Acta Cybernetica **21** (2014) 529–552.

Mobile Platforms and Multi-Mobile Platform Development*

Hassan Charaf, Péter Ekler, Tamás Mészáros, Imre Kelényi, Bence Kovari, István Albert, Bertalan Forstner, and László Lengyel†

Abstract

Mobile devices and mobile applications have a significant effect on the present and on the future of the software industry. The diversity of mobile platforms necessitates the development of the same mobile application for all major mobile platforms, which requires considerable development effort. Mobile application developers are multiplatform developers, but they prioritize the platforms, therefore, not all platforms are equally important for them. Appropriate methods, processes and tools are required to support the development in order to achieve better productivity. The main motivation of our research activity is to provide a method, which increases the development productivity and the quality of the applications and also reduces the time to market. The paper discusses our model-driven results on the field of multi-mobile platform development.

Keywords: Design Tools and Techniques, Domain-specific architectures, Domain engineering, Reusable libraries, Software Engineering Process

1 Introduction

Mobile devices play a significant role in the daily lives of the majority of people living in a consumer-based society [12] [32]. Many people own one or more mobile devices, from numerous device distributors, with a variety of special features, capabilities and application programming frameworks. The diversity of the mobile platforms requires to develop the same application several times, once for each of the supported mobile platforms.

*This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfred. This work was partially supported by the Hungarian Government, managed by the National Development Agency, and financed by the Research and Technology Innovation Fund (grant no.: KMR_12-1-2012-0441).

†Budapest University of Technology and Economics, E-mail: {hassan, peter.ekler, mesztam, imre.kelenyi, kovari, ialbert, bertalan.forstner, lengyel}@aut.bme.hu

DOI: 10.14232/actacyb.21.4.2014.2

In 2008 $4.2 billion was spent on mobile applications. In 2013, an estimated $29.5 billion will be spent [12]. Both the tendency and the magnitude of these numbers reflect the fact that mobile phones are a strong part of our everyday life. We always take our mobile phones with us, continuously check our e-mails, social networks and other websites.

The mobile developer mindshare in 2013 shows that Android is leading with more than 70% of developers using the platform, followed by iOS at about 55% [33]. The [33] survey is based on more than 6,000 developers' mind from over 115 countries. Currently HTML5 is also a mobile development technology, with more than 50% of the developer population using HTML5 technologies for developing mobile applications. In case of HTML5, there are different approaches to mobile development:

- Mobile websites: websites that are designed to be rendered on small screens. Responsive websites are included into this category.
- Mobile web applications: websites with offline storage and deeper browser integration.
- Hybrid applications, using native wrapper: in this case HTML code wrapped in a browser, within a native shell (e.g. PhoneGap [26]).
- Applications using native JavaScript API: platforms exposing software and hardware services through a JavaScript API (e.g. Firefox OS [9] and BlackBerry 10).
- HTML5/JavaScript applications converted to native: JavaScript is handled as a platform independent code and convert it to a native application (e.g. Appcelerator Platform).

The applications developed with the first two approaches are distributed as web applications, while others as native applications via the application stores.

The latest research [33] shows that developers' platform choices depend very much on the goal they aim to achieve. When it comes to platform selection, contract developers vote for platforms that will generate more revenue, Chief Information Officers (CIOs) prefer efficiency and low cost, Chief Marketing Officers (CMOs) focus on reach, while hobbyists want to experiment with newer platforms [34].

There is no one-size fits all across mobile platforms. iOS is selected more frequently than average by developers who focus on value revenue potential, graphics, application discovery and user reach. Developers tend to use HTML5 more frequently as their primary platform when they value porting and speed and cost of development. BlackBerry 10 is used more frequently than average as a primary platform by developers valuing developer community programs. Windows Phone is most popular for developers who are already familiar with the .NET Framework [33].

We can see that the current market is rather colorful. Today's mobile application developer is a multi-platform developer. Developers use almost 3 mobile platforms

concurrently. The surveys do not show a significant difference in the last 3 years, i.e. the value is always between 2.6 and 3.2. Therefore, the main motivation of our research activities is to support the developers with appropriate multi-platform development methods and tools.

Our team has a reasonable experience both on the filed of mobil application design and development, and on the multi-platform management. In our approach, multi-platform solutions are driven by model-based solutions and software artifact generation methods. During the last 12 years we have supported multi-mobile application development in different ways with several different methods. We have worked out common mobile platforms, provided methods to synchronize user interfaces of different mobile platforms [20], and applied multi-paradigm modeling techniques in multi-platform mobile development [17]. Furthermore, we have moved forward a model-based unification of mobile platforms method [18]. Also, we have introduced the VMTS mobile toolkit [19] that is based on out modeling and model processing framework (Visual Modeling and Model Transformation System [4] [35]).

In the last two years we have standardized the results of all these activities, redesigned and rebuilt our multi-platform development method. This paper summarizes the result of our actual activities and introduces the new method we use in our current multi-platform development software projects.

The rest of this paper is organized as follows. Section 2 discusses the mobile platforms, concentrating on the market share and platform diversities. Section 3 provides our earlier results on the field of multi-mobile platform development. Section 4 and 5 introduces the details of the actual results and current multi-mobile platform development method. The discussed model-driven method supports the generation of software artifacts from common mobile application models. Section 6 summarizes the related work and compares our solution with other multi-platform approaches. Finally, conclusions are elaborated.

2 Mobile platforms

The different mobile platforms and the different strategies of the device manufacturers continuously modify the market conditions. The manufacturers provide not only devices but different services as well. End users are served with several custom solutions and trendy features. The competition of the platforms and the handset makers led to the current situation.

The mobile device market currently (in 2013) is dominated by Android devices, from low-end feature-phone replacements to high-end devices. In the first half of 2013, Android had about 75% of all smartphone shipments, while iOS had another 18% leaving very little room for anyone else [13] [33].

Smartphone sales by handset makers show that Samsung is at the top end, and there are numerous competitors at the long tail of the distribution (Apple, LG, Nokia, Huawei, BlackBerry, ZTE). The other segment, which is not covered by the mentioned manufacturers of smartphone makers is currently selling as many devices as Samsung. These handset makers are made up of hundreds of Android

handset producers. Taking these into account, it is interesting that Samsung's main competitor in terms of market share is not Apple, but these handset makers who are able to supply the cheapest possible smartphones, customized for every corner of the developing world [33].

