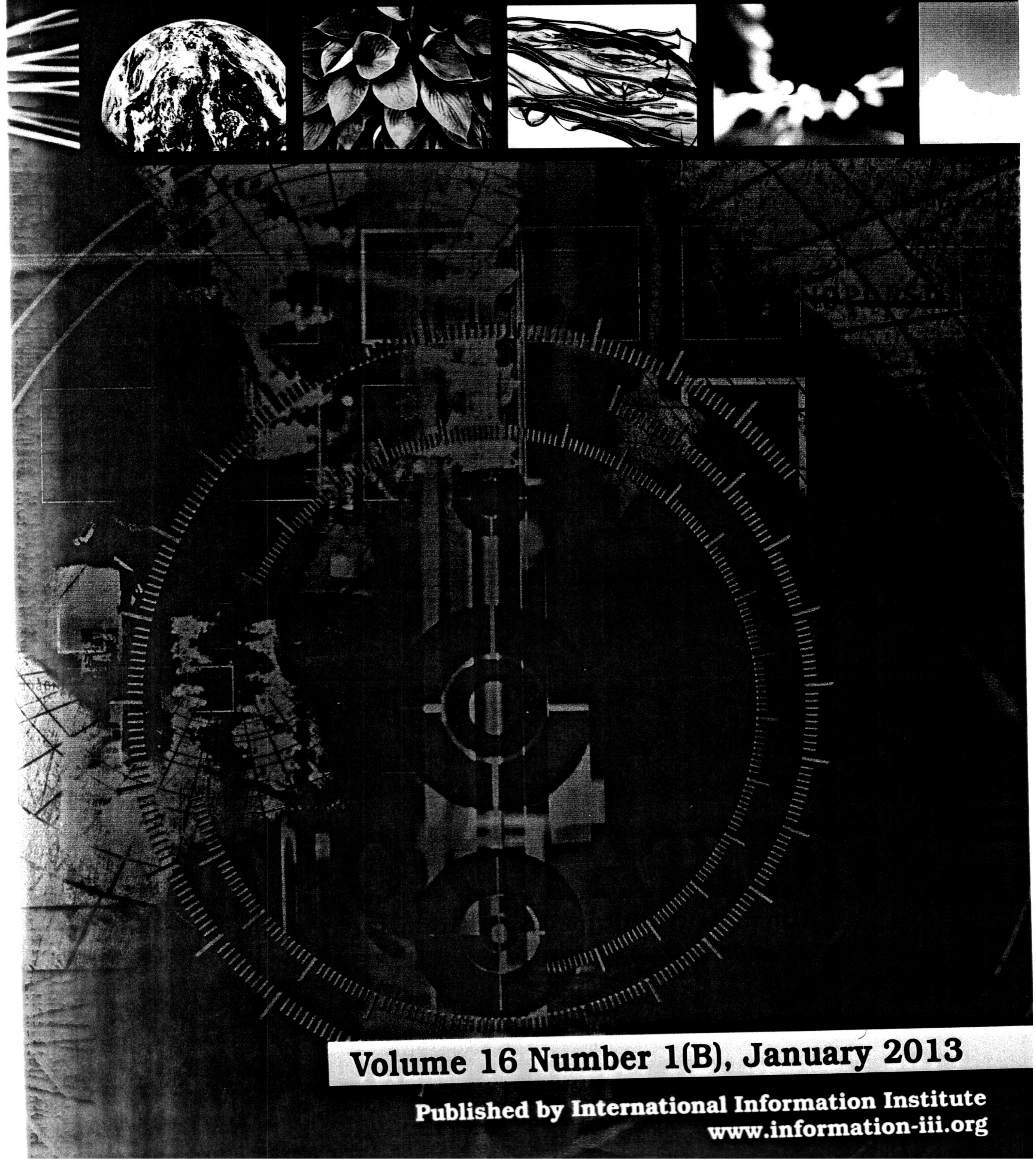
Printed in Japan

ISSN 1343-4500 (print)
ISSN 1344-8994 (electronic)

INFORMATION

An International Interdisciplinary Journal



Volume 16 Number 1(B), January 2013

Published by International Information Institute
www.information-iii.org

INFORMATION : *An International Interdisciplinary Journal*
Volume 16, Number 1(B), 2013

