Enhanced Code Conversion Approach for the Integrated Cross-Platform Mobile Development (ICPMD)

Wafaa Samy El-Kassas, Bassem Amin Abdullah, Ahmed Hassan Yousef, and Ayman M. Wahba

Abstract—Mobile development companies aim to maximize the return on investments by making their mobile applications (Apps) available on different mobile platforms. Consequently, the same App is developed several times; each time the developer uses the programming languages and development tools of a specific platform. Therefore, there is a need to have cross-platform mobile applications development solutions that enable the developers to develop the App once and run it everywhere. The Integrated Cross-Platform Mobile Applications Development (ICPMD) solution is one of the attempts that enables the developers to use the most popular programming languages like Java for Android and C# for Windows Phone 8 (WP8). ICPMD is used to transform both the source code and user interface to another language to generate full Apps on the target platform. This paper extends ICPMD by proposing a new code conversion approach based on XSLT and Regular Expressions to ease the conversion process. In addition, it provides the assessment method to compare the ICPMD efficiency with competing approaches. Several Apps are converted from WP8 to Android and vice versa. The ICPMD evaluation results show reasonable improvement over commercial cross-platform mobile development tools (Titanium and Xamarin).

Index Terms—Cross-platform mobile development, code conversion, code reuse, generated apps, ICPMD; source code patterns

1 Introduction

The number of smartphone users is growing rapidly because of the availability of various mobile applications (Apps) that serve them in their daily life. Therefore, there is a demand to produce more Apps in different fields such as education [1], [2], [3], tourism [4], [5], environment [6], governmental services, and entertainment. There are several mobile platforms such as Android, Windows Phone (WP), iOS, and BlackBerry. Each platform vendor provides different programming languages, APIs, and development tools for the developers. Therefore, the developer has to develop the same App several times using different programming languages and libraries in order to produce one App that runs on different mobile platforms. This causes the waste of a lot of time and efforts. Consequently, mobile development companies start to use the cross-platform mobile applications development solutions. The main concept of the cross-platform solutions is to develop the App once and run it everywhere. There are many cross-platform mobile applications development solutions that are used commercially like: Titanium [7], Xamarin [8], and PhoneGap [9]. On the other hand, several tools are still under research and development like: XMLVM [10], J2ObjC [11], MD2 [12], [13], and others.

The cross-platform solutions use different approaches such as Cross-Compilation, Model Driven Development (MDD), Runtime Interpretation, and Component-Based. There are many surveys [14], [15], [16], [17], [18], [19], [20], [21], [22] concentrated on the different cross-platform mobile applications development approaches. Most of these approaches are still under research and development and have limitations. These limitations include:

1. Some solutions require the developer to learn a new programming language to use the solution. MD2 has this limitation because it is based on a dedicated Domain Specific Language (DSL).
2. Most of the existing cross-platform solutions do not support reusing source code of existing native applications. For example, if the developer wants to deploy a legacy native application to other platforms using Titanium, he has to rewrite this application using the Titanium specific programming language (Javascript) and specific APIs of Titanium.
3. Many solutions do not support the user interface generation, so the developer has to develop the user interface natively for each generated application. This limitation applies to J2ObjC and JUniversal [23].

The Integrated Cross-Platform Mobile Applications Development (ICPMD) solution [24] attempted to overcome most of these limitations. The ICPMD applied transformations to convert the application project of Windows Phone 8 (WP8) platform (including source code, user interface, and resources) to the equivalent application project of Android

• *The authors are with the Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Cairo, Egypt. E-mail: wafaa.elkassas@gmail.com, {babdullah, ahassan, ayman.wahba}@eng.asu.edu.eg.*

Manuscript received 19 May 2015; revised 27 Feb. 2016; accepted 9 Mar. 2016. Date of publication 0 . 0000; date of current version 0 . 0000. Recommended for acceptance by R. Mirandola. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSE.2016.2543223

TABLE 1
Pros and Cons of Mobile Apps Types

	Pros	Cons
Web App	• Easy to learn and develop using the web technologies. • Developed once and is accessed on different platforms using mobile browsers.	• Cannot be downloaded from the Apps store. • Cannot access the mobile device features. • Less performance.
Native App	• Full access to the mobile device features. • Higher performance. • Native look and feel.	• Downloadable from the Apps store. • More difficult to learn and develop. • Developed separately for the different platforms.
Hybrid App	• Can access the mobile device features. • Developed using the web technologies and rendered in a native App, hence the webpages can be reused across different platforms.	• Downloadable from the Apps store. • Less performance than the native Apps.

platform. However, the code conversion approach that has been used in ICPMD was very complex.

This paper extends the previous work and proposes a new code conversion approach by using XSLT and Regular Expressions. The main idea of this approach is to search in the input source code files for matching a predefined set of code patterns and then generate the equivalent source code for the target platform.

The contributions of this paper include: 1) proposing a new approach for code conversion by using XSLT and Regular Expressions, 2) proposing a criteria to evaluate the efficiency of the proposed code conversion approach, and 3) evaluating the efficiency of the generated Apps from the enhanced ICPMD compared with the native counterpart and the generated Apps from other cross-platform mobile development solutions.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 gives a background about the ICPMD solution. Section 4 explains the methodology of enhancing the code conversion approach for the ICPMD solution. Section 5 focuses on the proposed code conversion approach in details and presents two code conversion examples. Section 6 includes the evaluation of the generated applications and comparison to other alternatives. Finally, Section 7 presents the conclusions and future work.

2 Related Work

In [22], El-Kassas et al. categorize the approaches adopted by several different cross-platform mobile development solutions as follows:

1. Compilation approach which includes two sub-approaches: Cross-Compiler and Trans-Compiler.
2. Component-Based approach.
3. Interpretation approach which includes three sub-approaches: Virtual Machine, Web-Based, and Runtime Interpretation.
4. Modeling approach which includes two sub-approaches: Model-Based User Interface Development (MB-UID) and Model Driven Development.
5. Cloud-Based approach.
6. Merged approach.

These approaches are used with the different types of mobile applications that include web App, native App, and hybrid App. Web App is mainly based on web technologies such as HTML5 and Javascript. Native App is developed using the tools and programming languages provided for a certain mobile platform. 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 a native App. Table 1 shows the pros and cons of the different types of mobile applications.

There are several tools that are used for cross-platform mobile development. For example, PhoneGap is the most commonly used solution to produce hybrid Apps [25], [26]. This solution uses the Web-Based approach. The most widely used commercial solutions [25], [26] to produce native Apps are: Titanium and Xamarin. Titanium uses the *Runtime Interpretation approach*. Xamarin produces native Apps in a different way for each supported platform. For iOS, C# is Ahead-of-Time (AOT) compiled to produce native ARM assembly code. For Android, C# is compiled to Intermediate Language (IL), which is Just-in-Time (JIT) compiled to produce native assembly when the application launches. In both cases, Xamarin Apps utilize a runtime that automatically handles issues like garbage collection, memory allocation, and underlying platform interop [27]. For WP, C# is compiled to IL and executed by the built-in runtime. The new proposed solution (ICPMD) focuses to produce native Apps and use the Merged approach by combining the *Trans-Compilation approach* and the *Model-Driven Development approach*.

In this section, some cross-platform mobile applications development approaches will be briefly introduced because they are important in understanding the whole paper domain. These approaches include: 1) Trans-Compiler, 2) Cross-Compiler, 3) Runtime Interpretation, and 4) MDD approaches. The selected approaches represent the approaches of the solutions: Titanium, Xamarin, and ICPMD. For each approach, one or more cross-platform mobile applications development solutions are provided. These examples are selected because they inspire and motivate the design decisions of the proposed ICPMD solution. More comprehensive details about the different cross-platform mobile applications development approaches and solutions are found in several survey papers including [14], [15], [16], [17], [18], [19], [20], [21], [22].

2.1 Trans-Compiler Approach

A trans-compiler is used to transform one high-level programming language to another high-level programming language. In general, the code conversion is a nontrivial task [28] because mapping the API between two languages is difficult for the following reasons: 1) the names of classes are different, 2) the names of methods are different, and 3)

the count and types of methods' parameters are different. Solutions that use this approach include: J2ObjC, JUniversal, and the tool described in [29].

In [29], a tool is proposed to convert the Android Apps to platform independent web Apps by using the Google Web Toolkit (GWT). The converter uses the Eclipse Java Development Tools (JDT) to process the source code, because both Android and target GWT system have their own custom Java library (subset of the Java runtime library). The main task of the converter is to handle the differences in these two Java libraries. The converter has to identify all statements in the source code which are not supported in the target system. The Android source code is parsed to produce AST. If the converter detects library calls which are not supported by the GWT (Java-to-JavaScript compiler), the syntax tree is going to be modified. After that, the GWT compiler translates the Java source for the client side to JavaScript code and for the server side to Java byte code. Features in the Android source system which are neither supported by GWT nor by HTML5 are ignored.

The J2ObjC tool [11] translates the Java code to Objective-C code for the iOS platform. The J2ObjC is an open-source command-line tool from Google, which is currently between alpha and beta quality. 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 UI in Java (the application logic) and then use this tool to convert the application to iOS application (without user interface). J2ObjC supports most of the features of the Java language and its runtime, including: exceptions, inner and anonymous classes, generic types, threads, and reflection. In addition, the JUnit test translation and execution are supported.

Microsoft recently announced the JUniversal tool [23] which converts the Java source code files of the Android platform to the equivalent C# files of the Windows/Windows Phone platform. JUniversal doesn't provide any support for UI today and the developer has to write it natively.

2.2 Cross-Compiler Approach

A cross-compiler is a compiler that runs on a computer that has an operating system that is different from the one on which the compiled program will run. XMLVM is a solution that uses this approach.

The XMLVM tool [30], [31], [32], [33] is a byte code level cross-compiler. Instead of cross-compiling on the source code level, XMLVM cross-compiles the byte code instructions from Sun Microsystem's virtual machine. XMLVM does not only cross-compile applications on a language level, but it also maps the APIs between different platforms. The advantages of choosing the byte-code transformation include: 1) byte codes are much easier to parse than Java source code, 2) some high-level language features such as generics are already reducedto low-level byte code instructions, and 3) the Java compiler does extensive optimizations to produce efficient byte codes. However, the mapping between the source language and the target language is very difficult to be achieved. Therefore, the cross-compiler supports a few platforms and focuses only on the common elements of these platforms [34].