The reason Samsung is still making profits among modular handset makers is because they have realized that there are no profits to be made in handset production itself. In other words, hardware is not enough. Instead, value has migrated upwards in the technology stack (to services) and downwards (to handset components). Also, it is a fact that Samsung spends more for marketing than Coca-Cola.

The lead platform is where new applications and features are first rolled out and which can be the star of the marketing launch. Prioritization also has an impact on focus, application quality, sales and revenue. The way developers prioritize the platforms has a direct impact on the overall perception of the platform. If most developers treat a platform as a second-class citizen, this will reflect negatively on the application quality and consequently, on developers' revenue opportunity on that platform. Developers that prioritize a platform will act as evangelists for that platform, as they are likely to create high quality, most up-to-date applications and praise the platform at events or social media. As a consequence, supporting developers with tools, documentation, frameworks and easy-to-use libraries can increase the market share of the platform [33].

Actual surveys show that more than 80% of mobile developers are using iOS, Android or HTML5 (mobile) as their primary target platform. In this world, not all platforms are equally important to a developer. Platform priorities also depend on the level of experience. Developers who are fresh to mobile have a much stronger preference towards Android, with almost twice as many new mobile developers preferring Android than iOS.

Asking a developer to switch to a different platform is like asking someone to learn a foreign language. This is a task that takes months. The challenge is not just about the language (Objective C, Java, C#, HTML or JavaScript) itself. It is the set of APIs, development environment, publishing process, and the 3rd party tools ecosystem that supports the platform. Learning a language nowadays is not very hard, they all look the same (literally), but what a developer has to learn is the API.

Developer tools are not just nice-to-have. Tools are in the must-have application development arsenal of the most sophisticated developers, and also those making most revenues. Appropriate tools can increase the productivity also the quality of the products. Tools can support to utilize certain artifacts for several different mobile platforms. Developer tools can make a platform more attractive for developers. Also the tool can reduce the time to market for mobile applications.

As a summary we can conclude that today's developers are multiplatform developers, but not all platforms are equally important for them. In case of the multiplatform developers the main decisions are often based on priorities. Appropriate methods, processes and tools are required to target several platforms, support the development in order to achieve better productivity and quality.

3 Multi-mobile platform development - Our earlier results

In embedded software development, reuse is recognized as a key factor for better productivity at lower costs. In the ideal case, the product family approach makes it possible to reuse elements in a whole range of related products. The family members are based on a common architecture and rely heavily on reusable components, thus allowing the developers to concentrate only on the required variation between the products. This approach also enables the developers to focus on design instead of implementation details.

We have realized that from the perspective of mobile software development one of the greatest challenges is caused by the fragmentation of mobile platforms. Each mobile platform has different advantages, thus it is hard to find an ultimate platform for applications. Not only the development tools but the supported languages and even the application life cycles are different, thus each platform requires developers with special knowledge related to the specific platform. To make it possible to support the creation of a common development platform, we have to apply methods that move a reasonable part of the development to a higher abstraction level.

One of our first solution was the **Common Mobile Platform** (CMP), which was a solution applied between 2005 and 2008 to model mobile applications and generate source code for different mobile platforms. CMP defines a model and XML language for describing mobile applications and provide a generator mechanism which generates working source code for the following mobile platforms: Symbian 3rd edition, 5th edition [14], Windows Mobile 5-6, Microsoft .NET Compact Framework (.NET CF) [37] and Java 2 Micro Edition (J2ME) [16]. The solution was based on the Model-Driven Architecture (MDA) [22]. We differentiated platform-independent models that described the common behavior of the required application and platform-specific models generated automatically from platform-independent models. The source code is also automatically generated from platform-specific models. The generated code utilized the frameworks prepared to support the area.

Domain-specific models [11] have another remarkable advantage over usual software development methods: by using different code generation templates, we can produce applications for different application platforms. This means that from a single model set, we can generate our application for all target platforms, including mobile platforms, web and desktop applications as well, and thus, changes applied on the models can be immediately implemented on all platforms.

The Simplian Framework. Even tough, Symbian was one of the most popular mobile platforms, Symbian-based software development was far more difficult and required more specific skills than the development of desktop applications. This stemmed not only from the prestandard C++ characteristics, but also from the absence of easy-to-use integrated development tools.

We can mention, for example, the complicated memory management (for instance cleanup stack and two-phase object construction), the special exception-handling (leave-mechanism), string- and array handling, or the unique aspects of

resource-management. Also, developers must strictly take care of these uncommon circumstances, since ignoring them could lead to bugs that are hard to identify.

We provided a class library that helped the programmer by hiding all the recurring tasks deriving from the mentioned facts. The Simplian Framework provided an API for constructing the user interface of the application, and other simple means to bind data to the widgets, or send and receive them through different communication channels. Simplian also provided tools to generate the C++ code from a well-defined, platform-independent XML file, which could be constructed by the modeling tool [1].

The Symbian platform related model processor supported user interface, data binding and database generation, thus, a database metamodel was also provided as an input. The model processing solution used the Microsoft CodeDOM technology [31] for code generation. The CodeDOM consisted of classes representing the syntactic elements of the .NET languages, like C# and managed C++.

The .NET Compact Framework. Applying the same method with different model processors we generated applications from the same models for devices with .NET Compact Framework. We do not introduce the model processors related to all aspects of different mobile platforms, but we note that the main difference between the two transformations (Symbian and .NET CF related transformations) was in the resource model (user interface model) processing. For the .NET CF platform certain properties of the user controls are generated based on the attributes of the resource model. The rewriting rules did not use these attribute values during the generation for Symbian platform, because the controls were placed strictly one below other.

The presented approach made possible to use visual languages to define user interface, database and communication models that described the different aspects of mobile applications. The solution provided model processors to generate the platform-specific source code. The solution was realized with domain-specific language engineering and graph rewriting-based model transformation.

4 Multi-mobile platform development - Overview of the actual method

Model-driven software engineering is an actively researched field. The growing size and complexity of software systems made software modeling technologies essential in application development. Model-driven development approaches can increase the development productivity and the quality of the produced software artifacts.

