MDA Based Multiplatform Mobile Application Modeling with Platform Compliant User Interfaces

ĽUBOŠ STARÁČEK[1]
VALENTINO VRANIĆ[2]

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Bratislava, Slovakia
[1]lubostar1@gmail.com
[2]vranic@stuba.sk

Abstract. Applications for mobile devices (mobile applications) represent a specific segment of the software market in which development of applications for multiple platforms is far more articulated issue than in applications intended for common computers. While multiplatform mobile application development tools, such as Marmalade, MoSync or Xamarin, generate quite usable (software) platform specific code out of its general representation developed upon something that might be considered as a super-platform, the user interface exhibits peculiarities that have to be addressed manually. Otherwise, the user interface will probably fail to meet the given platform compliance criteria that may result in worsening user acceptance of the application or even in not being accepted to the application marketplace at all. In this paper, an approach to design multiplatform mobile application at model level that employs OMG's MDA (Model Driven Architecture) to generate platform compliant user interfaces while still taking advantage of multiplatform tools to develop application logic is proposed. Navigation in mobile application user interfaces is modeled using UML state machine diagrams. A model-to-model transformation for the Android platform has been created and applied to a real application model.

Keywords: mobile applications; user interface; multiplatform; UML; MDA; state machine diagrams; software product lines.

(Received August 28, 2014 / Accepted January 20, 2015)

1 Introduction

Applications for mobile devices—commonly known as *mobile applications*—represent a specific segment of the software market in which development of the applications for multiple platforms is far more articulated issue than in applications intended for common computers. The notion of platform is usually being related to operating systems and software frameworks, which are very diverse on mobile devices, but it can comprise mobile device hardware properties, too. In other words, we can distinguish between software and hardware platform. Some software platforms are capable of running on different hardware platforms.

The migration to another platform is sometimes achieved by adapting the application, but if the need to run the application on several platforms is known from the beginning, it's possible to proceed deliberately and employ appropriate techniques to what is called *multi-platform*[1] mobile application development.

Hardware platform variability, such as display resolution or the very presence of some hardware components like camera, keyboard, GPS, gyroscope, or even SIM card, seems to be well managed by application

[1]also known as *cross-platform*

code. Software platform variability comprises dealing with application logic and user interface. Nowadays there is a huge expansion of quite different multiplatform development tools. They differ in the technology approach, e.g. web-to-native wrappers, runtime execution environment, source code translators, supported platforms, or target audience [5, 10].

While multiplatform mobile application development tools, such as Marmalade, MoSync or Xamarin, generate quite usable (software) platform specific code out of its general representation developed upon something that might be considered as a *superplatform*, the user interface exhibits peculiarities that have to be addressed manually. Otherwise, the user interface will probably fail to meet the given platform compliance criteria that may result in worsening user acceptance of the application or even in not being accepted to the application marketplace at all.

In this paper, an approach to design multiplatform mobile application at model level that employs OMG's MDA (Model Driven Architecture) to generate platform compliant user interfaces while still taking advantage of multiplatform tools to develop application logic is proposed. The rest of the paper is structured as follows. Section 2 presents the outline and context of the approach. Section 3 describes in details the role of MDA in the approach. Section 4 deals with navigation modeling in user interfaces. Section 5 brings some evaluation results. Section 6 discusses related work. Finally, Section 7 concludes the paper and proposes further work.

2 Multiplatform Mobile Application Modeling: The Approach Overview

The multiplatform mobile application development tools allow to develop mobile applications for multiple platforms simultaneously. Thanks to these tools it is possible to design improved multiplatform mobile application architecture. This architecture employs one multiplatform application core on all platforms in combination with the platform specific user interface. This architecture, depicted in Figure 1, requires a minimum of source code and can fulfill requirements on adaptation of the user interface according to conventions that user interfaces must adhere to in order to be platform compliant.

The approach to multiplatform mobile application modeling proposed in this paper is depicted in Figure 2. The first step consists of designing a platform independent model (PIM) of the multiplatform mobile application being developed including navigation. In the next step, this PIM is processed by the corresponding model-to-model (M2M) transformations resulting in a

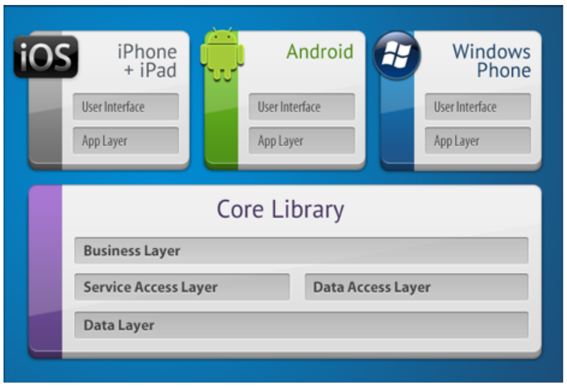

Figure 1: Architecture of multiplatform mobile application (adopted from [17]).

platform specific model (PSM) generated for each target platform.