2.3 Runtime Interpretation Approach

Runtime is an execution environment and a layer that makes the mobile App runs on the native platform. This approach translates the code to bytecode and then, at runtime, that bytecode is executed by a virtual machine that is supported by the mobile device. Titanium is a solution that uses this approach.

Titanium [7] 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 [18].

2.4 Model Driven Development Approach

This approach is used to generate platform specific versions of the App out of a platform independent model. MD2 is a solution that uses this approach.

MD2 [12], [13] 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.

2.5 ICPMD Relation to Different Approaches

ICPMD is inspired by many of the existing cross-platform mobile applications development solutions. The ICPMD scope and design requirements are formalized as follows:

1. Focus on the native mobile development.
2. Focus on generating full Apps.
3. Support code reuse.
4. The developer does not have to learn a new programming language to use this solution.

Some tools support the native-to-web Apps conversion including [29] and other tools support the native-to-native Apps conversion including J2ObjC and JUniversal. ICPMD focuses to support the native-to-native App conversion. However, ICPMD attempts to avoid the limitations of J2ObjC and JUniversal by generating full Apps that include: source code, user interface, project files, and resources. ICPMD uses the Trans-Compilr approach for the code conversion and the MDD approach for the UI and manifest files conversion. ICPMD is designed to avoid the MD2 limitations by allowing the developer to use his preferred programming languages for native Apps development (i.e., Java for Android Apps and C# for Windows Phone Apps). MD2 and Titanium did not support code reuse of the existing native mobile Apps. ICPMD is designed with the requirement to support code reuse.

ICPMD is similar to XMLVM since both of them use an XML-based intermediate language and generate native Apps. However, these solutions are different in the internal implementations and in many other aspects such as:

- *Input:* XMLVM is based on Java as input programming language and parses the bytecode. ICPMD extends that by trying to support different languages (e.g., Java and C#) on the source code level.
- *API Mapping:* XMLVM is based on the APIs of Android. For each supported platform, many API mapping layers are developed. ICPMD is designed to be independent of any platform.

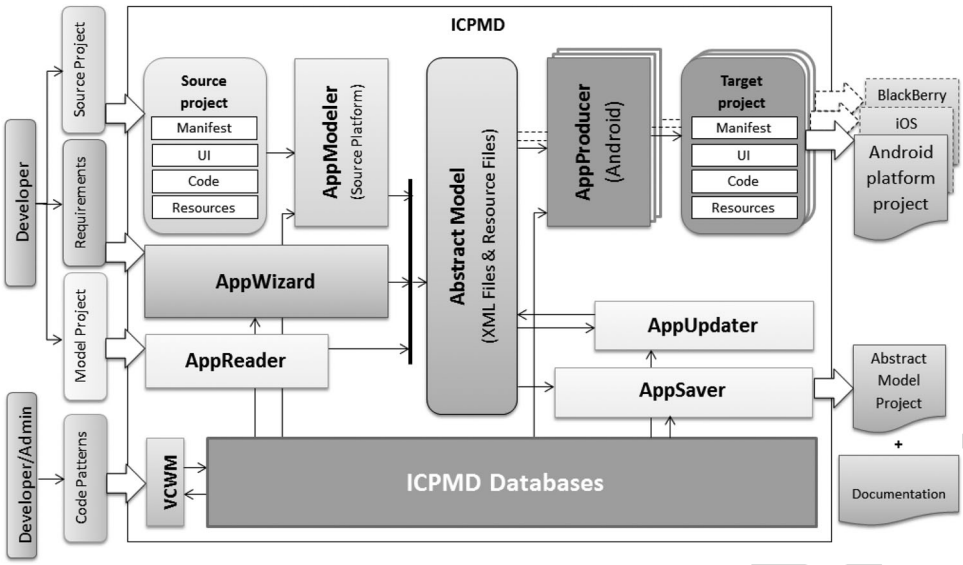

Fig. 1. The architecture of the ICPMD solution [24].

- *Readability:* ICPMD generates a readable source code that uses the native APIs of the target platform using the same names of variables and objects that are defined by the source App developer. On the other hand, the generated code from XMLVM is difficult to read and to follow due to the extensive creation of temporary variables and objects with difficult to read names (e.g., _r0, _r1_o, or _r2).

The next section presents a background about the ICPMD architecture design details.

3 ICPMD Background

The ICPMD solution focuses mainly on the native mobile applications development. To avoid the limitations of existing cross-platform solutions, ICPMD was designed to achieve a set of requirements as follows:

1. Generating full Apps: the solution handles the generation of both the user interfaces and the source code files along with other essential files (i.e., resource files and project files) to produce ready-to-compile Apps.
2. Supporting code reuse of the existing Apps source code.
3. The developer does not have to learn a new programming language to use this solution. The developer could use his preferred programming languages for native Apps development (i.e., Java for Android Apps and C# for Windows Phone Apps).

The architecture of ICPMD is shown in Fig. 1. The ICPMD solution converted WP8 source project to an intermediate abstract model project (i.e., XML files and resource files). This model project is used to generate the target platform project for Android. This solution converted the source project to an intermediate abstract model project by using the WP8 AppModeler module then the Android AppProducer module converted the abstract model project to the equivalent project for Android platform.

The AppModeler and AppProducer modules consist of four converters: Manifest Converter, User Interface (UI) Converter, Code Converter, and Resource Mapper. The Manifest and UI converters were implemented using XSLT and the Code Converter converted the input source code file to the equivalent Abstract Syntax Tree (AST) then searched for patterns in the AST to generate the equivalent source code for other platforms.

The code converter consisted of two sub-modules: Parser and Analyzer. The Parser and Analyzer convert the source code files to the corresponding XML files. The Parser (M) module was implemented by using the NRefactory open source library [35] which uses its internal "C# Language Rules" to convert the C# WP8 source code file to AST. Then, the generated AST is transformed to XML format. NRefactory was selected because it is open source and supports new versions of C#. The input of the Parser was the C# source code and the output was the AST. The Analyzer (M) module uses the AST generated by the parser with the "Patterns Rules" definitions. Its role is to identify each function pattern in the AST and replace it with the equivalent ICPMD unified component interface in the Abstract Model. This paper proposes a new code conversion approach for the ICPMD solution and the next section explains the research methodology.

4 Proposed Enhanced ICPMD Methodology

In ICPMD, although the code converters have been implemented and tested to convert Apps like miniBrowser and Built-in Camera from WP8 to Android, the definition of code patterns was very difficult because it was based on XPATH. Therefore, it is required to have a developer who is familiar with the AST structure generated from the NRefactory in order to be able to define the new code patterns. Because this solution was very complex for defining the code patterns, different code conversion alternatives are investigated as follows:

1. *The first alternative* is to build a huge database that maps the classes, methods, and attributes of one platform to the other supported platforms and then this database could be used to translate the input

source code file to the equivalent source code for other platforms. This solution has been avoided because the mapping among platforms is not a simple or a direct task. For example, a particular function could be implemented by calling one method in a certain platform while it is implemented using several lines of code in another platform as explained in example 1.

Example 1. loading URL in a webview/web-browser control is implemented in both WP8 and Android as follows:

- WP8 Input Source Code:
  web1.Navigate(new Uri(url, UriKind.Absolute));
- Equivalent Android Source Code:
  web1.setWebViewClient(new WebViewClient());
  web1.getSettings().setJavaScriptEnabled(true);
  web1.getSettings().setGeolocationEnabled(true);
  web1.setWebChromeClient(new WebChrome-Client());
  web1.loadUrl(url);

2. *The second alternative* is to treat the input source code file at a higher level by reading the source code file line by line, and then attempt to match it with a predefined list of patterns. If a pattern is matched; the equivalent source code for the other platform is retrieved from the database. The unmatched lines of code are migrated as comments in the generated source code files. The code patterns are written using Regular Expressions (Regex) and the code generators are implemented using XSLT. This paper provides a detailed explanation of this approach.

   The proposed code conversion approach adds the following advantages to the generated Apps from ICPMD:

   - *Source Code Readability*: the generated source code is readable. Usually, this code is generated with comments that explain the meaning of each line.
   - *Source Code Consistency*: all the lines of code in the source project are converted to the target platform project. The detected lines of code are converted to the target language. In addition, the undetected lines of code are copied to the target project as comments. The commented source code will be an initial hint to the developer instead of starting from scratch to implement the required functionality.
   - *Flexibility*: the developer can add new functionalities and do any customizations or updates on the generated App project before compilation.

Fig. 2 shows the workflow of the enhanced ICPMD solution (i.e., web-based system). The mobile developers enter a source project folder (e.g., Android project) to the system, and then this will be converted to the equivalent target platforms (e.g., WP8 and iOS projects) where the output includes a project folder for each target platform.

The enhanced ICPMD solution, which is inherited from the original architecture and design in [24], has the following strengths:

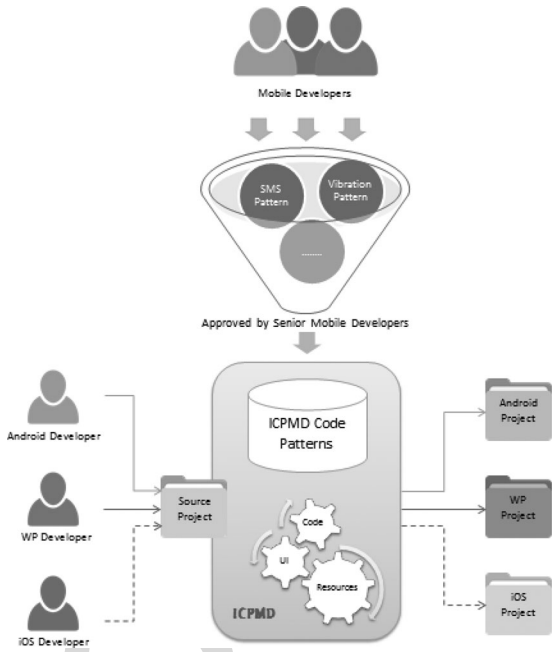

Fig. 2. Workflow of the enhanced ICPMD solution.

