A METHODOLOGY FOR MODEL-DRIVEN MULTIPLATFORM MOBILE APPLICATION DEVELOPMENT

Hassan Charaf
Budapest University of Technology and Economics,
Department of Automation and Applied Informatics
Magyar Tudósok körútja 2., 1117 Budapest, Hungary
hassan@aut.bme.hu

ABSTRACT

Mobile devices are significant part of our daily life, we actively use them in communication, administration and to reach and consume digital data. The diversity of mobile platforms and the device capabilities necessitates developing the same functionality for each relevant mobile platform. Further issue of the mobile devices is their limited availability of battery power, i.e. mobile applications should strain after energy efficient solutions. The methodology provided by this paper addresses both of these issues. We provide a model-driven solution, where mobile applications are designed applying domain specific languages, and the executable artifacts are generated by domain specific model processors. These model processors are available for different mobile platforms. Therefore, from the same models the application can be generated for different mobile platforms. The generated source code utilizes both the cloud computing and platform specific energy efficient programing libraries. In this way the methodology supports to effectively realize energy efficient mobile applications for different mobile platforms.

Keywords: Multiplatform Mobile Applications, Energy Efficient Mobile Applications, Model-Driven Development for Mobile Platforms

1. INTRODUCTION

Based on different surveys [1] [2] the role of the mobile devices is determining the present and also the close future of the software industry. Mobile phone and tablet owners continually consume digital content. The diversity of mobile platforms and the mobile device capabilities requires providing automatic application generation for different mobile platforms. Mobile device owners would like to utilize the special capabilities of their own devices. Developing the same mobile application for all relevant mobile platform requires relevant development effort. The
main motivation of our research activity is to provide a model-driven solution and also address the issue of the energy efficiency, because of the limited resources of mobile devices.

The mobile industry has developed immensely in recent years. The convergence of various types of devices and technologies resulted in very complex and powerful handheld computers, which are on par with personal computers in regard to both performance and functionality. Compared to mobile environments, the development of a personal computer reflects less possible target platforms and the web-based thin clients (considering Flash, Silver light or Java plugins) are also more accepted than on mobile phones. When programming on a traditional desktop, server or web applications, it is usually enough to specialize yourself and/or your team for one specific platform. However, in the case of mobile development, the IT market is much more diversified. There are numerous device vendors with a variety of different application programming framework. Web-clients are also not as popular and less widely accepted on mobile devices. In addition to the different framework, vendors also prefer to distinguish themselves from their competitors with unique user interfaces and distinct application design. Consequently, no convergence can be expected in this area in the near future. Our objective targets one of the most pressing problems in the area of mobile software development, stemming mainly from the diversity of mobile platforms.

In Section 2 we discuss or model-driven methodology for multiplatform mobile application development. Section 3 introduces the applied techniques, namely the (i) domain specific modeling and model processing, (ii) mobile platform related energy efficiency considerations, and (iii) the role of cloud computing. In Section 4 a case study is provided that highlights the applicability of this approach. Finally, we conclude the paper.

2. THE METHODOLOGY FOR MULTIPLATFORM MOBILE APPLICATION DEVELOPMENT

In order to decrease efforts in maintaining the same functionality in many different mobile environments, the main objective of our approach is the cross-fertilization of three different R&D research areas: (i) model-based software development, (ii) mobile application development and (iii) cloud-based services. The motivation is to provide a model-driven environment for the development of reliable, multi-platform, mobile applications. The goal is realized with a modeling and model processing framework [3] that facilitates modeling different aspects of mobile applications in a platform-independent way, while integrating the use of cloud services and to provide automatisms that generate verified, ready-to-use mobile applications for different platforms using the same models.

Fig. 1. Developing high quality, energy efficient mobile applications with model-driven methods.
The architecture of the application generation process is depicted in Fig. 1. The modeling of mobile applications and the processing of these models are performed in a framework. We apply mobile domain specific languages to define the required application logic, data structure, communication and user interface. Mobile platform specific model processors are applied to generate the executable artifacts for different target mobile platforms. The generated code is based on the previously assembled mobile platform specific libraries. These programming libraries provide energy efficient solutions for mobile applications. Furthermore, some data processing or computation intensive tasks are passed into the cloud to save the battery power of mobile devices.