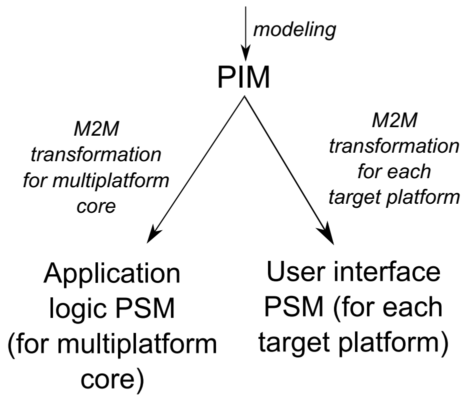
modeling
PIM
M2M transformation for multiplatform core
M2M transformation for each target platform
Application logic PSM (for multiplatform core)
User interface PSM (for each target platform)

Figure 2: An overview of the approach to multiplatform mobile application modeling.

The approach fits into a broader context as depicted in Figure 3. This assumes model-to-text transformations for each target platform to generate user interface source code on one hand, including navigation, while on the other hand, this code has to be merged with the platform specific code generated from the multiplatform application logic implementation by the appropriate mobile application multiplatform tool.

The following two sections explain the details of user interface and application logic structural modeling in the context of MDA (Sect. 3) and modeling the navigation aspect of user interfaces Sect. 4.

3 Employing MDA to Achieve Platform Compliant User Interfaces

This section presents the details of employing MDA to achieve platform compliant user interfaces. The pos-

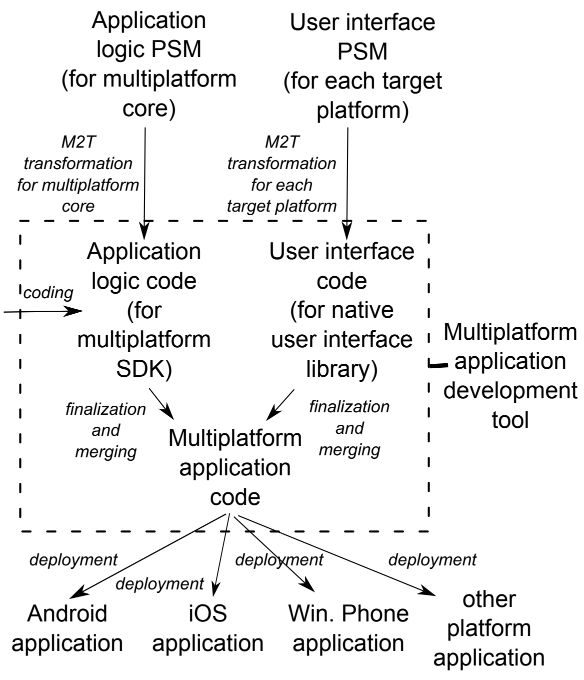
Application logic PSM (for multiplatform core)
User interface PSM (for each target platform)
M2T transformation for multiplatform core
M2T transformation for each target platform
Application logic code (for multiplatform SDK)
User interface code (for native user interface library)
coding
Multiplatform application development tool
finalization and merging
finalization and merging
Multiplatform application code
deployment
deployment
deployment
deployment
Android application
iOS application
Win. Phone application
other platform application

Figure 3: The context of the approach.

sibility to apply MDA to application logic modeling is embraced, too.

3.1 MDA

MDA (Model Driven Architecture), an Object Management Group standard, targets multiplatform model driven application development in general. MDA starts with a platform independent model (PIM) which is being adapted to the target platform through a series of transformations that result in creating one or more platform specific models (PSM).

Multiplatform mobile application development tools enable to implement platform specific user interfaces that meet the given platform compliance criteria by providing access to native user interface libraries. However, this means that the user interface for each platform has to be implemented separately. Here, we employ MDA to design the user interface once for multiple platforms and then to generate the corresponding platform specific models.