- *Modularity*: the modules of each platform (i.e., App-Modeler and AppProducer) are implemented separately from other platforms. If one platform needs updating, then this platform will be updated only without affecting the modules of other platforms.
- *Extensibility*: both the original ICPMD and the new enhanced version are mainly based on external configuration files (i.e., XSLT files and ICPMD database) that can be updated and extended to reflect new supported features and updates.
- *Generating Full Apps*: both the original ICPMD and the new version generate a project folder for the target platform, and then the developer can compile this project on the platform-specific compiler to produce the ready-to-publish applications.

The limitations of the new enhanced ICPMD include:

1. The AppModeler can only recognize lines of code in the source project that match the predefined list of code patterns. It converts these lines of code to the abstract model project. The unrecognized lines of code are converted to the abstract model project as comments.
2. The AppProducer does not support the following:
   - Different versions for Android or Windows Phone platforms.
   - Different screen sizes like phones and tablets.
   - Different screen resolutions when dimensions (width and height) of the user interface controls are defined.
   - Themes and style specifications.
3. The solution requires the developer to setup the compiler of each supported platform in order to compile the generated mobile applications projects.

The next section presents the proposed code conversion approach in details.

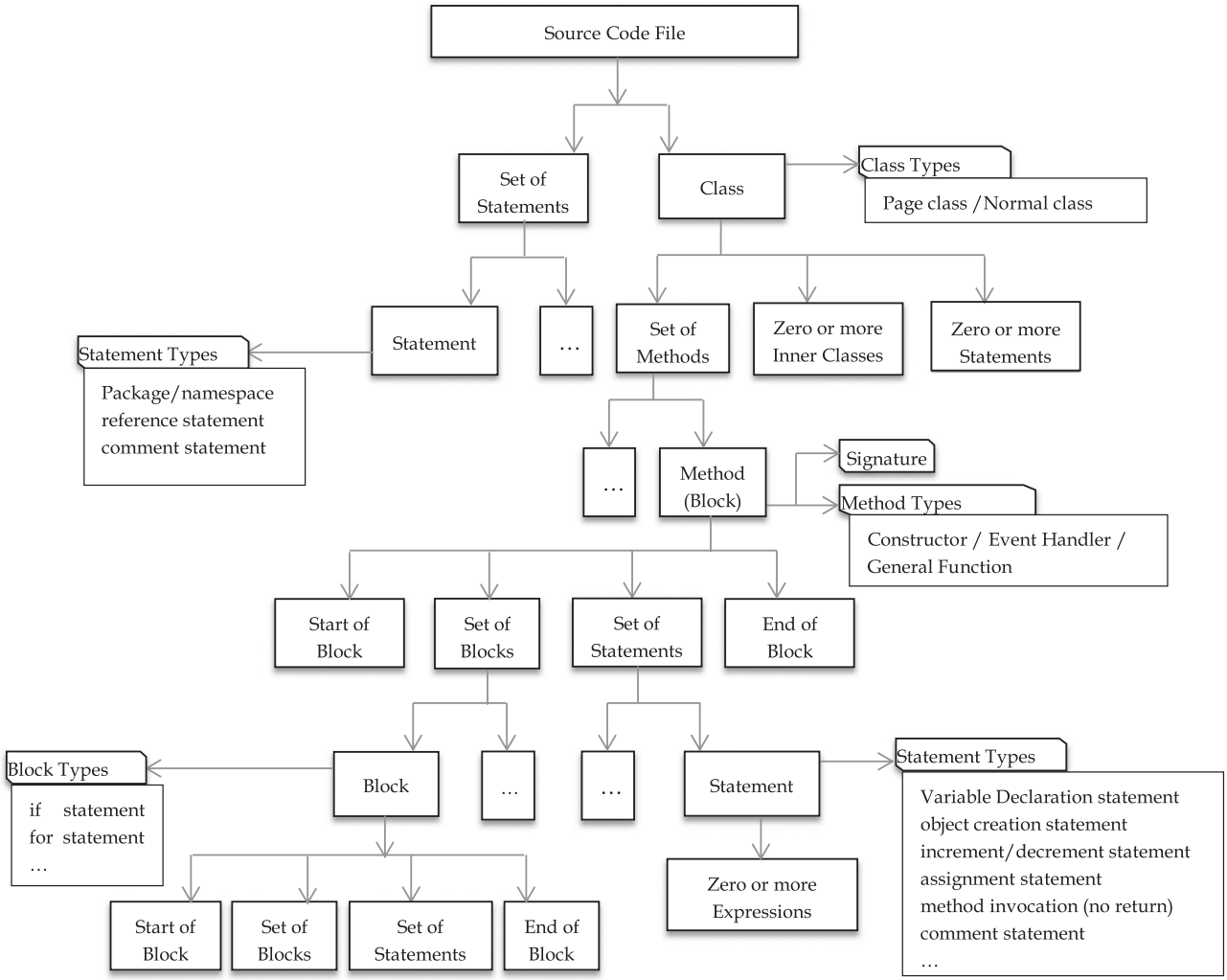

Fig. 3. Anatomy of the input source code file in mobile applications.

5 Proposed Source Code Conversion Approach

The proposed approach solves the direct mapping of APIs problem by focusing on the common functions among the mobile platforms. These functions are represented as code patterns which are written using Regular Expressions. The code patterns are stored in the new enhanced ICPMD patterns database where each record consists of the pattern for the input programming language along with the equivalent lines of code for each target platform programming language. There are different types of code patterns which are defined based on the anatomy of the input source code file.

The proposed algorithm which applies the proposed code conversion approach is called Source Code Migration (SCM) algorithm. The main idea of the SCM algorithm is to search in the input source code file for matching a predefined set of code patterns, and then get the equivalent source code for the target platform from the ICPMD database. The input to the algorithm is a source code file which is written in the source platform programming language. This input is converted to an intermediate XML representation file, and then the XML file is used to generate the equivalent source code file which is written in the programming language of the target platform.

The next sub-sections present the anatomy of the input source code file, the different types of code patterns, the SCM algorithm, and the results of implementing the proposed code conversion in the ICPMD solution along with two code conversion examples, respectively.

5.1 Source Code File Anatomy

Fig. 2 shows the anatomy of the input source code file. The source code file consists of a set of statements (package/namespace definition of the project, statements to reference other classes, and may include comment statements) and the class definition (mobile application page class or ordinary class). The class consists of a set of methods (each method could be a constructor, event handler, or an ordinary method), zero or more statements (i.e., variable declaration statement), and zero or more inner classes. Each method consists of a set of blocks (i.e., if statement, for statement, etc.) and a set of statements (i.e., variable declaration statement, and assignment statement). Each statement consists of zero or more expressions.

According to the source code anatomy in Fig. 3, each line of code in the source code file could be one of the following:

- Package/namespace statement
- Reference statement

- Class definition statement
- Comment statement
- Empty line
- Start of block (i.e., {)
- End of block (i.e.,})
- Event handler signature
- Variable declaration statement
- Object creation statement
- Increment/decrement statement
- Assignment statement
- Method invocation (no return)
- if statement
- for statement
- Others

5.2 Source Code File Patterns

The proposed approach focuses on converting the mobile functions/features from one platform to another. This applies also to the statements that exist in any programming language like: if statement, for statement, variable declaration statement, etc. The mobile platforms provide mainly the same built-in features, for example: making a call, send SMS, camera, play music, Global Positioning System (GPS), and Accelerometer. Usually, the implementation for each platform is different than the other and uses different classes and methods. In addition, the same function (e.g., send SMS) may be implemented using one statement (one source code line) in a particular platform while it is implemented using several statements in another platform. Therefore, the proposed approach focuses to convert the mobile functions from one platform to another instead of mapping the APIs in a one-to-one way. Different types of patterns are used as follows:

- A pattern that matches one line of code (Simple pattern).
- A pattern that matches part of source code line (Expression pattern).
- A pattern that matches several lines of source code at once (Composite pattern).

Each pattern statement/expression is represented by Regular Expressions in the proposed enhanced ICPMD patterns database. Each Regex pattern may consist of zero or more parameters. The values of these parameters are used to substitute in the equivalent source code generation for different platforms. There are three types of code patterns as follows:

1. *Simple Pattern*: the simple pattern targets to match single source code statement that represents the following statements:
   - Event handler signature (e.g., click event handler)
   - Assignment statement (e.g., set text of textbox control)
   - Method invocation statement (e.g., open another application page)
   - Basic statements in any programming languages such as:
     - Variable declaration statement
     - if statement
     - for statement
     - while statement
     - Others

   The simple pattern Regex may consist of zero or more parameters: @PARAM1, @PARAM2, etc. The values of these parameters are used while code is being generated to the target platforms. Example 2 explains the structure of a simple code pattern and how it matches the input source code line.

   Example 2. simple pattern:
   - Input WP8 statement from the input source code file:
     textStatus.Text = "Tap the camera button to take a picture.";
   - Input code pattern from the ICPMD database:
     ([\w]*)[\.]Text[\s]* = [\s]*(.*)[\s]*;
   - Stored Android statement in the ICPMD database:
     @PARAM1.setText(@PARAM2);
   - Output Android statement after matching the pattern:
     textStatus.setText("Tap the camera button to take a picture.");
2. *Composite Pattern*: the composite pattern matches two or more lines of the source code. These patterns target groups of source code statements that represent mobile functions/features such as:
   - Open camera
   - Get results from camera
   - Make a phone call
   - Send SMS
   - Create navigation menu
   - Play sound
   - Get Bluetooth devices list
   - Open URL in webview control
   - Others

   The composite pattern consists of two or more Regex patterns. Each Regex pattern consists of zero or more parameters: @PARAM1, @PARAM2, etc. In addition, there are shared parameters among all the statements of the same composite pattern: @SHARE1, @SHARE2, etc. The same composite pattern could be repeated several times in the input source code file. Therefore, it is mandatory to share the object name as a shared parameter. The values of the parameters (@PARAM or @SHARE) are then used to substitute the corresponding parameters in the generated source code of the target platforms. Example 3 explains the structure of a composite code pattern.