Model-driven development approaches emphasize the use of models at all stages of system development. In model-based development, models are used to describe the most artifacts of the system, i.e., interfaces, interactions, and properties of all the components that comprise the system. These models can be manipulated in a number of different ways to describe the system, and in certain cases to generate the complete implementation of the system. In order to capture the semantics that is as close as possible to the domain of the developed system in an effective manner,

building a domain-specific modeling language is a suitable choice. Using domain concepts to modeling systems helps increase productivity, makes systems easier to maintain and evolves and shortens the development cycle.

Our modeling and model transformation framework is the Visual Modeling and Transformation System (VMTS) [4] [35]. VMTS is a metamodeling environment which supports editing models according to their metamodels. Models are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel ("visual vocabulary").

Also, VMTS is a model transformation system, which transforms models using both template-based and graph rewriting techniques. Moreover, the tool facilitates the verification of the constraints specified in the transformation step during the model transformation process. VMTS has been developed since 2003.

The VMTS approach uses a graphical notation for control flow (the execution sequence of the transformation rules): stereotyped UML activity diagram [23]. The control flow language can express a transformation as an ordered sequence of the transformation rules. Classical graph grammars apply any production that is feasible. This technique is appropriate for generating and matching languages, but model-to-model transformations usually need to follow an algorithm that requires a stricter control over the execution sequence of the steps, with the additional benefit of making the implementation more efficient. The control flow language supports the following constructs: sequencing transformation steps, branching with C# conditions, hierarchical steps, parallel execution of the steps, and iteration.

The architecture of the application generation process is depicted in Figure 1. The modeling of mobile applications and the processing of these models are performed in a framework. We use mobile, domain-specific languages to define the required structure and application logic. Platform-specific model processors are applied to generate the executable artifacts for different target mobile platforms. The generated code is based upon the previously assembled mobile, platform-specific frameworks. These frameworks provide energy efficient solutions for mobile applications. Furthermore, some data processing or computationally intensive tasks are passed into the cloud to save the battery power of the mobile device.

Mobile, domain-specific languages address the connection points and commonalities of the most popular mobile platforms. These commonalities are the basis of further modeling and code generation methods. The main areas, covered by these domain-specific languages, are the static structure, business logic (dynamic behavior), database structure and communication protocol. Using these textual and visual languages, we are able to integrate the use of cloud services into the business logic.

For each target platform, a separate transformation should be realized since, at this step, we convert the platform-independent models into platform-specific executable code. The transformation expects the existence of the aforementioned frameworks and utilizes their methods. The generated source code is essentially a list of parameterized activities (commands) that certain functions of the mobile application should perform. This means that the core realization of the functions is not generated but utilized from mobile-platform specific frameworks, i.e., the

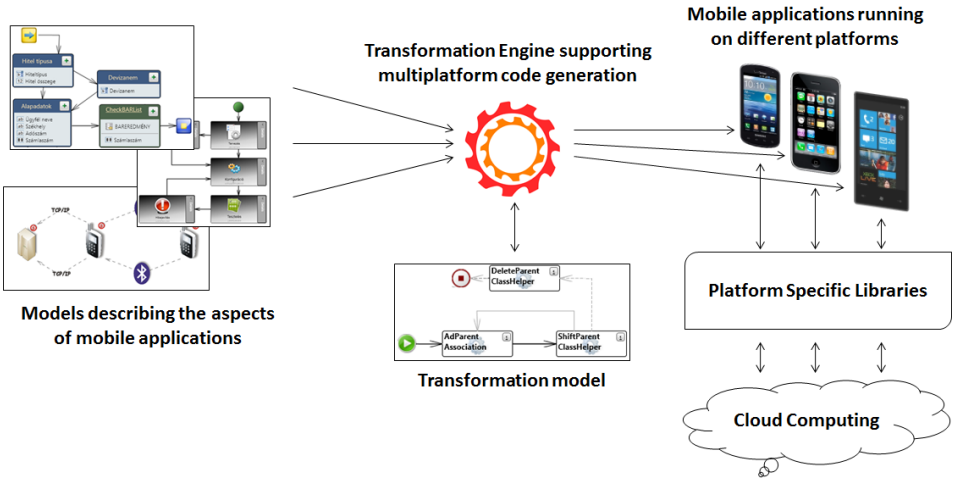

Figure 1: Supporting multi-mobile platform development

generated code contains the correct function calls in an adequate order, and with appropriate parameters. The advantages of this solution are the followings:

- The software designer has easier task. The model processors are simpler. The model processing is quicker. The generated source code is shorter and easier to read and understand.

- We use prepared mobile, platform-specific frameworks. These frameworks are developed by senior engineers of the actual mobile platform.

The current wave of our research activities is focused on the Android [3], iOS [15] and Windows Phone 8 [10] [29] platforms.

Model processors generate platform-specific source code. Developers integrate the generated code into the source of a native application. The generated code is based on well tested libraries created by platform specialists. Therefore, the generated code does not contain the whole implementation of the features, instead, it utilizes the services of the library via API calls. In this way the library can be utilized several times by different generated and hand written source code snippets. This is because the goal is not to generate all of the code, but (i) to perform the modifications where it is easier (either in the model world or in the source code), (ii) automatic synchronization from model to code, which is supported by separated source files for generated and hand written code (partial classes, inheritance), and to (iii) speed up the development of the same application for different platforms.

Table 1 summarizes the tasks that are typically required during mobile application development. The table highlights which tasks are supported with our approach and which tasks requires further manual coding. In the current state of

our solution the network communication and the resource management is already implemented in the framework while the others are under development.

Table 1: Mobile development tasks.

Task	**Approach**
User interface and screen flow	Manual (mockup can be generated)
Custom views	Manual
Persistence and data model	Generate
Network communication	Generate
Resources (localization)	Generate
Business logic	Generate
Multimedia (e.g.: camera, music)	Generate
Location based services and map	Generate

The key points and the evaluation of the method. In summary, we still do not believe that manual coding can be eliminated, because there are complex functions and platform-specific tasks, which require manual coding. This means that the gluing code required to integrate the generated source, platform specialties and the custom logic are manually added. But it is worth to model several parts of the application and generate the appropriate source code based on it. This increases the quality and shortens the time to market of the applications. We are applying the method in both mobile application development and server side component development. Key points of the approach are the prepared platform-specific frameworks and libraries. They can be utilized by both generated and handwritten code, therefore they support rapid development and provide higher quality.