These issues include several open questions that we are actively researching. The following list introduces these issues and their nature.

1. We exhaustively compare the most popular mobile platforms (Android, iOS, Windows Phone, and further platforms), explore connection points and commonalities of the UI and business logic capabilities that may be handled in a uniform way. These common parts provide the base of further modeling and code generation methods. Moreover, formalisms and languages are expanded and capable of describing the common parts in a more precise way. These languages are able to integrate the use of cloud services into the business logic.

2. We perform laboratory measurements to analyze the different aspects of energy consumption of all important functions regarding each mobile platform. Based on the results, we design optimal software development patterns for using these functionalities more efficiently. The patterns encompass energy efficient network communication, location based services and multimedia features.

3. In order to simplify the complexity of the code generation automatisms, a typical practice is building a framework that can be utilized by the generated code. Our framework incorporates the main methods of the underlying platform and extends them through additional features. Furthermore, in using a framework as opposed to standard platform methods, energy efficiency can also be improved and the framework itself will become constantly more efficient.

4. To translate the formal models into executable artifacts, model-to-code transformations are utilized to perform these operations automatically. For each target platform, a separate transformation is used, since at this step we convert the platform independent models into platform specific executable code. The transformation relies upon the existence of the aforementioned framework and the generated artifacts use its methods. The transformation focus on energy efficiency: the generated source code is high-quality and optimized, does not consume more CPU time and therefore more energy than required. The generated applications utilize the capabilities of the cloud in order to enhance the energy efficiency of mobile devices.

To summarize, our objective targets one of the most pressing problems of mobile software development, which derives mainly from the diversity of mobile platforms. To address this problem, we analyze the mobile platforms from different perspectives. We provide a modeling language family for mobile applications and develop optimal frameworks for all platforms supporting the code generated from the models. The result is a complete system that allows designing mobile applications and generates working applications for each major mobile platform.

Next section discusses the different techniques we apply to support the introduced approach.
3. APPLIED TECHNIQUES

We have seen the motivation and the goals of our approach. Also we have discussed the main architecture and provided the building bricks of the whole solution. This section is dedicated to further introduce the techniques and considerations we applied in our solution.

3.1 Domain Specific Modeling and Model Processing

As the size of software systems tend to grow and software become more and more complex, the software development methods increase the abstraction level used in specifying the application logic. Instead of using classic object-oriented techniques, the trend nowadays is to create models and generate source code from them. The key concept behind model-based software methods is to express vital information in the model and let code generators accomplish the manual work of generating the code. This solution has two main requirements: the model must use a representation comfortable to express vital information; the model and the code generator together must contain all information required by the code generation.

Domain specific modeling [4] [5] and model processing address both requirements. Domain specific languages (DSLs) provide a viable solution for improving development productivity by raising the level of abstraction beyond coding while at the same time narrowing down the design space. Domain specific models are composed of elements representing concepts that are part of the problem domain world and from this world only. The specialization makes possible to express issues and their solutions efficiently by using the natural language of the domain. This has also an additional benefit: domain experts do not have to learn new (programming) languages, they can work on the domain language already well-known. By using DSLs, non-programmer experts can easily express their needs in a much more comfortable way as in case of universal modeling language (such as UML).

Domain specific languages can be textual (e.g. a script language) or graphical (e.g. a workflow language). Both kinds have their role, while complex algorithms are often easier to express by text, explaining the relation between different concepts are easier to understand in a visual representation. Usually, programmers prefer the textual form, while non-programmer domain experts feel the graphical representation much more comfortable. Modern frameworks providing domain-specific modeling should offer provide solutions for both kinds of DSLs.