   Example 3. composite pattern:
   - Input WP8 statements from the input C# source code file:
     VibrateController vibrate = VibrateController.Default;
     vibrate.Start(TimeSpan.FromMilliseconds(1,000));
     vibrate.Stop();
   - Input code patterns from the ICPMD database:
     VibrateController[\s]*([\w]*)[\s]* = [\s]*VibrateController.Default[\s]*;
     @SHARE1.Start[\(][\s]*TimeSpan.FromMilliseconds[\(][\s]*([\w]*)[\)][\s]*[\)][\s]*;

@SHARE1.Stop[\(][\s]*[\)][\s]*;
- Stored Android statements in the ICPMD database:
  Vibrator @SHARE1 = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
  @SHARE1.vibrate(@PARAM1);
  @SHARE1.cancel();
- Output Android statement:
  Vibrator vibrate = (Vibrator) getSystemService (Context.VIBRATOR_SERVICE);
  vibrate.vibrate(1,000);
  vibrate.cancel();

3. *Expression pattern*: This pattern matches part of line in the source code. These patterns target the expressions used in statements such as:
   - Get substring of text
   - Get length of text
   - Get text of textbox control
   - Others

The expression pattern Regex may consist of zero or more parameters: @PARAM1, @PARAM2, etc. The values of these parameters are used to substitute in the generated source code of the target platform. The next section will explain in more details the SCM algorithm.

5.3 Source Code Migration Algorithm

The aim of the SCM algorithm is to convert the input source code written in one programming language (e.g., C#) to the equivalent source code of another programming language (e.g., Java). The main idea of the SCM algorithm is to search in the input source code file for matching a predefined set of code patterns, and then get the equivalent source code for the target platforms from the database. These patterns are written using Regular Expressions. The SCM algorithm uses different types of code patterns: Simple patterns, Composite patterns, and Expression patterns. The code patterns are stored in the ICPMD patterns database where each record consists of the pattern for the input programming language along with the equivalent lines of code for each target platform programming language.

The SCM algorithm consists of four main stages as shown in Fig. 4, these stages are:

1. *Code preparation*: this stage parses all the lines of code in the input source code file to remove any ambiguities that could hinder the detection of the code patterns. In most cases, two or more types of code statements could exist on the same line of source code which require to be separated into different lines like the following cases:
   - Source code statement and comment statement on the same line.
   - Start of block "{" or end of block "}" and source code statement or comment statement on the same line.
   - Many source code statements separated by ";" on the same line.
2. *Code survey*: this stage parses all the source code lines, in the updated source code file after the code preparation stage, to mark all the statements that match composite patterns to prevent parsing them as simple patterns in the code conversion stage of this algorithm.
3. *Code conversion*: this stage parses and translates all the source code lines, in the updated source code file, to the intermediate abstract model XML representation which contains translated code for all target platforms. The main idea is to compare each line in the input source code with the predefined patterns, then compose the output XML representation file as follows:
   - If line is matched with one of the code patterns:
     - Add a new element <comment> contains the recognized source code line.
     - Add a new element <pattern>:
       - Add inner element for each supported platform (i.e., <android> and <ios>) contain the equivalent source code for the recognized/matched source code line.
   - Else, if line is not matched with any code pattern:
     - Add a new element <comment> that contains the unrecognized source code line.
4. *Code generation*: this stage uses the language XSLT file to convert the intermediate XML representation file to the source code file of the target platform.

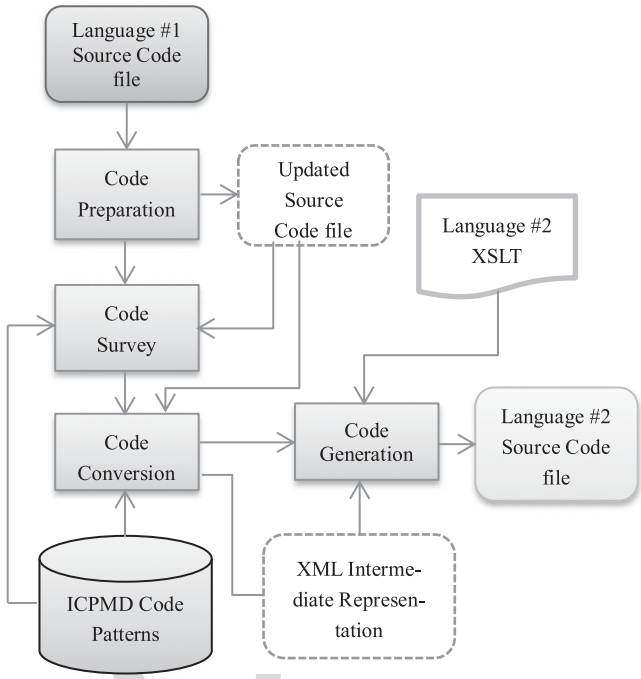

Fig. 4. Stages of the source code migration algorithm.

The proposed code conversion algorithm handles the differences between the current supported platforms (Android and WP8) on the source-code-line level. Our approach provides the following capabilities:

1. *Handling of unsupported statements in the target platform*: in the database, the source pattern is stored along with a separate equivalent source code field for each supported target platform. Therefore, if a target platform does not support this pattern; the field that corresponds to the target platform will be

TABLE 2
Supported Code Functions

	Supported Function	Details
1	Initialize the application page class	Define references to the UI controls
2	Handling the mobile application page life cycle events	Create/Start – Pause – Resume
3	Event Handlers	Click event handler - Touch event handler - Menu Item Click event handler - Sensors/Count down timer event handlers
4	Isolated storage in the mobile App (shared preferences in Android)	Store key-value pair Retrieve key-value pair Delete key-value pair
5	Application Menu	Menu of each application page
6	Built-in Camera	Open Built-in Camera - Retrieve the result from Built-in Camera
7	Page Navigation	Open another application pages
8	Messaging	Send SMS - Send Email
9	Webview Functions	Load URL
10	Vibration	Start – Stop
11	Countdown timer	Start – Stop
12	Sensors	Start and stop Accelerometer Start and stop Orientation/Motion
13	Listbox Functions	Fill list - Add item - Get item - Remove item
14	Bluetooth Functions	Is Bluetooth enabled? Get paired devices list
15	Device Information	Manufacturer - OS version - Device Name

filled with a comment explaining that this statement is not supported yet in this particular platform. In the case of patterns that are not found in the database, the undetected lines of code are converted as comments in the target application. This design guarantees that if some lines of code are specific to the source platform and not supported on the target platform, then their conversion issue will be solved implicitly.

2. *Mapping the differences between the platforms*: in the database, some code patterns are defined to be moved to certain methods/event handlers during the code conversion process. Reading the current changed values of sensors is a good example of this capability. In WP8, each sensor can register a separate event handler to read this sensor values. In Android, on the other side, the application page "Activity" must implement the interface "SensorEventListener" and override two methods "onSensorChanged" and "onAccuracyChanged" to read the current changed values of all sensors. So, any pattern related to reading values of sensors will be moved to these two particular methods.

Handling more source-code-line differences between the different supported platforms will be extended in the future work.

In the new enhanced ICPMD: the AppModeler implements the first three stages of the SCM algorithm and the AppProducer implements the fourth stage. Although this algorithm is designed mainly to serve the ICPMD solution, it could be adapted for different contexts.

5.4 Supported Functions and Code Conversion Examples

The code converters in the AppModeler and the AppProducer can detect and convert the code functions listed in Table 2. The list of supported features/functions could be extended by adding the Regex patterns in the ICPMD database so the code converters can detect and convert more Apps.

By using the proposed code conversion approach, which is implemented in the new enhanced ICPMD solution, several mobile Apps have been converted from WP8 platform to Android platform and vice versa. The generated Android/WP8 projects may require some minor updates to be done by the developer manually. These manual updates include:

- Fixing any compilation errors.
- Writing the corresponding target code for the undetected code functions that are converted as comments.

To illustrate how the SCM algorithm works, Fig. 5 shows how a line in the input source code file is matched with the different code patterns in the ICPMD database. Then, the parameters of the pattern are extracted and used to generate the intermediate XML representation. Table 3 shows a simple mobile application that uses the Isolation Storage feature to save data for a particular key, where the C# source code is converted to the intermediate XML representation, and then the Java code is generated. Appendix A, which can be found on the Computer Society Digital Library at http://doi.10.1109/TSE.2016.2543223, at the end of this paper includes the XSD schema of the XML intermediate language which is used in the proposed code conversion approach. Table 4 shows another code conversion example, in which the C# code for Windows Phone 8 is obtained from the Java code for Android for an application that uses a web view control to display a particular web page.

5.5 Design Assessment Discussion

The efficiency of the enhanced ICPMD solution depends mainly on the quality and quantity of the source code patterns that are used in the code conversion between the source and the target platforms. In order to assess the applicability of the proposed approach in the general case, several concerns should be discussed. For instance, the effort required to define the patterns is proportional to the number of patterns and to the number of source languages. Defining the patterns is simpler now than using the XPath technique used in the original ICPMD. The maintenance of the patterns and their target codes will depend on the frequency of having new versions. Therefore, this can be solved by having agreements with the platform vendors that enable them to update the database when they release new versions.