CONTENTS

INFORMATION
Volume 16, Number 1(B), pp.615-625

ISSN 1343-4500
©2013 International Information Institute

Rule Extraction Method for Model Transformations in Heterogeneous Smartphone Applications

Woo Yeol Kim*, Hyun Seung Son**, and Robert Young Chul Kim**

** Dept. of Computer Education, Daegu National University of Education*
Daegu, 705-715, Korea
E-mail: john@hongik.ac.kr

***Dept. Of CIC(Computer and Information Communication), Hongik University*
Sejong Campus, 339-701, Korea
E-mail: son@selab.hongik.ac.kr, bob@hongik.ac.kr

Abstract

Recently, intense competition has emerged among smart phone providers, each offering its own unique operating platform. Since smart phone applications are largely dependent upon the platform for which they are designed, this has made it extremely difficult to develop heterogeneous versions able to run on a variety of platforms. In response to this concern, we propose applying a mode-to-model transformation technique. This involves first analyzing platform differences and similarities, then classifying them into independent and dependent models. Each classification, or category, is used to create an API Matrix which in turn allows for model transformation rules to be generated. Once extracted, a given smart phone application may be adjusted to meet the parameters of a selected platform. To illustrate the proposed model-to-model transformation technique, this paper provides transformation rules extracted for the Android platform, itemized by ATL and adaptable to Android-based smart phone applications.

Key Words: Model Transformation, Android, Smart Phone, Model Transformation Rule, ATL, Meta-model

1. Introduction

Intensified competition among smart phone providers has recently raised concerns over application and platform compatibility. At present, the primary platforms include i-Phone for Apple, Android for Google, and Windows Mobile for Microsoft. This in turn has generated greater interest in and demand for heterogeneous applications able to accommodate a variety of platforms. For such applications to be properly developed, a reusable software system must first be created, capable of integrating and maximizing previously formed resources. Unfortunately, this type of software is difficult to produce since smart phone applications are dependent both on the system being used and the source code selected [1].

In response to this concern, researchers have begun examining the Model Driven Development (MDD) method as a means of achieving heterogeneous versatility. Essentially, this method uses an auto-management process from model to code formation. This involves designing a platform independent meta-model which is turned into a platform dependent model, thereby allowing code to be formed through the model itself. This transformation process permits a selected platform application to be adapted to another platform environment by re-creating code from within the virtual model. In this manner, the model automatically becomes a heterogeneous platform capable of transforming codes for a given smart phone application. Thus, neither the original platform nor the application program need to be changed for a heterogeneous result to occur. Consequently, the re-usability of the model and the auto-formation process of generating codes can potentially increase smart phone development, productivity, and adaptability [2-4].

To illustrate, this paper examines a model transformation technique for the Android platform capable of producing heterogeneous smart phone applications. To begin, model transformation rules must first be developed in order to create a reliable model transformation technique. Unfortunately, such rules are difficult to achieve without first analyzing the differences and similarities between independent and dependent models. To address this concern, an extraction method for model transformation rules is presented, consisting of three stages: first, the platform analysis phase during which model differences and similarities are identified and closely examined; second, the API Matrix writing phase where extracted independent/dependent categories are used to design a graph of the transformation relationship within the API; and third the model transformation writing phase in which the API Matrix is observed as it creates the model transformation rule language.

To execute a sample model transformation, model transformation rules were specifically created by ATLAS Transformation Language (ATL) [5]. For the purpose of this research, the transformation rules applied are for an Android-based platform. In this case, an independent model is created using an Android application, and ATL is used in eclipse to execute the model transformation.

This paper is organized as follows: Chapter 2 presents related work, including the basic concept of MDD; Chapter 3 describes the method for forming model transformation rules; Chapter 4 specifies model transformation rules and describes the model transformation process for Android applications; and Chapter 5 provides concluding comments and suggestions for further research.

2. Related work

As the center of software development has shifted from code to architecture, analysis, and design, the necessity of automating repetitive and cumulative software functions has increased. In response, MDD has emerged as a method for solving problems related to development period and quality [6, 7, 8, 9].