Domain specific models are rarely the final product of a modeling scenario. We can reports, document templates, or statistics from models. Moreover, the specialization makes also possible to create a framework containing the base knowledge of the domain and generate code from models building on this framework. The final products are then automatically generated from these high-level specifications with domain specific code generators. Therefore, there is no need to make error-prone manual mappings from domain concepts to design concepts, or to programming language concepts. Moreover, we can create more than one code generator for a domain allowing us to create applications for multiple platforms (e.g. for multiple mobile platforms). In this scenario, platform independent domain models are translated by the platform dependent code generators to software products.

In general, three kinds of model processors are used: traversing processors, template-based processors and graph transformations. Traversing processors use the model, visit the model elements and generate code from them (e.g. [6]). Template-based techniques extend this behavior by adding a template document as the base of code generation. Graph
transformations [7] follow a whole different approach; they are basically highly configurable, smart search and replace operations. We specify a domain specific pattern to search for and a replacement pattern (e.g., GReAT [8] and VMTS [3]). It works like regular expressions, but has much higher expression power. Graph transformations are easy to use even for non-programmers.

In order to use it graph transformation in industrial environments, more scalable graph transformation techniques are required. Recent research directions include incremental overlapping rule application [9], utilization of multi-core architectures [10].

One of the most recent trends is to move both modeling and transformation into the Cloud [11]. This grants accessibility/stability for models and processing power for the transformations. The cloud environment can also be used to host the domain-based functions and generated applications can directly call the framework in the cloud thus saving processing power and memory (which is especially important in case of mobile applications).

Domain-specific modeling improves and accelerates the software development process. Due its advantages it is now widespread and it is expected to spread even more in the near future.

3.2 Mobile Platforms and Energy Efficiency Considerations

Both the mobile device and the mobile application industry have leading role in the current IT sector. Currently mobile devices are full featured computers that can effectively replace personal computers in regard to both performance and functionality. The different mobile platforms challenge the mobile software industry. It requires the knowledge of different platforms.

The development of mobile applications is one of the key areas of modern IT companies and services. In [4] it was forecasted that worldwide mobile application store revenue surpasses $15 billion. According to another estimation, the global mobile applications market is expected to be worth $25 billion in 2015. This also reveals that software development is going to cover highly the development of mobile applications.

Since there is no dominant platform at the moment on the market (as there is in case of desktop applications), the applications should be implemented usually for all the popular 3-4 platforms, while hardly anything can be reused from the code base of the one application in case of the same application on another platform. Also, the same testing procedures should be performed in each case, as the code base of the applications for the different platforms is separated. In [5] it has been shown that mobile software developers have to use an average of 3.2 platforms concurrently, which is a 15% increase compared to 2010.

The most significant shortcoming of all portable electronics also applies to mobile devices: their operation time is limited by their battery capacity. In general, two ways can be used to improve the operation time. The first one is to create hardware components that consume less energy and design batteries with higher energy density. The other one is to write software that uses the resources of the system more efficiently.

The main energy consuming components within mobile devices include CPU, display, I/O, and data transmission. Several researchers have proposed different kinds of system-level power models for mobile devices [12] [13]. These models attempt to describe the instantaneous power draw of the whole device as a function of other measurable performance attributes like CPU counters and display brightness level. Other studies have focused on measuring and modeling particular components of the device, such as the display and
wireless communication[14], or specific kinds of applications, such as peer-to-peer applications on mobile devices [15] [16].

These and other measurement and modeling efforts have led to important discoveries concerning how software-based techniques and optimizations can help in energy saving. For example, it has been shown that with wireless radio, including WLAN and 3G, the higher the bit rate, the more energy-efficient the communication is[17], which implies that smart scheduling is very important for energy efficiency. For instance, scheduling data traffic simultaneously with voice calls allows the data to be transmitted with little extra energy consumed.

3.3 Cloud Computing

Cloud computing has received significant attention recently. Companies may store their data and perform their computations off-premise in a highly available and scalable environment where they only pay for the resources that they actually use. Compared to traditional infrastructures that only use in-house resources, cloud computing has many advantages that have been transforming the computing solutions used at companies. Naturally, already existing on-premise resources can still be used as part of the infrastructure by connecting them to cloud services and forming a hybrid environment.