Fixing the code of an App that is unsuccessfully converted has a reasonable user-friendliness degree as compared to writing the App from scratch because the undetected lines of code are converted as comments. This

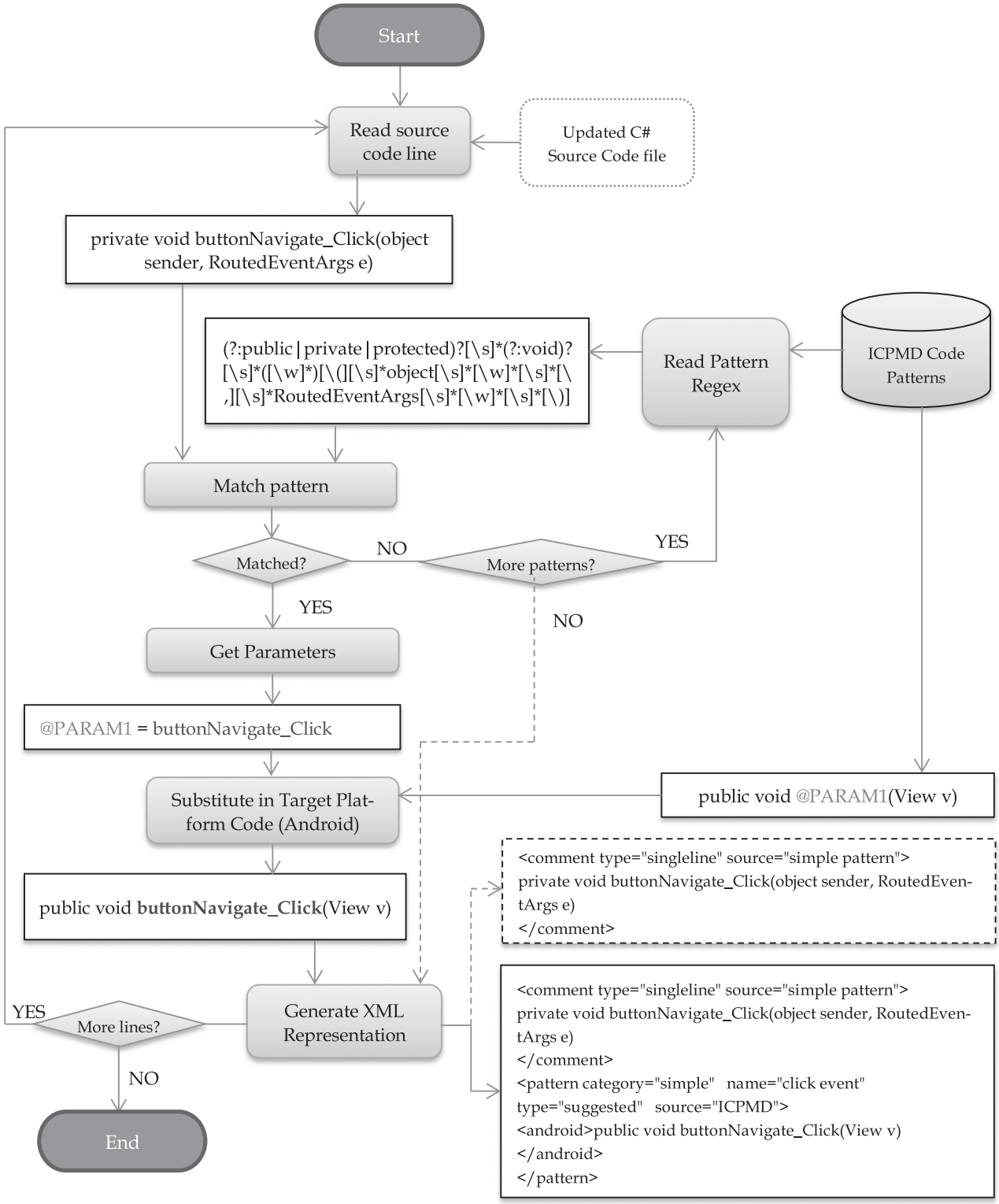

Fig. 5. Convert C# source code to the intermediate XML representation.

commented code will be an initial hint to the developer which helps him to implement the required functionality.

The management process of defining and maintaining the source code patterns is formalized by answering the following questions:

1. *How to categorize the source code patterns?*
   The mobile platforms provide almost the same mobile features but with different implementations. Each mobile feature is a pattern (e.g., using the camera), each pattern consists of a set of sub-patterns (e.g., take a photo, and record a video), and each sub-pattern consists of one or more statements.
2. *Who can define the source code patterns?*
   The platform-specific developers can contribute and define the code patterns for one or more mobile

TABLE 3
Source Code Conversion Example from C# to Java

Input C# Source Code (WP8 Platform)	Intermediate XML Representation	Output Java Source Code (Android Platform)
….. public partial class MainPage : PhoneApplicationPage { // Constructor public MainPage() { InitializeComponent(); } private void Save_Click(object sender, RoutedEventArgs e) { IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings; if (!settings.Contains("DataKey")) { settings.Add("DataKey", DataToSave.Text); Result.Text = "Data saved ! "; } else { settings["DataKey"] = DataToSave.Text; Result.Text = "Data saved ! "; } settings.Save(); } …..	….. <pageclass type="suggested" source="ICPMD"> <param name="name" value="MainPage" /> <param name="parent" value="PhoneApplicationPage" /> </pageclass> <block_start block="pageclass"/> <comment type="singleline" source="original comment"> // Constructor </comment> <comment type="singleline" source="original"> public MainPage() </comment> <constructor type="suggested" source="ICPMD"> <param name="modifier" value="public" /> <param name="output" value="" /> <param name="name" value="MainPage" /> </constructor> <comment type="singleline" source="simple pattern"> private void Save_Click(object sender, RoutedEventArgs e) </comment> <pattern category="simple" name="click event" type="suggested" source="ICPMD"> <android>public void Save_Click(View v)</android> </pattern> <block_start block=""/> <comment type="singleline" source="Composite pattern step 1 out of total steps 6"> IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings; </comment> ……..	….. public class MainPage extends Activity { //original comment: // Constructor EditText DataToSave; Button Save, Retrive, Delete; TextView Result; @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.mainpage); DataToSave = (EditText) findViewById(R.id.DataToSave); Save = (Button) findViewById(R.id.Save); Retrive = (Button) findViewById(R.id.Retrive); Delete = (Button) findViewById(R.id.Delete); Result = (TextView) findViewById(R.id.Result); } //simple pattern: private void Save_Click(object sender, RoutedEventArgs e) public void Save_Click(View v) { //Composite pattern step 1 out of total steps 6: IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings; SharedPreferences settings = getPreferences(Context.MODE_PRIVATE); SharedPreferences.Editor editor = settings.edit(); if(!settings.contains("DataKey")) { //Composite pattern step 2 out of total steps 6: settings.Add("DataKey", DataToSave.Text); editor.putString("DataKey", DataToSave.getText().toString()); editor.commit(); …….

functions. The efforts required to define the code patterns depends mainly on the number of platform-specific contributors and their level of experience.

3. *How to maintain/validate the source code patterns?* The senior mobile developers can approve these patterns before using them during the code conversion process. Senior developers can see all code patterns that belong to the same mobile function. Therefore, they can remove

TABLE 4
Source Code Conversion Example from JAVA to C#

Input Java Source Code (Android Platform)	Intermediate XML Representation	Output C# Source Code (WP8 Platform)
… public class WebViewActivity extends Activity { /** Called when the activity is first created. */ @Override public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main); WebView wv = (WebView) findViewById(R.id.webview1); WebSettings webSettings = wv.getSettings(); webSettings.setBuiltInZoomControls(true); wv.loadUrl("file:///android_asset/Index.html"); } } …	… <comment type="singleline" source="original"> public class WebViewActivity extends Activity </comment> <pageclass type="suggested" source="ICPMD"> <param name="name" value="WebViewActivity" /> <param name="parent" value="" /> <param name="projectname" value="WebView" /> </pageclass> <block_start block="pageclass"/> <comment type="multiline" source="original comment"> /** Called when the activity is first created. */</comment> <comment type="singleline" source="original"> @Override public void onCreate(Bundle savedInstanceState) </comment> <constructor type="suggested" source="ICPMD"> … <comment type="singleline" source="Composite pattern step 1 out of total steps 1"> wv.loadUrl("file:///android_asset/Index.html"); </comment> <pattern category="composite" name="Webview" type="suggested" source="ICPMD"> <wp>wv.Navigate(new Uri("android_asset/Index.html", UriKind.RelativeOrAbsolute));</wp></pattern> <block_end block="constructor"/> … …	… public partial class WebViewActivity : PhoneApplicationPage { //original comment: /** Called when the activity is first created. */ //original: @Override public void onCreate(Bundle savedInstanceState) public WebViewActivity() { InitializeComponent(); //suggested comment : super.onCreate(savedInstanceState); //suggested comment : setContentView(R.layout.main); //suggested comment : WebView wv = (WebView) findViewById(R.id.webview1); //suggested comment : WebSettings webSettings = wv.getSettings(); //suggested comment : webSettings.setBuiltInZoomControls(true); //Composite pattern step 1 out of total steps 1: wv.loadUrl("file:///android_asset/Index.html"); wv.Navigate(new Uri("android_asset/Index.html", UriKind.RelativeOrAbsolute)); } } …

redundant, duplicate, incomplete, obsolete code patterns. In addition, the senior developers have the responsibility to avoid the possibility of having false-positive conversions due to any overlapping code patterns.

The next section sheds the light on the results that are obtained from the new enhanced ICPMD and how the generated applications can be evaluated.

6 Proposed Evaluation Criteria and Results

There are two requirements of evaluation. The first evaluation requirement is to evaluate the efficacy and efficiency of the proposed code conversion approach. This evaluation focuses on the ability to detect and convert different simple mobile functions (e.g., send SMS). Therefore, the Apps that are used in the evaluation are selected to reflect various mobile functions. In addition, it was preferred to select simple mobile applications that focus mainly on one mobile function to evaluate each code pattern and its corresponding function separately in an independent way on other patterns and functions.

The second requirement is motivated by the lack of benchmark to compare the performance of the Apps that are generated from the different cross-platform mobile applications development tools. In this comparison, we adopted the benchmark used in [15], [36], [37] to compare the App generated from ICPMD with Apps generated by other alternatives or developed natively. This evaluation focuses on the performance of the generated App with its native counterpart and in comparison to the Apps developed using Titanium and Xamarin. This is done by developing particular benchmark Apps that use measurements.

The next sub-sections introduce the proposed criteria to evaluate the efficiency of the proposed code conversion approach and the evaluation results, and the results of comparing the performance of the ICPMD generated App with its native counterpart and the Apps developed using Titanium and Xamarin.

6.1 Evaluation of the Proposed Code Conversion Approach

The evaluation criteria steps, to evaluate the new enhanced ICPMD and the proposed code conversion approach, can be explained as follows:

1) Evaluate the enhanced ICPMD by computing the following measurements:
   - *Input LOC*: the Lines of Code (LOC) of the source App.
   - *Generated LOC*: the lines of code of the generated App.
   - *Generation Time*: the time taken by the ICPMD to convert the source App to the generated App.
   - *Compilation Errors*: the count of compilation errors when compiling the generated App for the first time. Besides, the different types of compilation errors are defined.
2) Compute the Correctly Detected Lines of Code (CDLOC) percentage in comparison to the total lines of code, Accuracy, Precision and Recall for the generated applications to evaluate the efficiency of the proposed code conversion approach: these computations are defined in equations (1), (2), (3), and (4); the last three computations are used in [38].