At the platform independent model level, the application is modeled without any platform specific properties. The application elements are relieved of the technical details of their realization. The PIM elements are marked with stereotypes from the UML profile that defines their basic semantics. The elements can be further configured by adding special attributes to them bearing the *arg* stereotype. Marking elements with stereotypes and providing them with the *arg* stereotyped attributes defines (partly) how these elements are to be transformed into a PSM by an M2M transformation.

QVT (Query/View/Transformation) is a standard defined by the Object Management Group for model transformations. The approach proposed here employs Eclipse M2M Operational QVT implementation.

An M2M transformation has to be designed for each target mobile platform including the transformation for the multiplatform development tool being used in the implementation phase since each tool is different.

3.2 UML Profile

The UML profile designed for the purposes of the approach to multiplatform mobile application design proposed here contains stereotypes applicable to the instances of the *Class* UML metaclass (Figs. 4 and 5) and to the *Property* UML metaclass (Figure 6).

For example, one of the stereotypes applicable to the *Class* metaclass is *ui View*. The classes in PIM that model the base user interface usage should be marked with this stereotype. In the M2M transformation for the Android platform, all the classes marked as *ui View* would be transformed into the classes that extend the Android *Activity* class with the addition of the corresponding methods like *onCreate()*, *onResume()*, *onPause()*, etc. In the iOS transformation, the transformation would end up with the *UIViewController* class, while in Windows Phone transformation the *PhoneApplicationPage* class would be employed.

One of the stereotypes applicable to the *Property* metaclass is *ui ListView*. This stereotype represents a list of items that appear in the user interface and it should be applied to the property of some class that represents a user interface (e.g. *ui View*). On Android, this would be represented by an instance of the *ListView* class. On iOS, *UITableView* would be used, while on Windows Phone *ListBox* would be employed.

All the elements marked with the stereotypes related to the user interface would be adapted to the corresponding target platform. Thus, these user interface elements are to be transformed by the Android, iOS, Windows Phone, or some other platform transformation. The elements that are part of the multiplatform core library (services) need not be adapted, so they would be transformed only by the multiplatform transformation. The multiplatform core library consists of services such as database service, navigation service, internal storage service, and so on. These services have their respective stereotypes, e.g. *database Service*, *navigation Service*, or *internalStorage Service*.

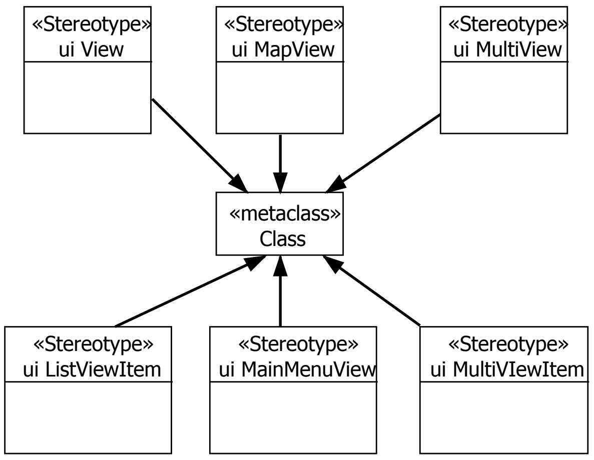
«Stereotype» ui View
«Stereotype» ui MapView
«Stereotype» ui MultiView
«metaclass» Class
«Stereotype» ui ListViewItem
«Stereotype» ui MainMenuView
«Stereotype» ui MultiVIewItem

Figure 4: The UML profile with stereotypes applicable to the *Class* UML metaclass instances that constitute the user interface.

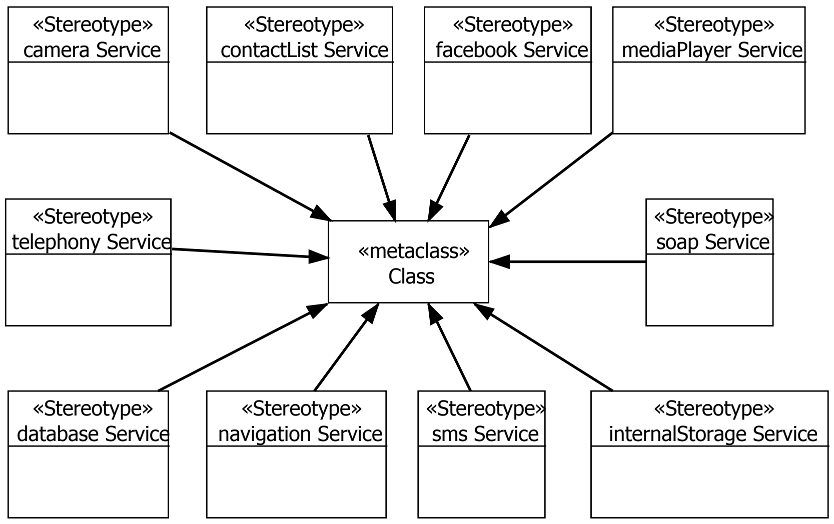
«Stereotype» camera Service
«Stereotype» contactList Service
«Stereotype» facebook Service
«Stereotype» mediaPlayer Service
«Stereotype» telephony Service
«metaclass» Class
«Stereotype» soap Service
«Stereotype» database Service
«Stereotype» navigation Service
«Stereotype» sms Service
«Stereotype» internalStorage Service

