); 2012. p. 531–7.