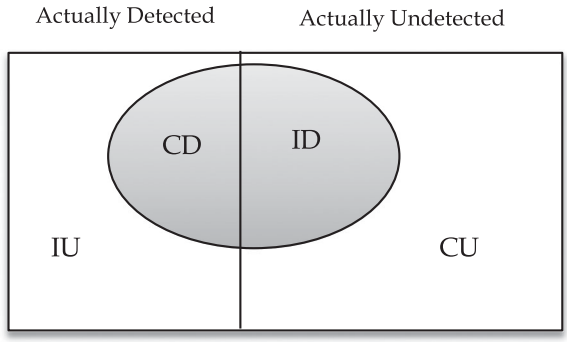

Fig. 6. Demonstration of the evaluation parameters.

$$CDLOC = \frac{true\ positives}{true\ positives + false\ positives + false\ negatives + true\ negatives}, \quad (1)$$

$$Accuracy = \frac{true\ positives + true\ negatives}{true\ positives + false\ positives + false\ negatives + true\ negatives}, \quad (2)$$

$$Precision = \frac{true\ positives}{true\ positives + false\ positives}, \quad (3)$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}. \quad (4)$$

The main idea is to manually check the generated Apps from the ICPMD solution to evaluate if the converted source code lines are correctly detected or not. The equations' parameters to evaluate the ICPMD are demonstrated in Fig. 6 and are defined as follows:

- True positive: the Correctly Detected (CD) lines of code.
- False positive: the Incorrectly Detected (ID) lines of code.
- True negative: the Correctly Undetected (CU) lines of code.
- False negative: the Incorrectly Undetected (IU) lines of code.
- The total number of input source code lines that should be Converted (C) because they have equivalent code patterns in the database, is computed as follows:

$$\mathrm{C} = \mathrm{CD} + \mathrm{ID}.$$

- The total number of input source code lines that should be Unconverted (U) because they do not

TABLE 5
Windows Phone 8 Samples List

	WP8 App Name	Sample URL
1	UsingIsolatedStorage	https://code.msdn.microsoft.com/windowsapps/Using-Isolated-Storage-fd7a4233
2	Mini-browser Sample	https://code.msdn.microsoft.com/windowsapps/Mini-browser-Sample-b5587d2b
3	Basic Camera Sample	Developed App
4	How to make your Windows Phone vibrate using C#	https://code.msdn.microsoft.com/windowsapps/How-to-make-your-Windows-c40d80a3
5	Get X and Y coordinates of user touch	https://code.msdn.microsoft.com/windowsapps/WindowsPhone-Get-X-and-Y-88c5948a
6	Windows Phone 8 SMS Compose Task	https://code.msdn.microsoft.com/windowsapps/Windows-Phone-8-SMS-6936c2fd
7	Application Bar Sample	https://code.msdn.microsoft.com/windowsapps/Application-Bar-Sample-For-8e990f04
8	Simple Motion Sample	https://code.msdn.microsoft.com/windowsapps/Simple-Motion-Sample-101d1027
9	Device Status Sample	https://code.msdn.microsoft.com/windowsapps/Device-Status-Sample-2ab5df82
10	sdkPhotosCS	https://code.msdn.microsoft.com/windowsapps/Photos-Sample-a38a2c8e/view/SourceCode

have equivalent code patterns in the database, is computed as follows:

$$U = CU + IU.$$

The equations (5), (6), (7), and (8) can be rewritten as follows:

$$CDLOC = CD/(C + U), \tag{5}$$

$$Accuracy = (CD + CU)/(C + U), \tag{6}$$

$$Precision = CD/C, \tag{7}$$

$$Recall = CD/(CD + IU). \tag{8}$$

The evaluation of the enhanced ICPMD solution and the proposed code conversion approach has been performed using the sample WP8 Apps which are listed in Table 5. The evaluation of the proposed code conversion approach in the reverse direction (from Android to WP8) will be included in the future work.

By using the above explained evaluation criteria, the evaluation results of sample Apps converted from WP8 platform to the equivalent Android Apps are listed in Tables 6, 7 and 8 respectively. The evaluation results of the ICPMD generated Apps show the following:

- The generation time is relatively small to generate a full simple mobile application in less than one minute.
- The generated lines of code "Generated LOC" exceeds the input lines of code "Input LOC" in Table 5 because of the following reasons:
  - If the input source code line is detected, the output file will include two lines as follows:
    - A copy of the input source code line as comment for the developer to check on the converted source code.
    - The equivalent source code in the target platform.
  - If the input source code line is not detected, the output file will include:
    - A copy of the input source code line, marked as a comment in order to enable the target language developer to write the equivalent source code by himself.
- The errors in detecting the code patterns (ID column) are relatively small compared to the correctly detected code patterns (CD column). This is clearly explained in the Accuracy and Precision columns in Table 7.
- The compilation errors are relatively small in Apps with small LOC. In Apps with more LOC like Apps

TABLE 6
Evaluation of the Enhanced ICPMD

	App Name	Application Pages	Input LOC	Generated LOC	Generation Time (Seconds)
1	UsingIsolatedStorage	1	64	162	7.4
2	Mini-browser Sample	1	39	108	2.6
3	Basic Camera Sample	1	48	121	3.6
4	Vibration	1	56	113	3.8
5	X and Y coordinates of Touch	1	31	90	3.7
6	Send SMS	1	48	119	3.4
7	ApplicationBar	4	551	1,136	43.0
8	Simple Motion Sample	1	93	199	6.0
9	Device Status Sample	1	65	187	5.9
10	sdkPhotosCS	2	601	844	22.2

TABLE 7
Accuracy, Precision and Recall of the Generated Applications

	App Name	C	CD	ID	U	CU	IU	Accuracy	Precision	Recall	CDLOC
1	UsingIsolatedStorage	64	62	2	0	0	0	97%	97%	100%	97%
2	Mini-browser Sample	39	39	0	0	0	0	100%	100%	100%	100%
3	Basic Camera Sample	48	48	0	0	0	0	100%	100%	100%	100%
4	Vibration	56	54	2	0	0	0	96%	96%	100%	96%
5	X and Y coordinates of Touch	29	29	0	2	2	0	100%	100%	100%	94%
6	Send SMS	48	47	1	0	0	0	98%	98%	100%	98%
7	ApplicationBar	515	458	57	36	36	0	90%	89%	100%	83%
8	Simple Motion Sample	83	77	6	10	10	0	94%	93%	100%	83%
9	Device Status Sample	61	58	3	4	4	0	95%	95%	100%	89%
10	sdkPhotosCS	505	489	16	96	96	0	97%	97%	100%	81%

7 and 10, there are more compilation errors and more undetected lines of code because these Apps have some unsupported functions. For example, App 10 allows the user to use two main functionalities: 1) using the camera to capture an image, and 2) cropping the captured image to produce a new cropped image. The ICPMD contains patterns for the camera usage function while there are no patterns for the image cropping function. Therefore, the camera usage function has been detected and converted correctly while the undetected lines of code are converted as comments. These comments can be rewritten later with the aid of platform-specific developers. Although the new proposed ICPMD database does not cover the entire code patterns and the applications could not be completely converted, the mobile developer will benefit from the saved time and efforts.

- Whenever there are new code patterns that will be added to the ICPMD patterns database, the detected and converted source code lines will increase and the values of the columns U and CU in Table 7 will decrease.

Table 8 shows the compilation errors details of the generated Android Apps. The number of compilation errors is small and the errors can be categorized as follows:

- *Partial detection*: this error occurs when a simple or a composite pattern has been detected by ICPMD, and part of the statement (expression pattern) has not been detected. This kind of error can be mitigated by adding more expression patterns to the ICPMD database.
- *Variable scope/definition*: this error occurs when there is a problem related to the following:
  - Scope of variables.
  - Variable assignment value.
  - Variable definition is not detected correctly.

  This type of errors can be solved easily be the developers due to the syntax similarity of C# and Java.
- *Unsupported feature*: this error occurs when the pattern has been detected and converted correctly but the generated statement uses a feature that is not supported in the target platform. For example, the switch statement in WP8 can use a string variable but it is not supported in Android. Switch on string is supported starting from Java 1.7 while Android is based on Java 1.6.
- *Require code*: this error occurs when the pattern has been detected and converted correctly but the target platform requires extra code to be written. For example, an event handler statement in WP8 return void is converted to the equivalent statement in Android which return Boolean; in this case the developer has to add "return statement".
- *Extra/missed bracket*: this error occurs when an extra bracket exists in the generated source code. Also, the same error applies to missed brackets.

TABLE 8
Compilation Errors of the Generated Apps

	App Name	Compilation Errors	Error Type					Other Errors/Notes
			Partial Detection	Variable Scope/Define	Unsupported Feature	Require Code	Extra/Missed Bracket	
1	UsingIsolatedStorage	2	0	2	0	0	0	Variable scope
2	Mini-browser Sample	0	0	0	0	0	0	-
3	Basic Camera Sample	0	0	0	0	0	0	-
4	Vibration	2	0	1	0	0	1	Variable assignment
5	X and Y coordinates of Touch	1	0	0	0	1	0	Return statement
6	Send SMS	1	0	0	0	0	1	-
7	ApplicationBar	18	0	0	18	0	0	Switch on string
8	Simple Motion Sample	6	2	0	1	2	1	-
9	Device Status Sample	3	3	0	0	0	0	Missed property in UI
10	sdkPhotosCS	18	2	9	7	0	0	Resource name problem

TABLE 9
Compilation Errors of the Generated Apps

	ICPMD	Titanium	Xamarin
Cross-Platform Mobile Apps Development Approach	Merged approach (Trans-Compilation and Model-Based Development)	Runtime Interpretation approach	Different approach for each supported platform
Supported Platforms	Android and WP	Android, iOS, Blackberry, Tizen and WP	Android, iOS and WP
Programming Language	C# or Java	Javascript	C#
User Interface	Reused and converted	Written programmatically	Written for each platform separately
Output	Ready-to-compile App	Ready-to-publish App	Ready-to-publish App

- There are other kinds of errors that will be handled in the next version of ICPMD, including:
  - The resource name is not supported. For example, the image name "appbar.check.rest.png" could be used in a WP8 App while it is not acceptable to be used in an Android App.
  - Some user interface controls (e.g., the slider and video controls) and their properties are not supported yet by ICPMD. Therefore, they are not converted to the target platform App.