5 The methods of multi-mobile platform development

5.1 Modeling: Defining the Mobile Applications on a Higher Abstraction Level

Instead of discussing all aspects of mobile platforms that the method supports, we decided to focus on one area of the covered fields and provide more details about it. The selected area is the rapid prototyping of REST-based communication channels (Representational State Transfer) [8] [27]. The methods supporting this area, i.e. the way how we model some aspects of the mobile platforms and how the model processors generate the source code, are similar for several different aspects of mobile platforms as well.

Depending on the expectations of the customer, the mobile application developers need to develop either only the mobile clients for an existing server application

or the server applications supporting the clients as well. In the most common scenario, there is one server-side application, and the mobile clients should be developed for multiple target platforms concurrently. Therefore, the communication layer should be developed for each platform. Though, the concrete implementations of the layers on the different platforms show significant similarity, it is not possible to reuse the source code between the platforms because of the different languages and runtime environments. These facts are motivating issues to apply platform-independent modeling and model processing to support these types of challenges.

Mobile applications usually apply a REST-based communication channel when it is about data exchange between the different devices. In REST, the operations of the server application can be accessed with the help of a properly formatted HTTP request. The parameters of the request may be encoded either into the request URL itself, or into the body of the HTTP request. In the latter case, the formatting of the parameters may be arbitrary, although the two most often applied serialization mechanisms for the parameter objects are XML and JSON. The server application responds to such a request with a HTTP response that contains the response parameters in its body (again, usually as XML or JSON). Even though we know if the request and response body is formatted as XML or JSON, their concrete schema may be arbitrary and is not tied to strict rules like in case of SOAP [30]. Consequently, the serialization and deserialization procedures as well as the URL generators and interpreters should be developed on both the client and the server sides. Furthermore, in case of the client application, the same development must be performed in case of each targeted mobile platform as well.

As an initial step, we focus on REST APIs exclusively and automate this error prone, monotonous coding for the client side. We have elaborated a modeling language that is able to describe server-side operations, the data types used and the way of parameter serialization. We have also prepared the generators that automate the creation of the client-side communication layer based on the models.

There are various forms to represent a modeling language. For practical reasons and to speed up the initial development we have chosen to realize this language based on an already existing programming language: C#. The idea is to define the communication API with the help of C# interfaces and to generate the concrete implementations based on these interface definitions for various platforms. Utilizing C# as a base language has several benefits over implementing a proprietary modeling language, or using general purpose data description languages like XML or JSON to describe the data models. First, the syntactic and semantic verification of the language can be performed using existing C# compilers without any additional effort. The appropriate usage of the types is forced by the strongly-typed property of C#. By compiling the interfaces into a .NET assembly, we can easily traverse and interpret the models by traversing the assembly using reflection. Next, since the final generated code is using languages similar to C# (C#, Java, Objective C), the data structures and interface definitions are close to the final implementation and can be described in a way familiar to the developers. Furthermore, the VMTS environment handles C# as the textual concrete syntax of the models, and

in a similar way visual concrete syntax can be also provided for the same model. Users can decide which concrete syntax (textual or visual) they want to use for the interface definition.

With the help of interfaces, we can precisely define the methods that can be called on the server application, and we can also define the possible data types we can pass to these methods or we can expect from these methods as a result value. The way how such a method call is performed, how the parameters are serialized when passing them to the methods and how the result values are deserialized on a successful call can be customized with the help of .NET attributes. The code generators processing these custom domain-specific models (interfaces) discover the attributes attached to the elements of an interface definition and modify the code generation accordingly.

Now we introduce our approach with an example, where the server has a simple method that is used to create a new user in the target system. The method expects one parameter (the name of the user) and returns nothing. The corresponding C# interface is the following.

```
[RestApi]
public interface MyService
{
    void insertUser(string userName);
}
```

To indicate that this interface defines the API of a server application we denote it with the *[RestApi]* attribute above the interface. By finding this attribute the code generator will recognize that this interface should be treated as a REST API interface, and it should generate the client-side proxy class for that. This *[RESTUrl]* attribute specifies which Url to invoke. This way, the generated proxy will navigate to the insertuser.php page, and pass the username as url parameter (like *insertuser.php?userName=XXXX*).

If we would like to use different parameter names instead of the names of the parameters in the C# interface, we may customize that using the *[RestParam]* attribute.

```
[RestApi]
public interface MyService
{
    [RestMethod(Url = "insertuser.php")]
    void InsertUser ([RestParam(Name="usr")]string userName);
}
```

In the example above, though, the name of the parameter is username, it is mapped to the *usr* http GET parameter (*insertuser.php?usr=XXXX*). If we need to use different HTTP methods, e.g. POST instead of GET to call the server-side service, we can specify it as the *CommandType* a parameter of the *[RestMethod]* attribute.

Changing the command type to POST (possible values are GET, POST, PUT, DELETE) we just instruct the code generator to generate a proxy code that uses the HTTP POST command to send the request. The passed parameters are still encoded into the request url, as before. If we would like to pass the parameter inside the body of the HTTP request as HTTP form parameter instead of the request url itself, then we can set it up using the *Mapping* parameter of the *[RestParam]* attribute.

```
[RestApi]
public interface MyService
{
    [RestMethod(Url = "insertuser.php",
       CommandType = CommandType.POST)]
    void InsertUser ([RestParam(Name="usr", Mapping =
       RestMethodMappingType.Body)]string userName);
}
```

The default value for Mapping is *RestMethodMappingType.Url*. Although, we can already pass single parameters both in the request URL and in the HTTP body as form parameters, often the argument should be handled as not a parameter of the target resource but a locator for the target resource. E.g. consider that the user we create is assigned to a specific client. But the client is not passed as a parameter to the insertuser.php page, but the insertuser.php page is located inside the appropriate client folder like *http://..../client1/insertuser.php...* . To map a specific parameter into the resource Url at a specific position, we must set the Mapping argument for that parameter to *RestMethodMappingType.Custom*, indicate the position of this parameter with the $ character.