Cloud providers offer services on different abstraction levels. The most basic cloud service model is Infrastructure as a Service (Iaas), where simple virtual machines are offered. The virtualized environment provides a good base for scalability, and its simplicity means that developers can coordinate every aspect of their systems. Cloud service providers do not strictly follow these models, thus a provider may fit into several service models. Amazon Web Services (AWS) can be considered an IaaS provider as it is possible to rent machine instances and put any kind of operating systems on them. Platform as a Service (PaaS) providers deliver a computing platform with operating system, programming language execution environment and web server. An example is the Microsoft Windows Azure platform, where it is possible to install web applications onto systems having Windows, IIS, and .NET preinstalled. The third main model is Software as a Service (SaaS), in which providers install and operate an application in the cloud and cloud users can access these services. Examples are for instance the cloud based e-mail providers.

Cloud computing provides a cost-effective, highly-scalable solution for problems that include large amount of data or computation. Both of these appear in model transformation, where models can be arbitrarily large and transformation may be complex. Thus the weaving of model-driven engineering (MDE) and cloud technologies seems natural and may result in unprecedented performance boost in model-based tools. Several architectural options arise for MDE tools: (i) the data can be stored in the cloud [11] and transformations may be executed on the clients, (ii) on-premise data can be transformed in the cloud, (iii) both data and computation can be handled in the cloud, and (iv) a hybrid solution where both on-premise and cloud resources are used for either storage or computation. All of these options excel in different scenarios. For instance (i) can be considered if the transformations are simple, and thus, the communication overhead would be too high to run the transformations in the cloud. However, having the data in the cloud results in easily available collaborative work. On the other hand (ii) excels in transformations that can be easily distributed to several machines. Option (iii) provides a complete SaaS solution for model transformations. And finally, the hybrid solution may be used in situations where complex and simple transformations are mixed. In this case complex tasks should be handled in the cloud, and
simple ones should be executed locally to avoid the communication overhead. The approach is referred to as Modeling as a Service (MaaS) according to [18].

A completely different application area of cloud technologies is integrating online, cloud-based services into the executable application during code generation. Here we have two possibilities: either the language and the model describing the system contains explicit references to the cloud services (what, when and how to do with a cloud service, e.g. login to the cloud), or the cloud layer is completely hidden by the code generators, and the language describes only the expected behavior (e.g. login), not the way how to realize it.

Both of the above mentioned general ideas are productively incorporated into our approach.

4. CASE STUDY

The case study provides an example for a mobile application that is quite useful on all relevant mobile platforms. These types of mobile applications are within the focus of the presented methodology.

We can easily get into difficult situations in foreign countries, if we do not speak the national language. For example in a restaurant, it is usually important to understand exactly what we will eat. A few decades ago, printed dictionaries were the only solution, but nowadays they are often replaced by a dictionary application running on the mobile devices. Unfortunately, these dictionaries rarely support translation of a whole sentence, or complex expressions, they offer a word-by-word translation instead. This is not always enough. Another problem is to handle non Latin alphabets, e.g. to translate a Cyril, or an Arabic text, since it is hard to find out the word to translate. Our case study proposes a solution for this issue.

The mobile application offers a smart mobile dictionary. The application takes photos using the mobile device, recognizes the text on the photo and translates it into the preferred language. Since text recognition and translation require serious computational power, these functions are not executed on the mobile device directly, but they use services running in the cloud environment. The translator application is not free, users have to register in order to use the application and they have to pay a small fee (via PayPal) for each translation. The application logic is represented in Fig. 2.

![Fig. 2. Case study overview.](image-url)
The figure represents a domain specific model. This language can be used to define a set of similar applications. Shortly, we will see that by using the language, we can use typical elements of mobile application development on a high abstraction level. These elements can be simple or complex tasks, and they can be based on cloud services. For example, translation is a complex task. Now we investigate the model to understand its mechanisms.