Figure 5: The UML profile with stereotypes applicable to the *Class* UML metaclass instances that constitute multiplatform services.

3.3 Element Configuration

Some user interface elements could be transformed into PSM in several ways. Consider the *ui MultiView* stereotype, which represents an element that includes multiple views. On the Windows Phone platform, this element would be transformed into an element called *Panorama View*. This element may contain a main menu, but it doesn't have to since on the Android and iOS platforms the main menu should never be a part of a *ui MultiView* element. To be able to deal with this situation, the *ui MultiView* element should be configurable. The configuration is performed by adding *arg* stereotyped attributes into the component. Thus, if a *Panorama View* should include a main menu on the Windows Phone platform, then a *MainMenu* type attribute named *true* bearing a *arg* stereotype would be added into the PIM element marked with the *ui MultiView* stereotype that would subsequently have to be handled in the Windows Phone PSM transformation.

Services at the PSM level may offer more functionality than needed for the given purpose. Therefore, these services should be configurable, too. Service configuration should be performed in the same way as user interface element configuration: by adding *arg* stereotyped attributes into corresponding elements. Consider the navigation service that may use the GPS module and/or network connection to achieve navigation. Each navigation method would have its own *arg* stereotyped attribute that would indicate the presence of the given navigation method at the implementation level. Thus,

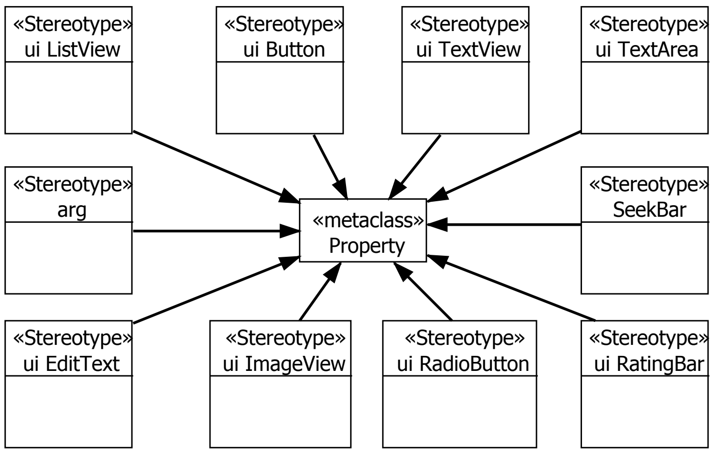

Figure 6: The UML profile with stereotypes applicable to the *Property* UML metaclass instances.

for example, adding the *GPSLocation* type attribute named *true* bearing the *arg* stereotype to the *navigation Service* element will indicate that the *getLocation()* method should be implemented and that it would return the location given by the GPS module.

3.4 User Interface Stereotypes

All defined user interface stereotypes are listed here. These are the stereotypes applicable to the *Class* UML metaclass instances that form the user interface and what they are used to denote:

- *ui View*—the base view of the user interface hierarchy, already described in this section
- *ui MapView*—the view that contains a map, each major mobile platform vendor (Google, Apple, Microsoft) has its own implementation of maps
- *ui MultiView*—the view that contains multiple views, displays one view simultaneously and can switch between these views; on some platforms, various realization options are possible for this element (e.g. sliding views with or without text page indicator or tabbed views on Android etc.), so a special argument that would enable switching these options would be needed
- *ui MultiViewItem*—a view that is contained in *ui MultiView* element, should be connected with the corresponding *ui MultiView* element using aggregation
- *ui MainMenuView*—very similar to *ui View*; the only reason for defining this stereotype is that on Windows Phone the main menu may be a part of the *ui MultiView* element, but on Android and iOS it can't
- *ui ListViewItem*—a view that is forming a single item of a list displayed by *ListView*; it should be connected with the corresponding element that includes *ui ListView* with aggregation; if the corresponding element contains several *ui ListView* fields, then *ui ListViewItem* element should contain a special attribute that would identify the corresponding *ui ListView* instance (e.g. «*arg*»list1 ListViewID)