Since a server method usually has a return value as well, it must be handled by the generated proxy code. Consider that the *InsertUser* method returns the unique id of the newly created user inside the HTTP response as plain text. This return value can simply be returned by the generated proxy method by setting the return type of the *InsertUser* method to string.

```
[RestApi]
public interface MyService
{
    [RestMethod(Url = "$client/insertuser.php", CommandType =
       CommandType.POST)]
    string InsertUser ([RestParam(Mapping =
       RestMethodMappingType.Custom)]string client,
       [RestParam(Name="usr",
       Mapping = RestMethodMappingType.Body)]string userName);
}
```

If the service returns a more complex value serialized in XML format, we may use custom return types as well, and indicate the type of serialization using the *ReturnFormat* property of the *RestMethod* attribute.

```
[RestApi]
public interface MyService
{
    [RestMethod(Url = "$client/insertuser.php", CommandType =
         CommandType.POST, ReturnFormat = FormatType.XML)]
    UserInfo InsertUser ([RestParam(Mapping =
         RestMethodMappingType.Custom)]string client,
          [RestParam(Name="usr", Mapping =
          RestMethodMappingType.Body)]string userName);
}
```

Here we support four options: *XML*, *JSON*, *Forms* and *Raw*. The Forms option covers the case when the parameters are formatted as HTTP POST key-value pairs, while the *Raw* formatting means transferring the value as it is, without any formatting.

The type of serialization and the way how parameters are passed is usually the same for each method within the same service. Thus, to avoid the tedious setup of the *RestParam* attribute at each method (if they differ from the default one), it is also possible to set up the common parameters for the entire service by placing this attribute above the interface declaration:

```
[RestApi]
[RestParam(Mapping = RestMethodMappingType.Body, Format =
FormatType.Form)]
public interface MyService
{
    [RestMethod(CommandType = CommandType.POST)]
    UserInfo InsertUser(string client, string userName);
}
```

In the example above, both parameters of the *InsertUser* method are serialized as form parameters inside the body of the HTTP request. If someone needs different behavior for specific method parameters, the default settings may be overridden by a *RestAttribute* parameter placed in from the related parameters.

Declaration of the custom data types. In most practical cases, the parameters expected by the methods or the return values of them are not only primitive types like string, integer or floating point number, but complex types consisting of multiple fields.

For this purpose, we have defined another interface-level attribute called *RestDto* that is the abbreviation of REST Data Transfer Object. Interfaces marked by this attribute will be handled by the code generator as simple classes used for representing and transmitting data. Of course, in addition to the standard data storing feature of such an object, the code generator may extend it with various additional features like change notification, equality comparison and so on.

Assume that when creating a new user, we would like to pass also the full name and the age of the user to be created, and we do not want to use a separate

method argument for them but handle them as one unit. Then we may wrap these parameters into a new DTO interface.

```
[RestDto]
public interface User
{
    string UserName { get; set; }
    string FullName { get; set; }
    int Age { get; set; }
}
```

A complex type cannot be simply encoded into the request Url, thus, we must set it up to be serialized inside the body of the HTTP POST request.

Using the default settings, the user parameter would be serialized as converting its fields into HTTP form parameters. But typically, in the REST communication rather XML or JSON serialization is applied. The way how complex parameters should be serialized can be specified with the *Format* argument of the *RESTParam* attribute.

```
[RestApi]
public interface MyService
{
    [RestMethod(Url = "$client/insertuser.php",
    CommandType = CommandType.POST)]
    string InsertUser ([RestParam(Mapping =
      RestMethodMappingType.Custom)]string client,
      [RestParam(Mapping = RestMethodMappingType.Body,
      Format = RESTFormatType.XML)]User user);
}
```

Of course, the same data may by serialized as XML in an infinite number of ways. By default, we assume each member field to be serialized as an XML tag identified by the name of the tag, while the value of the field is serialized as the content of the XML tag. If this value is of primitive type, then simply its printed value, if the value is of a complex type, then the same method is applied recursively. If we would like to change the way how the XML document is generated and parsed, we can perform it using the standard .NET XML formatter attributes by attaching them to the DTO definition.

Often, it is more comfortable to pass parameters to a method call as separate arguments, however, we would like to serialize them as one connected XML document. For example, we would like to use separate username, password and age arguments for the *InsertUser* method, but would like to serialize them in the request body as they would be part of one User class. For this purpose, you may set the IsCoupled property of the RestParam attribute to true:

```
[RestMethod(Url = "$client/insertuser.php",
```

```
CommandType = CommandType.POST)]
UserInfo InsertUser([RestParam(Mapping =
      RestMethodMappingType.Custom)]string client,
      [RestParam(Mapping = UrlMappingType.Body, Format =
      FormatType.XML, IsCoupled = true)]User user);
```

The method signature generated based on this declaration will look like this:

```
UserInfo InsertUser(string client, string userUserName,
      string userPassword, int userAge)
```

However, the last three parameters will be serialized as one XML document like

```
<User usr="joe" full="John Doe" age="30"/>
```

Figure 2 summarizes both the Service interface and the DTO specification attributes.