6.2 Comparison of the Enhanced ICPMD with Other Alternatives

It is a difficult task to select a comparison criteria of the Apps generated from different cross-platform mobile development solutions. We adopted the benchmark measurements that are described in papers [15], [36], [37]. This section compares the performance of the native Apps and the Apps that are generated from ICPMD and the other cross-platform mobile development solutions.

In this research, the ICPMD has been compared to the following cross-platform solutions:

- *Titanium* which is a widely used cross-platform tool that is based on Javascript development.
- *Xamarin* which is a commercial tool that is based on C# development.

Titanium and Xamarin have been selected because they are the most widely used solutions [25], [26] that produce full native Apps that include both UI and source code. Therefore, they are comparable to the proposed ICPMD solution which supports native Apps and generate both UI and code. Phonegap is not selected because it supports hybrid Apps. Other tools are not selected for comparison because they were not available in a stable widely-used version. Table 9 shows a comparison between ICPMD, Titanium, and Xamarin.

The main target of this benchmark is to perform performance comparisons between the cross-platform developed Apps using Titanium and Xamarin, the native developed App, and the ICPMD generated App. The benchmark for comparing the Apps is based on the following measurements and experiments:

1) *Application size*: the size of the App after installing it to a real mobile device.
2) *Application memory usage*: the RAM used by the running App. This value is acquired with the "Task Manager" App in the Android platform. This measurement could not be measured in the WP8 platform.
3) *Data manipulation performance*: implement the function "Array Sorting" for the selected cross-platform solutions to sort an array with 1,000 items, then measure the elapsed time to complete this task (in milliseconds) to measure the data manipulation performance among the different Apps.
4) *User interface rendering performance*: implement the function "UI Rendering" for the selected cross-platform solutions to draw 1,000 buttons on the application page, then measure the elapsed time to complete this task (in milliseconds) to measure the user interface rendering performance among the different Apps.

Five benchmark mobile applications have been implemented to have simple user interfaces and perform the same benchmark functions (Array Sorting and UI Rendering). The benchmark applications are developed as follows:

- *App1*: a native WP8 application that was developed then compiled using the Visual Studio 2012. This App has been installed in a Lumia device. The obtained measurements were: the size of this App is 44.5 KB, the time of the array sorting function is 57 milliseconds, and the time of the UI rendering function is 376 milliseconds.
- *App 2*: a native Android App that was generated by using the ICPMD to convert App 1, the WP8 application project, and then compiled by using the Android Development Tool (ADT).
- *App 3*: the corresponding native Android application of App1. It was developed from scratch and then compiled by using the ADT.
- *App 4*: an application that was developed using Javascript in Titanium. Then it was compiled for Android.
- *App 5*: an application that was developed using C# in Xamarin. Then it was compiled for Android.

Benchmarking was performed using two mobile devices with the following specifications:

1) Nokia Lumia 625: (Processor type: Dual-core 1.2GHz, Mass memory: 8 GB and RAM: 512 MB, and run Windows Phone 8.0).
2) Samsung Galaxy Grand Neo (GT-19060): (Processor type: Quad-core 1.2GHz, Mass memory: 8 GB and RAM: 1 GB, and run Android version 4.2.2).

All the benchmark results are shown in Table 10 and Fig. 7. Fig. 7(a) depicts the application size after installing it in the mobile device. It can be seen that the ICPMD generated App has the smallest application size. The ICPMD generated application size is about 96.9 percent smaller than the native application size. The Titanium App has the worst application

TABLE 10
Results of Comparing ICPMD with Other Cross-Platform Solutions

Benchmark Application	Application Size (KB)	Memory Usage (MB)	Array Sorting (Milliseconds)	UI Rendering (Milliseconds)	Run on Device
Android App (output from ICPMD)	32	10.80	55	1,140	2
Native Android App	1034.24 (1.01 MB)	10.43	60	1,079	2
Titanium App	11939.84 (11.66 MB)	14.04	116	5,341	2
Xamarin App	3747.84 (3.66 MB)	31.46	53	15,88	2

size, which is about 1,054 percent larger than the native App application size. Titanium App has a bigger application size because the output App contains a JavaScript interpreter runtime and a Webkit rendering engine [18]. ICPMD converts the UI, code, manifest, and resource files from WP8 project and generates a native ready-to-compile Android project. It is worth mentioning that the WP8 *platform-specific files* are not converted to the target platforms. The application size of ICPMD is smaller than the native one because the native App contains more libraries and is generated with icons for the different mobile sizes. On the other side, ICPMD moves the application icon that exists in the source WP8 App to the target Android ready-to-compile App.

Fig. 7(b) depicts the memory usage (required RAM) of the benchmark applications. It can be seen that the ICPMD generated App memory usage is very close to the native App memory usage. The Titanium App has a moderate memory usage, which is about 34.6 percent larger than the native application size. The Xamarin App has the worst memory usage, which is about 201.6 percent larger than the native App memory usage. Xamarin uses more memory than others because its garbage collector is not effecient [39].

Fig. 7(c) depicts the results of the "Array Sorting" function of the benchmark applications. It can be seen that the ICPMD generated App and Xamarin App measurements are very close to the native App measurement. The Titanium

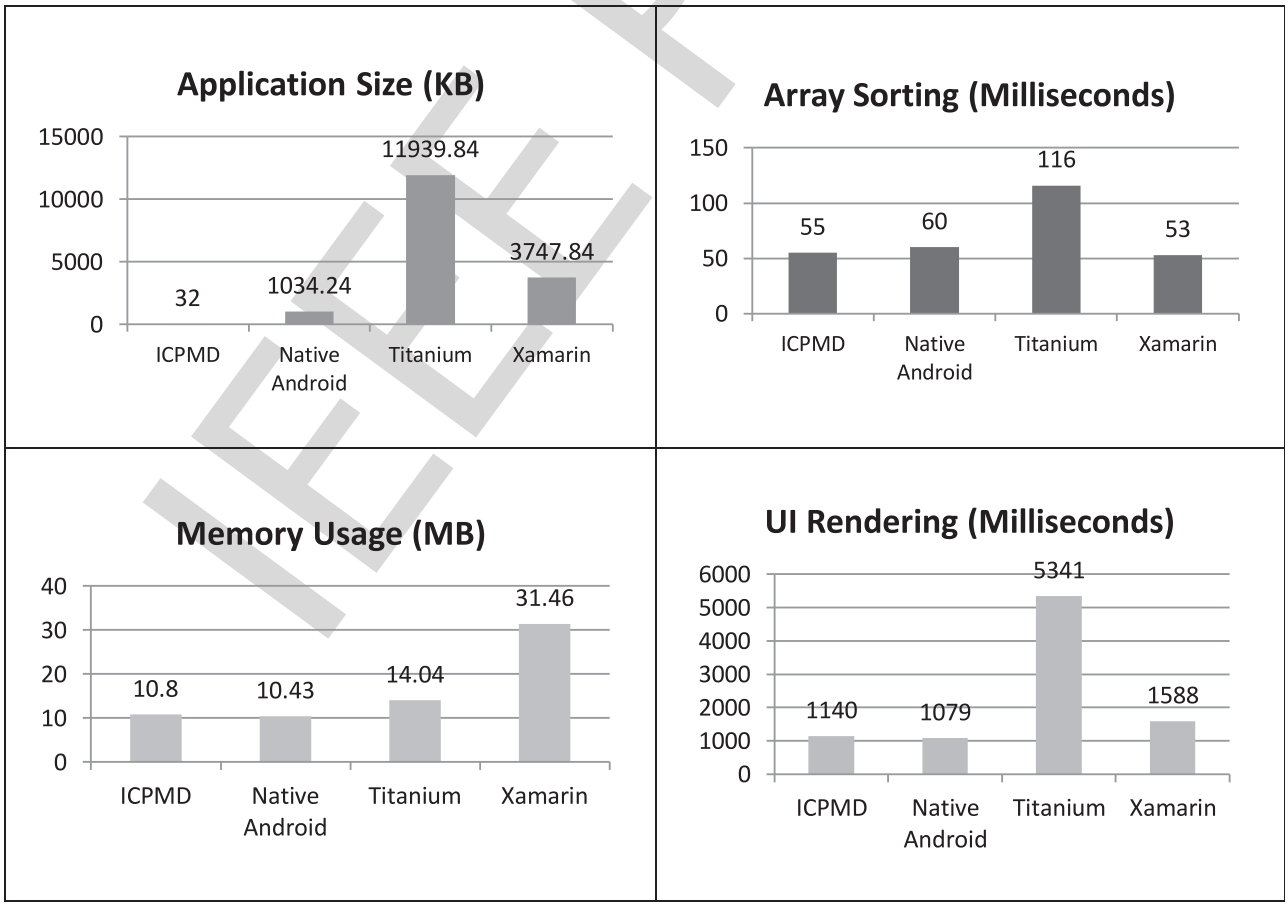

Fig. 7. Results of comparing ICPMD with other cross-platform solutions: (a) Application size, (b) Memory usage, (c) Array sorting, and (d) UI rendering.

App has the worst measurement which is about 93.3 percent larger than the native application size. Titanium takes more time to perform the array sorting because of its approach (Runtime Interpretation). The interpretation provides less performance than the native compilation.

Fig. 7(d) depicts the results of the "UI Rendering" function of the benchmark applications. It can be seen that the ICPMD generated App measurement is very close to the native App measurement. The Xamarin App measurement is about 47.2 percent larger than the native App measurement. The Titanium App has the worst measurement which is about 395 percent larger than the native application size. Titanium takes more time to perform the UI rendering because the UI is written programmatically and the titanium approach (Runtime Interpretation).