These are the stereotypes applicable to the *Property* UML metaclass instances and what they are used to denote:

- *ui ListView*—view that shows items in a scrolling list, may be horizontal or vertical; displayed items are represented by a *ui ListViewItem* element
- *ui Button*—a standard button view
- *ui TextView*—a standard text view
- *ui TextArea*—a text view that contains multiple lines of text
- *ui SeekBar*—a visual progress indicator in the operation while the user can touch the thumb and drag left or right to set the current progress level
- *ui RatingBar*—an extension of *ui SeekBar* that shows a rating in stars
- *ui RadioButton*—a standard radio button
- *ui ImageView*—a view that displays an image

- *ui EditText*—an extension of *TextView* that is editable by the user

- *arg*—a special attribute

All the instances to which any of the stereotypes listed is applied, except for the *arg* stereotype, are user interface elements.

These are the stereotypes applicable to the *Class* UML metaclass instances that represent service elements and are part of the multiplatform core of the mobile application being developed, as well, as what are these stereotypes used to denote:

- *camera Service*—a service that handles the mobile device camera

- *contactList Service*—a service that handles access to the contacts stored in the mobile device

- *facebook Service*—an access to the Facebook API

- *mediaPlayer Service*—handle playing of audio/video media files

- *soap Service*—a service element that would implement web services using SOAP protocol

- *internalStorage Service* Handle data stored in internal storage of a mobile device

- *sms Service*—handle SMS managing (e.g. read, write, send, delete, etc.)

- *navigation Service*—handle a GPS module and navigation generally (e.g. navigation by network)

- *database Service*—a service element that would implement access to the database

- *telephony Service*—handle telephony services (e.g. making a call, getting signal strength, etc.)

Consider an example of a PIM depicted in Figure 7 that models a simple application displaying location coordinates of a mobile device containing a label and the closing button. Suppose this PIM has to be transformed into an Android PSM by the Android M2M transformation. The user location is obtained by *navigation Service* mentioned earlier in this section.

After processing this PIM by the Android QVT transformation created for the purposes of the approach proposed here, the Android PSM would be generated as depicted in Figure 8.

A snippet of the Android QVT transformation that is responsible for the transformation of the given example is depicted in Figure 9.

```
modeltype UML uses "http://www.eclipse.org/uml2/2.0.0/UML";
transformation Android_PSM_transformation
(in model : UML, out model1 : UML);

main() {
-- map all model elements
      model.rootObjects()[Model]->map Model();
}

mapping Model::Model() : Model {
-- concatenate "Android" to model name
      name := self.name + 'Android';
-- query all model elements of type "Class" with stereotype
-- that starts with "ui View" string and process them with
-- UIView mapping
      ownedType += self.getOwnedTypes()[Class]
      ->select(c|c.getAppliedStereotypes()
      ->exists(s|s.name.startsWith("ui View")))
      ->map UIView();
}

mapping Class::UIView() : Class {
      name := self.name + 'Activity';
      ownedAttribute += self.attribute
      ->map attributes(); -- map all attributes
      ownedOperation += object Operation{
            name := 'onCreate';
      }; -- add "onCreate" method

      ownedOperation += object Operation{
            name := 'onDestroy';
      }; -- add "onDestroy" method
}

mapping Property::attributes() : Property {
      name := self.name;
-- resolving type of attributes by stereotypes
      if(self->exists(c|c.getAppliedStereotypes()
      ->exists(s|s.name.startsWith("ui Button"))))
      then {
            type := object Class{
                  name := "Button";
            };
      } endif;

      if(self->exists(c|c.getAppliedStereotypes()
      ->exists(s|s.name.startsWith("ui TextView"))))
      then {
            type := object Class{
                  name := "TextView";
            };
      } endif;
}
```

Figure 9: A snippet of the Android QVT transformation.

The M2M transformation should preserve the stereotypes. This is important for the code generation, which itself is out of the scope of this work.