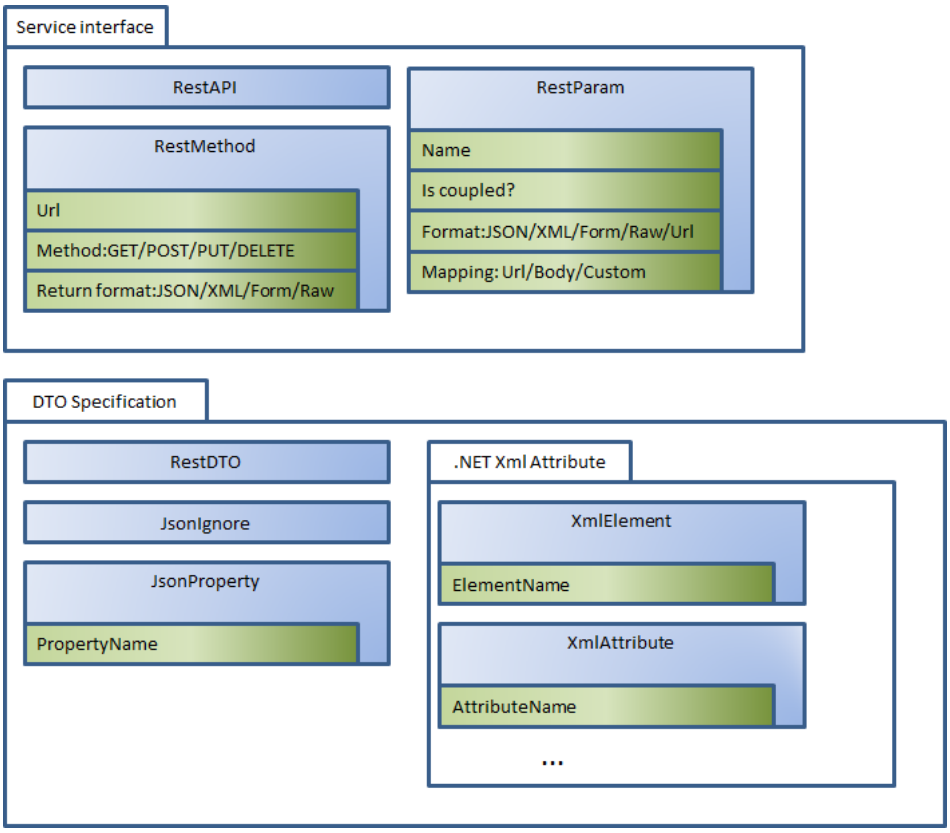

Figure 2: Service interface and DTO specification attributes

5.2 Model Processing: Generating the Software Artifacts

Having the features of a network service, including its methods and the applied data types already defined, the next step is that to generate executable source code which is able to perform the communication with the server component. There are various solutions that can be utilized when it is about code generation, we have chosen the Microsoft T4 (Text Template Transformation Toolkit) [24]. T4 is a mixture of static texts and procedural code: the static text is simply printed into the output while the procedural code is executed and it may result in additional texts to be printed into the output. Recall that, the interface definitions are compiled into a .NET assembly that can be loaded and traversed using reflection afterwards. The T4 templates we write also work on the reflected content of the assemblies.

Currently we are targeting two mobile platforms: Windows Phone 8 (C#) and Android (Java). Therefore, we need to prepare two different T4 templates for the two platforms. In case of Windows Phone, we expect the data transfer objects be represented by C# classes, the fields of the DTO entities should be represented as .NET properties, and the generated classes should also support some kind of change notification about changes in the properties. A possible implementation of the template is the following.

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute
  ("System.Runtime.Serialization", "3.0.0.0")]
[System.Runtime.Serialization.DataContractAttribute
  (Name="<#=type.Name#>")]
public partial class <#=type.Name#> :
  System.ComponentModel.INotifyPropertyChanged
{
<# foreach (var pi in type.GetProperties()) { #>

    private <#= pi.PropertyType.FullName #> <#=pi.Name#>Field;

    public <#= pi.PropertyType.FullName #> <#=pi.Name#>
    {
        get
        {
            return this.<#=pi.Name#>Field;
        }
        set
        {
<# if (!pi.PropertyType.IsValueType) { #>
          if (!object.ReferenceEquals(this.<#=pi.Name#>Field,
             value)) <# } else { #>
          if (!this.<#=pi.Name#>Field.Equals(value))<# } #>
          {
             this.<#=pi.Name#>Field = value;
```

```
            this.RaisePropertyChanged("<#=pi.Name#>");
          }
        }
    }
<#}#>

    public event System.ComponentModel.
      PropertyChangedEventHandler PropertyChanged;
    protected void RaisePropertyChanged(string propertyName)
    {
        System.ComponentModel.PropertyChangedEventHandler
          propertyChanged = this.PropertyChanged;
        if ((PropertyChanged != null))
        {
          PropertyChanged(this, new
            System.ComponentModel.PropertyChangedEventArgs(
            propertyName));
        }
    }
}
```

The input of the template (type) is the reflected DTO type. The resulting code will describe a partial class the name of which corresponds to the name of the interface. Then, the template iterates through all the fields declared in the interface, generates a private variable and a wrapper property for the variable. When setting up the value of the property, it checks if the new value is really different from the previous one, and changes the value of the underlying variable if it is really different. Then it also calls the *RaisePropertyChanged* method (passing the name of the changed property to it as parameter) that fires the *PropertyChanged* event.

For the Android (Java) implementation we do not need the DTOs to support any special features thus, here we just generate plain old Java (POJO) classes collecting private fields with getter and setter methods. The corresponding T4 template is much simpler as well.

```
<# Type type = this.DTOType; #>

<#= RenderCustomAttributes(type) #>
public class <#=type.Name#> {
<# foreach (var pi in type.GetProperties()) { #>

    <#= RenderCustomAttributes(pi) #>
    private <#= AHelper.MapType(pi.PropertyType) #> <#=pi.Name#>;

    public <#= AHelper.MapType(pi.PropertyType) #>
        get<#=pi.Name#>() { return this.<#=pi.Name#>;
```

