Modeling the User Interface of Mobile Devices with DSLs

István Madari, László Lengyel, Tihamér Levendovszky

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann György tér 3, H-1111 Budapest, Hungary
{pityusz, lengyel, tihamer}@aut.bme.hu

Abstract: Developing to multiple mobile platforms meet difficulties, due to hardware incompatibilities, various programming languages, and implementations. Mobile communication devices evolved dynamically in the past few years and the number of operating systems and runtime environments are significant too. Developing to multiple mobile platforms or porting applications could be very expensive. Modeling based on Domain-Specific Languages, could provide a possible solution. By the help of metamodeling, and multi-paradigm modeling (MPM) we can develop user interfaces for different mobile platforms with the same functionality. This paper presents how metamodels should be developed for multiple mobile platforms. The presented metamodels and examples are developed in our metamodeling and model transformations system: Visual modeling and Transformation System (VMTS).

Keywords: Multi-Paradigm Modeling, Model-Driven Architecture

1 Introduction

Complexities of application development on various mobile platforms have become very difficult. The varied programming languages and hardware differences, incompabilities of mobile devices are the cause of complication. When the application needs to be run on other mobile platform, it is necessary to rewrite it. Model-based solutions help to reduce the time of development, creates an opportunity to port applications from a platform onto the other one, and the issues, caused by the different devices and platforms, could be solved.

To model a system, different aspects must be analyzed. All of the aspects should cover some functionality of the application, including the user interface, communication, underlying database, system architecture, and so on. Different abstraction levels could be chosen for the modeling: low-level abstraction for detailed models to generate applications, and higher level models to write system review or management purposes [1].

Current paper is driven by a case study, where the goal is to develop the same functionalities, starting with the user interface, for different mobile platforms. Metamodeling, model abstraction, model transformation are the key enablers for this complex problem.

2 Motivation

The cost of developing applications for different mobile platforms could be very high, due to compatibility issues. Using modeling technologies, lower development cost, and less development time could be achieved.

Domain-specific modeling (DSM) is an engineering methodology for developing software applications, and it can cover different abstraction levels [2]. The basic idea of DSM is that the platform commonalities and diversities are described in a domain-specific modeling language (DSML) that is based on the domain itself, and the final products are automatically generated from these models [1].

During the research, Java, Symbian, and .NET Compact Framework [7] have been chosen for our target platforms. User interfaces are modeled for these platforms. To achieve the goals, the target platforms are analyzed, and then the metamodels of the user interfaces are composed for each platform.

There are two possibilities how to build the metamodels for this purpose. The first solution contains one common metamodel. In this case, the metamodel must contain all user interface component from all of the platforms. When an instance is created, the user has to fill all attributes, and these attributes will cover the capabilities of user interfaces on all target platform. Due to the common metamodel the model size is huge (because all user interface element of all target platforms are contained), and really hard to maintain it. During the code generation, the system will check these attributes, and selects the corresponding attributes for the actual target platform. This method has some problems due to the issues of the different user interface concepts and structures (e.g. Symbian has a *Tabpage* based concept, but in Java, the *Tabpages* could not be found). The different attribute names and types are the other problem, implementation of these differences are circuitous.

The second solution is to create separate metamodels for each platform. In this case, the user should develop separate models for each platform. By the power of model transformation, we can define transformations between the models of the target platforms, and the user has to develop one user interface, then the rest of models will be generated automatically by model transformations. The Visual Modeling and Transformation System (VMTS) [3] is a modeling graph rewriting-based model transformation framework. The results of this paper are implemented within the VMTS. Our goal is to define metamodels for the target platforms, using the VMTS, and supply corresponding plugins for the development.

3 Backgrounds

Visual Modeling and Transformation System (VMTS) is our implemented medamodeling framework. It supports to create models based on metamodels, and it can execute model transformations between models. VMTS has plugin architecture. By the help of plugins the model elements could presented as known shapes of the current domain. The plugin is a DLL file, written in C#, and it follows the Model-View-Controller architecture. By using VMTS plugins, every model elements can be assigned to a graphical representation. It makes possible to use the visual editor as a user interface designer. The currently supported mobile user interfaces are depicted in Figure 1.