The M2M transformation should also preserve the elements that represent multiplatform services, even though they are not transformed by it. The purpose of this is to make possible for the developer to observe how platform specific elements are connected to the elements contained in the multiplatform core. Also, with this, the developer would be able to run a mobile platform M2M transformation on the PIM first, and then a multiplatform tool M2M transformation on the PSM obtained from that mobile platform M2M transformation. This process would produce a complete PSM. For

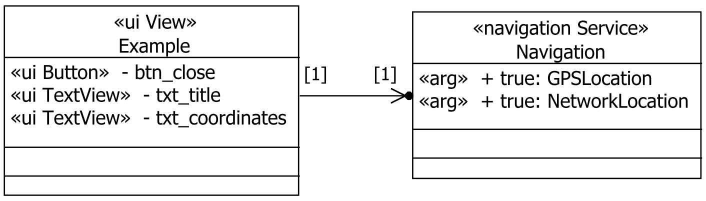
«ui View»
Example
«ui Button» - btn_close
«ui TextView» - txt_title
«ui TextView» - txt_coordinates
[1]
[1]
«navigation Service»
Navigation
«arg» + true: GPSLocation
«arg» + true: NetworkLocation

Figure 7: An example of a PIM.

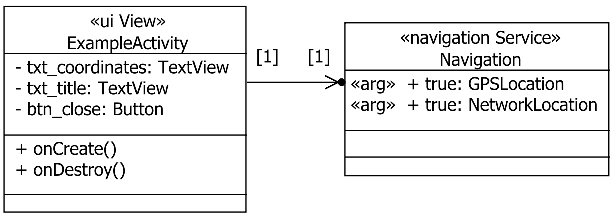

Figure 8: An Android PSM.

example, one might run the Android M2M transformation on the PIM, and then run the multiplatform tool M2M transformation on the obtained Android PSM resulting in a complete PSM.

4 Mobile Application Navigation Model

In the previous section we focused mainly on the mobile application structure. A very important aspect of the mobile application user interface design is navigation. Navigation can successfully be modeled using UML state machine diagrams [4] and we employ this approach to model navigation in mobile application user interfaces.

In the navigation model, each *ui View* element is modeled as a state of the state machine diagram where names of the states and the corresponding *ui View* elements must match. A state machine diagram transition represents the user interface element instance that initiates navigation change between *ui Views* connected with this transition. Therefore, *ui Views* must contain all the given user interface element instances that are represented as transitions coming out of the state that represents the given *ui View* element.

The transition syntax in UML state machine diagrams is [11]:

> *event[guard expression]/action*

User interface elements are mapped onto transitions by setting an event of the given transition to be the same as the name of the user interface element.

Each user interface element can handle multiple input events that can initiate different navigation changes. Therefore, navigation model has to include the information about which input event (*onClick*, *onLongCLick*, *onKeyPressed*, etc.) initiates the given navigation change. The user interface elements in the navigation model would not initiate other actions than navigation changes, thus input event can be put into the navigation model as an action of given state machine diagram transition that represents the given user interface element.

On different platform user interface elements may initiate different navigation changes by their input event, which makes navigation model platform specific. This opens two possibilities: to make one navigation model for each target platform or to express all navigation transitions in one model and distinguish their platform adherence with tagging. Assuming that the most of the navigation transitions would be shared between target platforms, the second possibility is more appropriate. A platform specific transition would be tagged by setting a precondition in the guard expression in it as:

> *platform==<target_platform>*

To sum up, a transition in the navigation model would be described as:

> *user interface element's name[target platform]/input event*

Figure 10 shows an example of the navigation model. This example includes two states named *MainMenuView1* and *MainMenuItemView1*. Therefore, the PIM should contain the elements with these names. Such an element must be able to contain and display user interface elements, thus an appropriate stereotype, like *ui View*, *ui MainMenuView*, or similar, has to be applied to it. There is a transition from the *MainMenuView1* state to the *MainMenuItemView1* state. This transition is specified as:

btn_menu_item1[platform==iOS]/onClick

Therefore, *MainMenuView1* should contain a user interface element named *btn_menu_item1* that—when clicked—would initiate displaying *MainMenuItemView1* on the iOS platform.

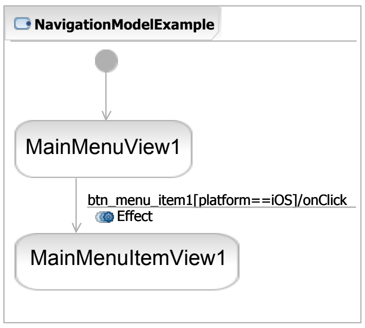

Figure 10: A user interface navigation model.