```
    }

    public void set<#=pi.Name#>(<#=
         AHelper.MapType(pi.PropertyType) #> value) {
         this.<#=pi.Name#> = value;
    }
<#}#>
}
```

There is a big difference compared to generating target code for C#, though. Since the interface was also written in C# and uses the primitive types of the .NET Base Class Library, and we are traversing the .NET assembly, we must translate the .NET types to Java types. In case of the C# generator template, we could simply jump over this step, since we could use the same type names as in the interface definition itself (see *PropertyType.FullName* in the template). In case of the code template for Java, we perform this translation using the *TTHelper.MapType* method. This is a custom implementation handling some primitive types only, but can be arbitrarily extended with further types as well.

Proxy generation for the service methods. In general, it is advised to keep the generators as simple as possible, and outsource all the common implementations into helper classes or base classes. We followed this principle during realizing the code templates that generate the service proxy classes.

In case of the .NET implementation, all the communication-specific parts of the implementation are moved into the *RestClient* class. Its *SendRequest* method is used to perform the serialization and deserialization of the method parameters and the return value as well as the sending and receiving of the HTTP requests. For this purpose, the *SendRequest* method needs to know how the parameters should be processed during the assembling of the messages. Therefore, we utilize the same interface-, method- and parameter-level attributes into the generated code as well and let *SendRequest* discover these settings via reflection.

In case of the Android implementation, all the communication-specific parts of the implementation are outsourced into the *RESTTask* class that is subclassed from *AsyncTask*. In case of an Android *AsyncTask*, the network communication is performed on an asynchronous thread, and the caller of the thread is notified about the result via a *BroadcastReceiver* object. This object can then identify the called method and interpret the answer for the HTTP request. Therefore, we have split the implementation of the service proxy into two parts: the one is responsible for serializing the method parameters and instructing *RESTTask* to perform the HTTP request with the appropriate parameters, the other one (*RestBroadcastReceiver*) is responsible for receiving the asynchronous HTTP responses and to call the appropriate method to process the answer.

We use T4 templates to perform the above introduced method. Beside the discussed Android and Windows Phone platforms related templates the solution also supports the iOS platform with platform-specific templates.

5.3 Evaluation of the Solution

In summary, we can say that our objective targets one of the most pressing problems in the area of mobile software development, originating from the diversity of mobile platforms. To address this problem, we have provided a modeling language for mobile applications and developed frameworks for all platforms, which support the code generated from the models. The result is a method that allows the design of mobile applications and generates source code for major mobile platforms.

The presented domain-specific language that facilitates the definition of different aspects of the mobile application can have both textual and visual concrete syntax.

The main strength of the method is that it effectively supports the application design and development: speeds up the development and increases the code quality by automatically generating both the client and server side components. The generated code is not a full, buildable and executable artifact. It requires integration, i.e. further gluing code between the different mobile application aspects (user interface, communication, database management, others). This is a conscious decision behind the method. The objective is to support the repetitively occurring and lengthy coding processes and not to eliminate all aspects of programming. There are several areas that are easier to define with models and also several aspects that is more effective to directly write within the appropriate programming environment. The method helps to utilize both of these issues.

The method provides a general guidance that other teams can follow. The tool support still not available in a public form. We are continuously working on the tooling environment and will make it available for the community.

6 Related Work

This section introduces the most known cross-platform development frameworks and solutions. We summarize their capabilities and compare their achievements with our method.

PhoneGap [26] is a mobile development framework enabling developers to build applications for mobile devices using standards-based web technologies (HTML5, JavaScript and CSS3) instead of device-specific languages like Java, Objective-C or C#. The resulting applications are hybrid, meaning that they are neither truly native, because all layout rendering is done via web views instead of the platform's native UI framework, nor purely web-based, because they are not just web applications. Applications are packaged as applications for distribution and have access to native device APIs. It is possible to mix native and hybrid code snippets.

Earlier versions of PhoneGap required a developer making iOS applications to have an Apple computer, a developer making Windows Phone applications to have a computer running Windows, and so on. Currently, the PhoneGap Build service allows a programmer to upload his source code to a cloud compiler that generates applications for the supported platforms.

Appcelerator Titanium [2] is a platform that, similarly to PhoneGap, supports the development of mobile, tablet and desktop applications using web technologies. The Appcelerator Titanium framework is available since 2008.

Appcelerator Titanium Mobile framework allows web developers to apply existing skills to create native applications for iPhone and Android. However, in case of Appcelerator Titanium Mobile, developers should not only be familiar with web technologies and JavaScript syntax, but they also have to learn the Titanium API, which is different from familiar web frameworks like jQuery.

All application source code gets deployed to the mobile device where it is interpreted. Being interpreted means that some errors in the source code will not be caught before the program runs. Program loading takes longer than it does for programs developed with the native SDKs, as the interpreter and all required libraries must be loaded before interpreting the source code on the device can begin.

At the end of 2012, there were more than 30,000 applications shipped to the application stores built with Titanium. Appcelerator also offers cloud-based services for packaging, testing and distributing software applications developed on the Titanium platform.

Xamarin [36] is a company created by the engineers that created Mono [21] MonoTouch and Mono for Android, which are cross-platform implementations of the Common Language Infrastructure (CLI) and Common Language Specifications (Microsoft .NET).

Xamarin.Mobile is a library that exposes a single set of APIs for accessing common mobile device functionality across iOS, Android, and Windows platforms. The solution allows to use C# programing language and with the same code support all the mentioned platforms. Xamarin.Mobile currently abstracts the contacts, camera, and geo-location APIs across iOS, Android and Windows platforms. In the future, it will include notifications and accelerometer services.

Firefox OS [9] is a Linux-based open-source operating system for smartphones and tablet computers. It is being developed by Mozilla, the non-profit organization best known for the Firefox web browser. Firefox OS is designed to provide a complete community-based alternative system for mobile devices, using open standards and approaches like HTML5 applications, JavaScript, a robust privilege model, open web APIs to communicate directly with cellphone hardware and application marketplace.

As such, it competes with proprietary systems like Apple's iOS, Google's Android, and Microsoft's Windows Phone as well as other upcoming open source systems under development. Firefox OS was publicly demonstrated in February 2012.

Comparing our approach with PhoneGap, Appcelerator, Xamarin.Mobile and other multi mobile platform solutions, we can say that the goal is similar but not exactly the same. Available solutions target to produce the final executable files, i.e. the applications that are ready to use, that can be downloaded and installed. This approach is quite comfortable from both the end users and the developers point of view. But, of course these types of applications are limited to certain functions. Automatically generated applications can contain only those functions that have