For the simple benchmark application, these application measurements could be tolerable especially when these applications are executed on high-specifications smartphones. However, for more complex applications, the increased application size, memory usage, or performance measurements could quickly pose problems. The results show reasonable improvement of the applications that are generated from the new enhanced ICPMD over the other existing cross-platform mobile development solutions.

7 Conclusion and Future Work

The new enhanced ICPMD solution overcomes some limitations of the existing cross-platform mobile development solutions by: helping the developer to develop with the most popular programming languages like Java for Android and C# for Windows Phone, focusing on both the source code and user interface transformations to generate full applications, and supporting code reuse. This solution has been implemented to convert the source project (WP8 App project) to the target platform project (Android App project) and vice versa. The new code conversion approach has been implemented for the ICPMD by using the XSLT and Regular Expressions.

An algorithm (SCM algorithm) is proposed to apply the proposed code conversion approach by searching in the input source code file to match some predefined sets of code patterns, and then get the equivalent source code for the target platform from the database. The code patterns are written using Regular Expressions. The proposed algorithm was applied to convert the C# source code written for WP8 platform to the equivalent Java code for Android platform and vice versa. The conversion is based on an extendable code patterns database.

The ICPMD has been compared with stable and widely used native development tools like Titanium and Xamarin. The evaluation results show reasonable improvements in the speed, memory usage, and application size of the new enhanced ICPMD generated Apps compared with the applications generated from other cross-platform tools like Titanium and Xamarin. The results obtained are very comparable to the results of the native Apps.

The ICPMD was designed to achieve the following requirements:

1. Focus on the native mobile development.
2. Focus on generating full Apps (ready-to-compile).
3. Support code reuse.
4. The developer does not have to learn a new programming language to use this solution.

The first, second, and third requirements have been achieved successfully. However, the fourth requirement is not completely achieved yet. Although the ICPMD minimizes the development efforts by generating the target platform App, there are two cases that require manual updates:

1. Unconverted lines of code.
2. Compilation error due to error in the pattern detection.

The first case could be minimized by adding more patterns in the ICPMD database by encouraging volunteer developers to contribute and add source code patterns. The second case could be solved by hiring a platform-specific developer to fix the compilation errors.

The current version of the ICPMD solution is available under the "GNU GPLV3" license in the URL: (https://github.com/wafaa-elkassas/ICPMD). The complete system with approval workflow that enables mobile developers to add/update/search/approve code patterns will be available for online use later. The future work includes:

1. Extending the list of the supported features by adding more Regex patterns in the ICPMD database. This could be achieved by encouraging volunteer developers to contribute and add source code patterns.
2. Evaluating the proposed code conversion approach in the reverse direction from Android to WP8.
3. Using more complex mobile applications to test the ICPMD in a more production-like setting (from WP8 to Android and vice versa).
4. Handling more source-code-line differences between the different supported platforms.
5. Supporting new platforms like iOS and Blackberry.

Acknowledgments

Wafaa Samy El-Kassas is a corresponding author.

References

[1] B. B. Akinkuolie, L. Chia-Feng, and Y. Shyan-Ming, "A cross-platform mobile learning system using QT SDK framework," in *Proc. 5th Int. Conf. Genetic Evol. Comput.*, 2011, pp. 163–167.

[2] H. F. ElYamany and A. H. Yousef, "A mobile-quiz application in Egypt," in *Proc. 4th IEEE Int. E Learn. Conf.*, 2013, pp. 325–329.

[3] T. Kim, B. Kim, and J. Kim, "Development of a lever learning webapp for an HTML5-based cross-platform," in *Proc. Multimedia Ubiquitous Eng.*, 2013, vol. 240, pp. 313–320.

[4] W. El-Kassas, A. Solyman, and M. Farouk, "mTourism multilingual integrated solution: A case study "EgyptTravel"," in *Proc. Conf. eChallenges e-2014*, 2014, pp. 1–9.

[5] P.-J. Lin, C.-C. Kao, K.-H. Lam, and I. C. Tsai, "Design and implementation of a tourism system using mobile augmented reality and GIS technologies," in *Proc. 2nd Int. Conf. Intell. Technol. Eng. Syst.*, 2014, vol. 293, pp. 1093–1099.

[6] T. Schlachter, C. Düpmeier, R. Weidemann, W. Schillinger, and N. Bayer, ""My environment"—A dashboard for environmental information on mobile devices," in *Proc. Int. Symp. Environ. Softw. Syst. Fostering Inform. Sharing*, 2013, vol. 413, pp. 196–203.

[7] *Titanium*, [Online]. Available: http://www.appcelerator.com/titanium/ [Last Visited: 22/3/2015].

[8] *Xamarin*, [Online]. Available: http://xamarin.com/ [Last Visited: 22/3/2015].

[9] *PhoneGap*, [Online]. Available: http://phonegap.com/ [Last Visited: 22/3/2015].

[10] *XMLVM*, [Online]. Available: http://xmlvm.org [Last Visited: 22/3/2015].
[11] *J2ObjC*, [Online]. Available: https://code.google.com/p/j2objc/ [Last Visited: 22/3/2015].
[12] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md^2," presented at the *Proc. 28th Ann. ACM Symp. Applied Comput.*, Coimbra, Portugal, 2013.
[13] H. Heitkötter and T. Majchrzak, "Cross-platform development of business apps with MD2," in *Proc. Des. Sci. Intersection Phys. Virtual Des.*, 2013, vol. 7939, pp. 405–411.
[14] A. Holzinger, P. Treitler, and W. Slany, "Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones," in *Proc. Multidisciplinary Res. Practice Inform. Syst.*, 2012, vol. 7465, pp. 176–189.
[15] J. Ohrt and V. Turau, "Cross-platform development tools for smartphone applications," *Computer*, vol. 45, pp. 72–79, 2012.
[16] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," in *Proc. 16th Int. Conf. Intell. Next Generation Netw.*, 2012, pp. 179–186.
[17] R. Raj and S. B. Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," in *Proc. IEEE Annu. India Conf.*, 2012, pp. 625–629.
[18] A. Ribeiro and A. R. da Silva, "Survey on cross-platforms and languages for mobile apps," in *Proc. 8th Int. Conf. Quality Inf. Commun. Technol.*, 2012, pp. 255–260.
[19] P. Smutny, "Mobile development tools and cross-platform solutions," in *Proc. 13th Int. Carpathian Control Conf.*, 2012, pp. 653–656.
[20] H. Heitkötter, S. Hanschke, and T. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Proc. Web Inform. Syst. Technol.*, 2013, vol. 140, pp. 120–138.
[21] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proc. 6th Balkan Conf. Informatics*, Thessaloniki, Greece, 2013.
[22] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Taxonomy of cross-platform mobile applications development approaches," *Ain Shams Eng. J.*
[23] *JUniversal*, [Online]. Available: http://juniversal.org/ [Last Visited: 22/3/2015].
[24] W. El-Kassas, B. Abdullah, A. Yousef, and A. Wahba, "ICPMD: Integrated cross-platform mobile development solution," presented at the *9th Int. Conf. Comput. Eng. Sys.* Cairo, Egypt, 2014.
[25] V. Tunalı and Ş. Z. Erdoğan, "Comparison of popular cross-platform mobile application development tools," in *Proc. 2. Ulusal Yönetim Bilişim Sistemleri Kongresi*, 2015, pp. 1357–1365.
[26] J. Friberg, "Evaluation of cross-platform development for mobile devices," Master thesis, Dept. Comput. Inform. Sci., Linköping University, Linköping, Sweden, 2014.
[27] *Introduction Mobile Develop. – Xamarin*, [Online]. Available: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/ [Last Visited: 3/11/2015].
[28] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining API mapping for language migration," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 195–204.
[29] P. Klima and S. Selinger, "Towards platform independence of mobile applications," in *Proc. Comput. Aided Syst. Theory - EUROCAST*, 2013, vol. 8112, pp. 442–449.
[30] A. Puder and O. Antebi, "Cross-compiling Android applications to ios and Windows phone 7," in *Proc. 3rd Int. Conf. Mob. Netw. Appl.*, 2013, vol. 18, pp. 3–21.
[31] A. Puder, "Cross-compiling Android applications to the iPhone," presented at the in *Proc. 8th Int. Conf. Principles and Practice of Programming in Java*, Vienna, Austria, 2010.
[32] O. Antebi, M. Neubrand, and A. Puder, "Cross-compiling Android applications to Windows phone 7," in *Proc. Mobile Comput., Appl. Services*, 2012, vol. 95, pp. 283–302.
[33] A. Puder and J. Lee, "Towards an XML-based bytecode level transformation framework," *Electron. Notes Theor. Comput. Sci.*, vol. 253, pp. 97–111, 2009.
[34] J. Perchat, M. Desertot, and S. Lecomte, "Component based framework to create mobile cross-platform applications," in *Proc. Comput. Sci.*, 2013, vol. 19, pp. 1004–1011.
[35] *Using NRefactory for Analyzing C# code*, [Online]. Available: http://www.codeproject.com/Articles/408663/Using-NRefactory-for-analyzing-Csharp-code [Last Visited: 2/5/2015].
[36] S. Dhillon and Q. H. Mahmoud, "An evaluation framework for cross-platform mobile application development tools," in *Proc. IEEE 2nd International Conf. Mobile Services*, 2013.
[37] H. J. Kim, S. Karunaratne, H. Regenbrecht, I. Warren, and B. C. Wunsche, "Evaluation of cross-platform development tools for patient self-reporting on mobile devices," in *Proc. 8th Australasian Workshop Health Inform. Knowl. Manag.*, Sydney, Australia: 2015, pp. 55–61.
[38] H. Zhong, S. Thummalapenta, T. Xie, L. Zhang, and Q. Wang, "Mining API mapping for language migration," presented at the in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, Cape Town, South Africa: 2010, vol. 1.
[39] *Garbage Collection*, [Online]. Available: https://developer.xamarin.com/guides/android/advanced_topics/garbage_collection/ [Last Visited: 3/2/2016].

Queries to the Author

Q1. Listing has been renumbered as it was starting with 5—8 we have renumbered it to 1—4. Please check for correctness.
Q2. Please provide year in Refs. [7], [8], [9], [10], [11], [23], [27], [35], [39].
Q3. Please update Ref. [22] with complete bibliographic details.
Q4. Please provide author photos and bios.