The application starts with the Main Screen, which displays the main menu, with three options: Registration, Login & Use and Quit:

Quit menu item is rather simple, it closes the application. It is used to define an exit point only.

Registration starts with a welcome screen (Register) containing a standard registration how-to and a few marketing images. Next, we have to register via PayPal. The model does not describe the steps of this registration, since applications in our domain always register in the same way, using the same steps. For example, we have to log in to our PayPal account and confirm access to the application. The procedure does not depend on whether we want to pay for a translation application, a web warehouse, or anything else. Common procedures, like this are implemented in an underlying domain specific library. When generating code from the model above, we only have to generate a function call from this step and the framework will handle the rest. Thus, common tasks of the domain are not detailed in the model, the focus is on the unique, domain specific issues and their solutions.

If the PayPal-based registration is successful, the user receives a text message with login information. SMS sending is not part of the PayPal registration, domain applications may use it, but it is not necessary to do so. However, the technical details of sending an SMS are not elaborated in the model, it is implemented in the library.

The Login also starts with a welcome screen (Login). Next, we force the PayPal login and validate whether the PayPal account is empty. The PayPal-based login is also part of the underlying library. Therefore, the framework provides a common way to handle it. The next step however, is completely application specific. We take a photo by using the camera of the mobile device. This photo contains the text to translate. We do not process the photo on the mobile, but call a cloud-based OCR service (Optical Character Recognition service). Although modern smart devices could apply OCR algorithms locally, it is not optimal. By applying the text recognition in the cloud, we can save energy. This is beneficial for the environment as well. The OCR service finds the text on the photo and forwards the text to another cloud service, the Translator service. Here again, it is possible that we could translate the text locally, but forwarding this task to the cloud is more energy efficient. In case of cloud service calls, the domain specific framework allows us to specify the exact address of the services easily. This is the same principle as the one mentioned earlier: we hide technical details and focus on domain related questions. When the translation is ready, we pay. After paying, we can read the translated text. This “pay-first” approach makes cheating with the application more difficult. After reading the translation, the user can choose another target language (useful in a multi-language group) and the cloud service translates the source text again.

As we have shown, our domain language and the domain specific framework made possible to specify PayPal registration and login, or SMS sending very simple. Moreover it has simplified the usage of cloud services from mobile. We have successfully identified the common elements of our domain and built a language on these elements. By using these bricks and others such as welcome screens, we can build our application model easily and rapidly. For example the application logic of a package delivery system could also be
specified in a similar way. Moreover, since the language is rather specific to a domain, we can create efficient solutions for this language. Each domain model will use these solutions; therefore improving the efficiency affects all domain applications immediately.

Finally, note that the model is domain specific, but not platform specific. We can process and generate applications for multiple mobile platforms (e.g. Android, iOS and Windows Phone) based on the same application logic specified by the model. We create domain development environments (code generation templates, domain framework) for each platform and use these environments in model processing. We do not implement upgrades separately from the platforms. Changes in the application logic can be propagated into the applications by code generation.

5. CONCLUSION

Currently the development of mobile applications is one of the most important areas in the software industry. With the increasing number of different target devices the need for reliable as well as efficient applications is also rapidly growing. Since platform providers tend to distinguish themselves from the others, the same application must be developed and verified before publishing for each platform independently. Another drawback of mobile application development is the limited availability of battery power. A wasteful application or a badly designed algorithm may significantly shorten the availability time of the devices.

This paper has discussed a methodology that applying model-driven solutions facilitates the efficient multiplatform mobile application development. The resulted mobile applications are based on energy efficient mobile platform specific libraries. We have introduced the key techniques applied by the approach. These techniques are (i) domain specific modeling and model processing, (ii) mobile platform related energy efficiency solutions, and (iii) the cloud computing capabilities. We have also provided a case study to underpin the applicability of our mobile application development approach.

We believe that the role of the mobile devices and the need for native platform specific mobile applications are continuously growing, therefore, the relevance of the solutions, like the presented one, is high.

ACKNOWLEDGEMENT

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013).

REFERENCES