the appropriate implementation or support in the mobile platform-specific libraries, in the supporting SDKs or APIs. In contrary, the goal of our solution is to speed up the development and not to eliminate the native programming. We use software modeling to design different aspects of the mobile applications and generate some part of the source code for different mobile platforms. Our approach supports and often requires further development activities after the source code generation, e.g., to integrate the generated source code into the already existing source code, or to extend the functionalities with platform-specific native code. We still believe that each software application requires some human contribution on the programming level. The goal is to cut down the required time to complete the tasks, to effectively support development efforts, but not to fully eliminate manual programming.

Further difference is that the presented multi-mobile platform solutions are providing hybrid applications, i.e. the applications are partly web applications and partly they are based on platform-specific libraries. Our solution produces truly native applications, therefore usually they are providing better performance, and usually it is easier to perform their testing and management.

The Vision Mobile Developer Economics study [33] states that it does not make sense for a startup to do native *development for multiple platforms, both in terms of time and money.* We agree, but if we provide the appropriate methods and tools, like the presented one, then the native development for multiple platforms can be available for small and medium sized companies as well.

Backend as a service (BaaS) [7] is a model for supporting mobile and web application developers with common functionalities as services. The services are provided via the use of custom software development kits (SDKs) and application programming interfaces (APIs). Providing a consistent way to manage backend data means that developers do not need to redevelop their own backend for each of the services that their applications need to access, potentially saving time and increasing the quality because reusing tested solutions.

Different BaaS solutions offer a slightly different set of backend services [28]. Among the most common services provided are user management, file storage and sharing, push notifications, integration with social networks, location services, messaging and chat functions, and running business logic. There are several BaaS providers, for example, Parse [25], cloudbase.io [6] and Buddy [5]. They mostly differ by the set of services provided, the platforms supported and the pricing, however, as they are constantly evolving, one can not make a general selection disregarding the actual development project.

The Parse cloud application platform supports iOS, Android, JavaScript, Windows 8, Windows Phone 8, and OS X platforms. Parse provides scalable backend solutions, push notifications, social integration, data storage, and custom logic possibility. cloudbase.io maintains and scales backend infrastructure including push notifications, database management and mobile analytics. Buddy provides the Buddy Development Platform and the Buddy Analytics Dashboard to support application development and service providing.

The method provided by this paper is not competing with BaaS solutions, but it is about to utilize them. Utilization means that some library functionalities are

realized with different BaaS services.

7 Conclusion

People use their mobile devices every day to access a wide variety of digital content. We have seen that the diversity of mobile platforms and that of mobile device capabilities requires providing applications for each different platform. The paper has provided a technology for developing multi-platform mobile applications. We have also introduced our model-driven solution for developing mobile applications for multiple mobile platforms. This approach increases both the efficiency of mobile application development and the quality of the resulting software artifacts. This is achieved by providing a mobile, platform-independent, high-abstraction level environment for mobile application design. We support it with innovative, mobile domain-specific languages and effective model processing solution.

We work on to support mobile application developers, i.e. we continuously extend the capabilities of the introduced approach and framework by covering more areas of mobile applications.

References

[1] Aczél, K. and Charaf, H. Automatic user interface code generation in Symbian, MicroCAD 2005, International Scientific Conference, Hungary, pages 1–5, 2005.

[2] Appcelerator platform homepage, *http://www.appcelerator.com*

[3] Android webpage, *http://www.android.com/*

[4] Angyal, L., Asztalos, M., Lengyel, L., Levendovszky, T., Madari, I., Mezei, G., Mszros, T., Siroki, L. and Vajk, T., Towards a fast, efficient and customizable domain-specific modeling framework, In Proceedings of the IASTED International Conference, pages 11–16, Innsbruck, 2009.

[5] Buddy, *http://www.buddy.com/*

[6] cloudbase.io, *http://cloudbase.io/*

[7] DZone's Definitive Guide to Cloud Providers, *http://www.dzone.com/page/comparison-guide-to-cloud-providers-2013*

[8] Fielding, R.T. and Taylor, R.N. Principled design of the mod-ern Web architecture, *ACM Trans. Internet Technol.* 2(2):115–150, 2002. DOI=10.1145/514183.514185 http://doi.acm.org/10.1145/514183.514185

[9] Firefox OS, *http://www.mozilla.org/en-US/firefox/os/*

[10] Foley, M.J. Microsoft's Windows Phone 8 finally gets a 'real' Windows core, *http://www.zdnet.com/blog/microsoft/microsofts-windows-phone-8-finally-gets-a-real-windows-core/12975*

[11] Fowler, M., Domain-specific languages, Addison-Wesley Professional, 2010.

[12] Gartner survey 2010, *http://www.gartner.com/it/page.jsp?id=1529214*

[13] The Gelmato Netsize Guide 2013: M-commerce on the move, 2013.

[14] Harrison, R. Symbian OS C++ for mobile phones: Program-ming with extended functionality and advanced features, John Wiley & Sons, ISBN 0470856114, 2004.

[15] iOS, *http://www.apple.com/ios/*

[16] Li, S. and Knudsen, J. Beginning J2ME: From Novice to Professional, Apress, ISBN 1-59059-479-7, page 480, 2005.

[17] Lengyel L., Levendovszky T. and Charaf H. Applying Multi-Paradigm Modeling to Multi-Platform Mobile Development, In Proceedings of the Workshop on Multi-Paradigm Modeling: Concepts and Tools. Nashville, USA, 2007.09.30-2007.10.05. pages 9–21.

[18] Lengyel L., Levendovszky T., Mezei G., Forstner B. and Charaf H. Towards a model-based unification of mobile platforms, In 4th IEEE/ACS International Conference on Computer Systems and Applications, Sharjah, 2006. IEEE, pages 866–873.

[19] Levendovszky T. Lengyel L. Mezei G. and Mszros T. Introducing the VMTS mobile toolkit, 3rd International Symposium on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2007, Kassel, Germany, 2007, pages 587–592.

[20] Madari I. and Lengyel L. Synchronizing user interfaces of different mobile platforms, International IEEE Conference Devoted to the 150-Anniversary of Alexander S Popov: EURO-CON 2009, Saint-Petersburg, Russia, 2009. New York: IEEE, pages 1852–1859. ISBN: 978-1-4244-3967-6.

[21] Mono Project homepage, *http://www.mono-project.com*

[22] OMG Model-Driven Architecture (MDA) specication, OMG document ormsc/01-07-01, *http://www.omg.org/*

[23] OMG UML specification, version 2.3, OMG document for-mal/2010-05-03, *http://www.uml.org/*

[24] Microsoft's Text Template Transformation Toolkit (T4), Code Generation and T4 Text Templates, *http://msdn.microsoft.com/en-us/library/vstudio/bb126445.aspx*

[25] Parse, *https://parse.com/*

[26] PhoneGap homepage, *http://phonegap.com/*

[27] Richardson, L. and Sam, R. RESTful web service, O'Reilly Media, 2007.

[28] Rowinski, D. The Rise of Mobile Cloud Services: BaaS Startups Grow Up, *http://readwrite.com/2012/04/17/mobile-backend-as-a-service-ec#awesm= oiUfyQ4lngjNcS*

[29] Rubino, D. Overview and review of Windows Phone 8, 2012, *http://www.wpcentral.com/overview-and-review-windows-phone-8*

[30] SOAP Version 1.2, *http://www.w3.org/TR/soap/*

[31] Thai, T. and Lam, H. .NET framework essentials, O'Reilly, 2003.

[32] Vision Mobile: Developer Economics 2013, *http://www.visionmobile.com/devecon.php*

[33] Vision Mobile, Developer Economics Q3 2013: State of the Developer Nation, 2013.

[34] Vision Mobile, The European App Economy, 2013, Creating Jobs and driving growth, 2013.

[35] Visual Modeling and Transformation System, *http://www.aut.bme.hu/vmts*

[36] Xamarin homepage, *http://xamarin.com/*

[37] Yao, P. and Durant, D. .NET compact framework program-ming with C#, Addison-Wesley Professional, 2004.

Received 13th January 2014