5 Evaluation

The approach to mobile application modeling proposed here was demonstrated on several—so to say—fabricated examples in previous sections. However, the approach was also applied to a real mobile application case. This application is a generalized version of a real application developed by the iNeed developer team.[2] The original application was intended to be an "encyclopedia" of beer and breweries located in Slovakia and Czech Republic. For the purposes of this work, the application was adjusted to be able to cover any products and its user interface was simplified.

The Android M2M transformation proved its usability and ability to generate arbitrary PSM from PIM, which speaks in favor of the possibility to define M2M transformations for any mobile platform. In the Android M2M transformation, an element with the *ui MultiView* stereotype produced multiple elements that form *ui MultiView* on the Android platform. Also, all other elements specific to the Android platform were transformed into their corresponding platform specific form. Without the proposed approach, this would have to be performed manually.

Extending the proposed approach to a new platform requires designing and implementing the corresponding M2M transformation for this platform. However, if the platform is not supported by the multiplatform development tool that is intended to be employed in the implementation phase, there are two possible solutions. The first one is to change the multiplatform development tool to one that would support all required platforms. Since changing the tool may require adaptation of existing M2M transformations, this option may be too expensive, but if such M2M transformations are available, this possibility is recommended.

The other possibility is to write an M2M transformation that would generate the application model on a new platform in its native API. This would require:

- the M2M transformation to change the architecture of the application in PSM from multiplatform to single platform, so it would transform services to a platform specific form
- to use its integrated development environment (e.g., Xcode for iOS) in the implementation phase

6 Related Work

A multiplatform mobile application designed according to our approach can be considered to be a software product line. Although common software product lines represent different software products with shared components among them and with some product specific components, the software product line here is constituted by one product adapted to multiple platforms. However, there are shared components in the multiplatform core library and product specific components that represent the user interface. Several authors have addressed the software product line approach in mobile application domain [1, 8, 14, 16]. One of the mostly discussed topics is how high heterogeneity in mobile platforms may lead to a significant increase of software variability. Improved multiplatform mobile application architecture that employs the software product line approach also brings advantages like decrease in time to market, increase in productivity, and quality improvement [15].

Dolog and Nejdl [4] proposed an approach for generating navigation sequences in web based systems. In this paper, Dolog's idea of navigation modeling using UML state machine diagrams has been applied and adapted to the domain of mobile applications. Specifically, it was necessary to design methods for marking target platform and add the event type into state machine diagram transitions (described in Sect. 4).

Myllymaki et al. explored variability and commonality management in the spirit of software product line as a solution for device fragmentation on the Symbian

[2] `http://ineed.sk/`

mobile platform [9]. They employ software product lines as an additional layer of abstraction to cover various hardware properties of mobile devices on single platform, while our approach employs software product lines to deal with the heterogeneity of mobile platforms since device fragmentation is successfully managed by application code.

Balagtas-Fernandez et al. [3, 2] provided a tool called Mobia Modeler that allows the development of fully functional software applications for mobile platforms by employing MDA. Mobia Modeler has a user friendly user interface, which make developing process more intuitive and it also prevents users from performing invalid actions. The tools employs MDA with code generation techniques (XMLT) to improve time to market and development complexity.

Porubän et al. [12, 13] reported they created the Graphical User Interface Interaction Language using their annotation based parser generator that significantly simplifies development of domain-specific languages. This could be seen as an alternative to graphical modeling employed in our approach.

7 Conclusion and Further Work

In this paper, an MDA based approach to multiplatform mobile application modeling with platform compliant user interfaces has been proposed. The approach addresses the problem of inadequate support for platform specific user interface features in existing multiplatform mobile application development approaches.

The approach employs MDA's platform independent models (PIM) to express a multiplatform user interface along with a part or whole application logic (optional). Platform independent models are marked with general user interface and application logic stereotypes contained in the UML profile for modeling multiplatform mobile application proposed in this paper.

Both user interface and application logic elements can be configured by adding the corresponding attributes bearing the *arg* stereotype. By this, the UML profile for modeling multiplatform mobile application remains relatively stable.

Navigation represents an important aspect of user interface modeling. For this, state machine diagram based approach known from web modeling [4] has been applied.