MDD is a cumulative software development process that transforms a design model into a workable system. It also is capable of re-using models that are repeatedly re-defined. In addition, model transformation can be partially or completely automated. Furthermore, when using Unified Modeling Language (UML) as part of the development process, problems may arise if the developer changes the program code during the execution period [10]. Such problems can, however, be solved by using MDD based automation tools [4]. The model transformation methods for supporting MDD include the UML Model Transformation Tool (UMT) [11], Model Transformation Language (MTL) [12], ATLAS Transformation Language (ATL) [5], and Query/View/Transformation (QVT) [13].

Table 1. Evaluation of existing model transformation languages

Evaluation items	UMT	MTL	ATL	QVT
Expansibility	Y	Y	N	Y
Accuracy	Y	N	Y	N
Bi-directional model transformation	Y	N	N	Y
UML meta-model	Y	N	Y	Y
Multi-model	N	N	N	N
Abstract level	N	Y	Y	Y
Re-use and synthesis	N	N	Y	Y
Complex transformation rules	N	N	Y	Y
Heterogeneous model	N	N	Y	N

Table 1 presents an evaluation of the existing model transformation languages according to their respective advantages and disadvantages. This comparison is based on model transformation language items as provided by OMG MOF2.0 QVT. As noted, although UMT provides expansibility, accuracy, bi-directional model transformation, and UML meta-modeling, it does not support multi-models or abstraction. By contrast, MTL only supports expansibility and abstraction. Although ATL provides accurate UML meta-modeling, abstract level output, synthesis, and heterogeneous models, it does not support expansibility, bi-directional model transformation and multi-model facilitation. By comparison, QVT provides expansibility, bi-directional model transformation, UML meta-models, abstraction, re-use, and complex transformation rules; however, it lacks performance quality in heterogeneous models, multi-models, and overall accuracy.

3. Model transformation rule extraction

Model transformation rules must be formed to apply a model transformation technique in the development of heterogeneous smart phone applications. The most important factors of this model transformation process are the extraction of the independent model and the writing of model transformation rules. The steps presented in figure 1 illustrate the manner in which these two factors are achieved. First, independent model and dependent model categories are extracted through platform analysis. In this case, category refers to information regarding class, method, and parameter for executing a model transformation. Next, extracted categories are organized using the independent/dependent model API Matrix. The data formed in the extraction of the independent model categories is then used to create the independent model.

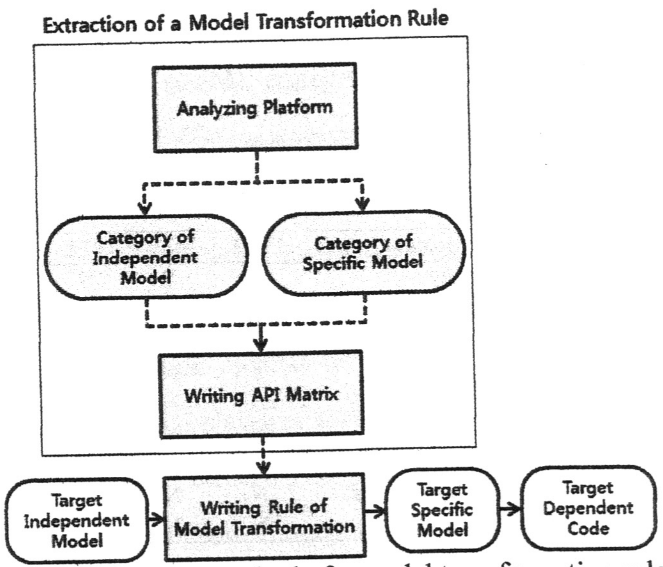

Extraction of a Model Transformation Rule
Analyzing Platform
Category of Independent Model
Category of Specific Model
Writing API Matrix
Target Independent Model
Writing Rule of Model Transformation
Target Specific Model
Target Dependent Code