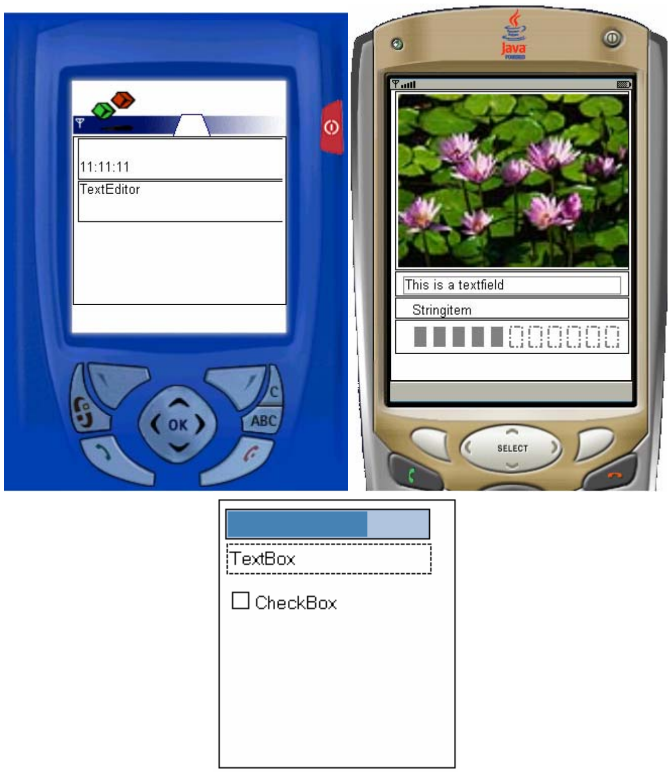

Figure 1
User interface design for (a) Symbian, (b) Java and (c) .NET platforms

VMTS provides model transformation functionality. The model transformation is based on graph rewriting, a powerful technique for graph transformation with strong mathematical background [1]. Rewriting rules must be given to define a transformation step. Rules are consisting of an LHS (left-hand side) and an RHS (right-hand side) graph, and the transformation engine finds an isomorphic occurrence of LHS in the source graph, and it replaces the matched atoms by the

RHS graph (an example LHS and RHS rules depicted in Figure 2). The example rewriting rule is transforming *Symbian TabPages* into *JAVA* forms. The connection between the LHS and RHS graphs is realized by internal causalities and the attribute transformations are realized by imperative OCL [5] scripts. By imperative OCL scripts we can define modifications, removing steps, or create new elements.

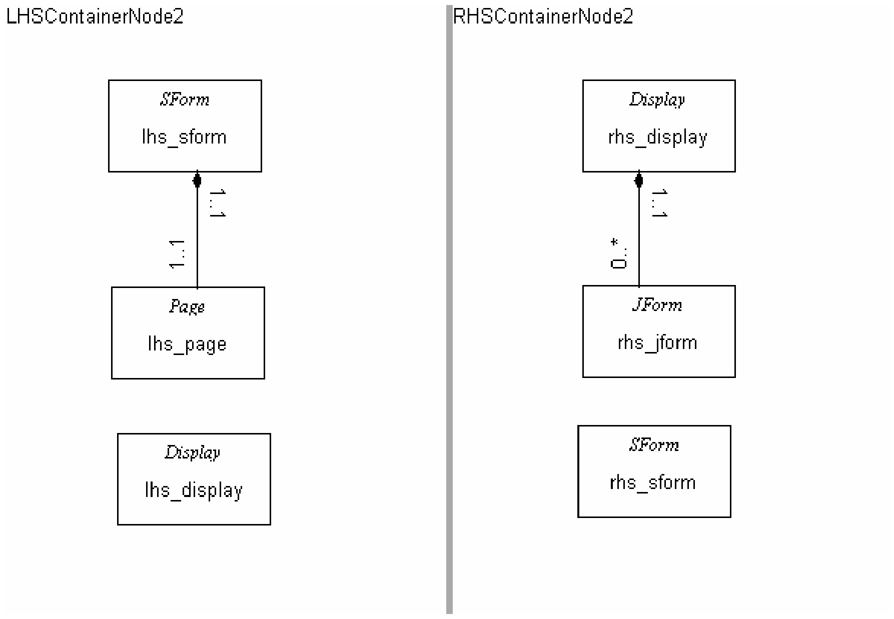

Figure 2
An example rewriting rule

The metamodel defines the abstract syntax of the target platform, and the user will develop a model, based on this metamodel. By the developed model, the framework can build a CodeDOM [6] model, which supports source code generation. The CodeDOM model is hierarchical structure, and it contains a language independent representation of the classes, attributes to generate.

The VMTS framework contained a common metamodel to define the user interface resources for mobile platforms. This common metamodel was not efficient, because incompatible issues of the target platforms, and the diversities of the used user interface structures.

4 Contributions

The common metamodel for the mobile platforms defines all of the user controls, which occurs on any of the supported mobile platforms, but the above-mentioned (in Sections 2 and 3) reasons inspired us to improve the existing metamodel, and apply a better method to define the abstract syntax of the target platforms.

The method is to develop separated metamodels, and each metamodel belongs to a target platform. In this case the common metamodel does not exist anymore. The code generation, and the model building are based on these separated metamodels too. The separated metamodels are illustrated in Figure 3.