Each target platform has to be covered by the corresponding model-to-model transformation that transforms a platform independent model to a platform specific one (PSM). For this, QVT can be used and it was actually used to demonstrate the approach: a model-to-model (M2M) transformation for the Android platform was developed.

The approach is presented on examples. Furthermore, it was successfully applied to create a platform independent model of a real mobile application to which the proposed Android M2M transformation has been applied to get the corresponding platform specific model. This has proved the applicability and usability of the Android platform M2M transformation. The situation with a missing M2M transformation for a given platform has also been evaluated (see Sect. 5).

The context in which this approach to multiplatform application modeling is proposed—presented in Figure 3 (Section 2)—embraces code generation by model-to-text MDA transformations. This is certainly one way to extend the approach. Another area we would like to target is providing a common user interface to mobile devices participating in complex event processing [6, 7].

Acknowledgments

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12.

This contribution/publication is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] Alves, V. Identifying variations in mobile devices. *Journal of Object Technology*, 4(3):51–56, Apr. 2005.

[2] Balagtas-Fernandez, F., Tafelmayer, M., and Hussmann, H. Mobia Modeler: Easing the creation process of mobile applications for non-technical users. In *Proceedings of 15th International Conference on Intelligent User Interfaces, IUI '10*, pages 269–272, Hong Kong, China, 2010. ACM.

[3] Balagtas-Fernandez, F. T. and Hussmann, H. Model-driven development of mobile applications. In *Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE '08*, pages 509–512, L'Aquila, Italy, 2008. IEEE Computer Society.

[4] Dolog, P. and Nejdl, W. Using UML and XMI for generating adaptive navigation sequences in

web-based systems. In *Proceedings of 6th International Conference on the Unified Modeling Language, UML 2003*, volume LNCS 2863, San Francisco, USA, Oct. 2003. Springer.

[5] Jones, S., Voskoglou, C., Vakulenko, M., Measom, V., Constantinou, A., and Kapetanakis, M. Cross-platform developer tools 2012. Technical report, Vision Mobile, Feb. 2012.

[6] Lang, J. and Janík, J. Reactive distributed system modeling supported by complex event processing. In *Proceedings of 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2013*, Budapest, Hungary, 2013. IEEE Computer Society.

[7] Lang, J., Jantošovič, M., and Polášek, I. Reusability in complex event pattern monitoring. In *Proceedings of 10th Jubilee International Symposium on Aplied Machine Intelligence and Informatics, SAMI 2012*, Herl'any, Slovakia, 2012. IEEE.

[8] Muthig, D., John, I., Anastasopoulos, M., Forster, T., Dorr, J., and Schmid, K. GoPhone—a software product line in the mobile phone domain. Technical report, Fraunhofer IESE, 2004.

[9] Myllymaki, T., Koskimies, K., and Mikkonen, T. On the structure of a software product-line for mobile software. In *Software Infrastructures for Component-Based Applications on Consumer Devices (in conjunction with EDOC 2002*, pages 85–91, Sept. 2002.

[10] Ohrt, J. and Turau, V. Cross-platform development tools for smartphone applications. *Computer*, 45(9):72–79, 2012.

[11] OMG. OMG Unified Modeling Language (OMG UML), superstructure, version 2.4.1. Technical report, 2011.

[12] Porubän, J., Forgáč, M., Sabo, M., and Běhálek, M. Annotation based parser generator. *Computer Science and Information Systems Journal (ComSIS)*, 7(2):291–307, 2010.

[13] Porubän, J., Sabo, M., Kollár, J., and Mernik, M. Abstract syntax driven language development: Defining language semantics through aspects. In *Proceedings of International Workshop on Formalization of Modeling Languages (FML '10), ECOOP 2010*, pages 6–10, Maribor, Slovenia, 2010. ACM.

[14] Rosa, R. E. V. and Lucena, V. F., Jr. Smart composition of reusable software components in mobile application product lines. In *Proceedings of 2nd International Workshop on Product Line Approaches in Software Engineering, PLEASE '11*, pages 45–49, Waikiki, Honolulu, HI, USA, 2011. ACM.

[15] SEI. Software product line overview. `http://www.sei.cmu.edu/productlines/`.

[16] White, J. and Schmidt, D. C. Model-driven product-line architectures for mobile devices. In *Proceedings of 17th Annual Conference of the International Federation of Automatic Control*, Seoul, Korea, 2008.

[17] Xamarin Inc. Building cross platform applications. `http://docs.xamarin.com/printpdf/18634`, 2012.