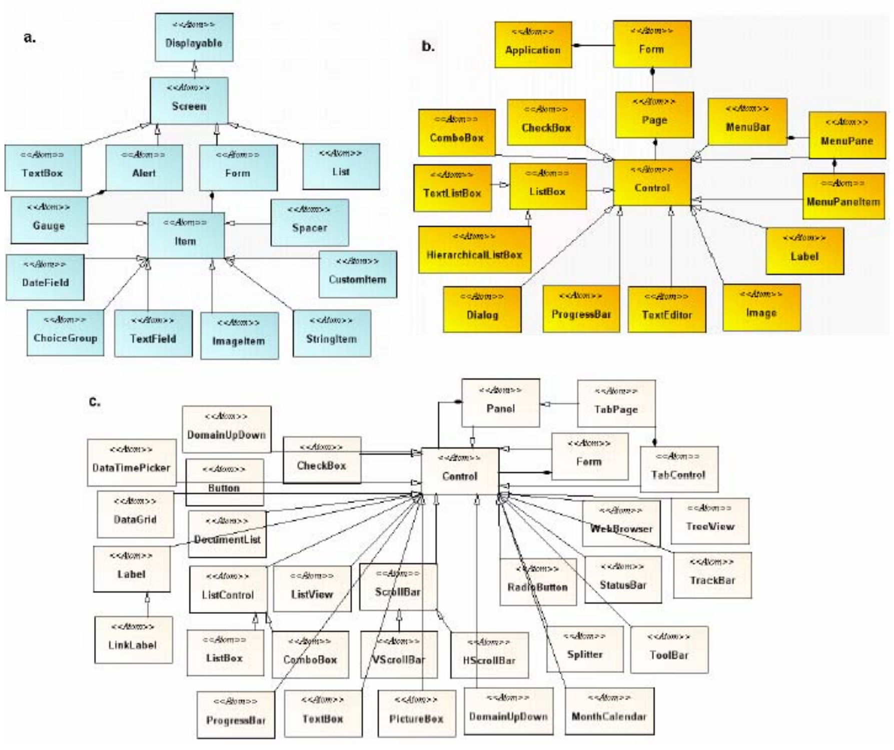

Figure 3
The separated metamodels. (a) Java, (b) Symbian, (c) .NET Compact Framework

By the separated models, the users can develop user interfaces for just one target platform. The user has to choose which platform is the most familiar for him, and he develops user interfaces for that platform. The other user interfaces will be generated using model transformations.

The separated metamodels has many advantages: easy to maintain, there are no large numbers of controls, and the attribute names are matched to attributes of classes of the target platform. The diversities of the target platforms will be hidden from the user: they will be solved by the model transformation. Opposite the common metamodel the user does not have to fill all attributes of all platforms, the user has to work on just one model.

Then again we have to develop model transformations between the different models. These model transformations could be very difficult.

Conclusions and Future Work

We have provided a case study, where metamodels are defined for the target platforms. The incompatible structures, user interfaces, diversities of the target platform will be superable by the graph transformations. The same control is not accessible on all platforms, but we have to develop the same services on the different platforms. For example, Symbian using *Tabpage* concept on user forms, but in Java we can not find *Tabpages*. The graph transformation has to convert the tab pages into forms, and it has to create some navigation functions. The goal is to develop the same service on different mobile platforms.

Acknowledgement

The fund of 'Mobile Innovation Centre' has supported, in part, the activities described in this paper. This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1] Lengyel, L., T. Levendovszky, H. Charaf, Applying Multi-Paradigm Modeling to Multi-Platform Mobile Development, 2007

[2] Sztipanovits, J. and G. Karsai, Model Integrated Computing, IEEE Computer, 1997

[3] VMTS Homepage: http://vmts.aut.bme.hu

[4] OMG MDA: http://www.omg.org/docs/bei/02-06-08.pdf, 2003

[5] OMG, Object Constraint Language, http://www.omg.org/docs/ptc/03-10-14.pdf

[6] Roger, R. Generating Design Patterns using CodeDOM, 2002

[7] Wei Meng L., B. Jepson, Programming the .NET Compact Framework, 2007

[8] Vangheluwe, V. and J. de Lara, Computer automated multi-paradigm modelling: Meta-modelling and graph transformation, Winter Simulation Conference, 2003, pp. 595-603

[9] Lengyel, L., T. Levendovszky, G. Mezei and H. Charaf, Model Transformation with a Visual Control Flow Language, International Journal of Computer Science, 2006 (I), pp. 45-53

[10] Gamma, E., R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable Object- Oriented Software”, Addison-Wesley, 1995

[11] Ehrig, H., G. Engels, H. J. Kreowski, G. Rozenberg, “Handbook on Graph Grammars and Computing by Graph Transformation: Application, Languages and Tools”, World Scientific, 1999 (Vol. II), Singapore