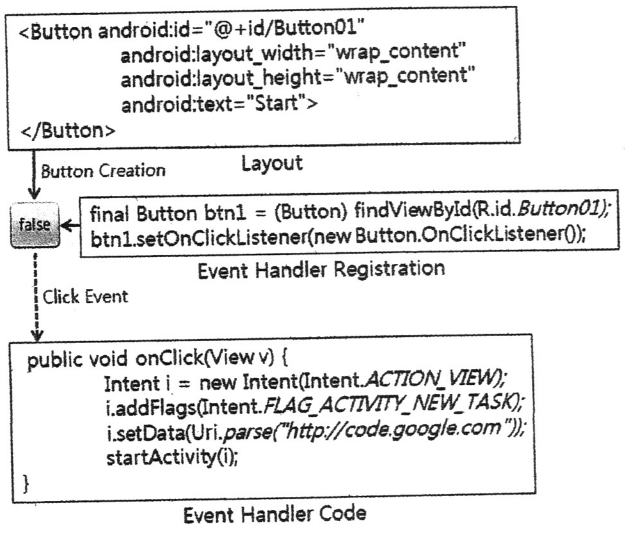
Fig. 1. Extraction method of a model transformation rule

3.1. Platform analysis

The primary smart phone platforms currently used are iPhone of Apple, Android of Google, and Windows Mobile of Microsoft. Each of these 3 each platforms possesses a different operating system, language, UI code, and development environment. To unite these different platforms into one model, careful analysis is required to identify their similarities and differences. For each platform, common factors are classified as Target Independent Model (TIM) items, while differing factors are classified as Target Specific Model (TSM) items. It should be noted that system resources, such as UI component, view, Handler, and Timer, are classified as TIM while the application structure is classified as TSM.

For the Android platform, its architecture is divided into five areas: Applications, Application Framework, Libraries, Android Runtime, and Linux Kernel layers. Of particular importance in this case is the Application Framework since application development can be achieved by using only this architectural tool. In addition, subordinate-level Libraries do not need to be directly controlled as Android's Application Framework is already well-defined.

Closer examination of the Application Framework reveals it to consist of Activities, Service, Broadcast Receivers, and Content Providers. The Activities component is responsible for user interfacing, as well as determining screen layout, arranging UI, and attaching view. Basic applications can be developed by only using this component. Service is not UI presented on screen, but is rather a type of demon executed in the background. In Android, this can be defined as the items that are repeatedly executed by the program. Broadcast Receivers provide communication with other application programs. For example, when certain data is downloaded, a message is sent to related applications to make use of the relevant device. Lastly, Content Providers facilitate the use of functions provided by other applications. For example, the database SQLite may be used during file processing.


```
<Button android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Start">
</Button>
```

```
final Button btn1 = (Button) findViewById(R.id.Button01);
btn1.setOnClickListener(new Button.OnClickListener());
```

```
public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_VIEW);
        i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        i.setData(Uri.parse("http://code.google.com"));
        startActivity(i);
}
```

Fig. 2. Process order of a button event in Android

When a button is pressed to execute a certain action, the event handler is required to process this request. To execute the button handler, the process presented in figure 2 is used in Android.

In order for an Android button application to be executed, a button must first be formed through the Layout designer in the development tool. This allows button specific XML code to be automatically formed. Next, formed Java code is connected to the XML code through the findViewById method. Java is then used to connect and register the button with the event handler before writing and executing the final output.

In analyzing the Android platform, independent model and dependent model categories can be formed, as presented in Table 2. Type indicates the classes and methods used in the platform. Independent Model Category refers to the stereotype present during the model transformation phase. Specific Model Category denotes the dependent class or method to be written which represents the Independent Model Category item to be transformed. Lastly, Comment provides further explanation of each item. In all, these category items are used as the basis for creating the API Matrix.

Table 2. Example of writing independent model and dependent model categories

Type	Independent Model Category	Specific Model Category	Comment
Class	View	View	View class
Method	Draw	onDraw	Method for drawing screen
Class	Controller	Activity	Control method
Class	Button	Button	Button class
Method	Click	onClick	Method generated when clicking a button

3.2. Independent/dependent model API Matrix

Differences between input and transformation models must first be analyzed to achieve model transformation. In examining such differences between independent and dependent models in Android, it can be seen that the primary elements subject to transformation use the basic structure and library provided in the platform itself. Unfortunately, this underlying information is only known to the designer. However, since it is designated by a UML-based design diagram, it is possible to apply a UML-based stereotype to simulate this environment and to clarify the target independent model.

The UML-based stereotype thus added to the class diagram incorporates results of independent and dependent model category analysis. For classification, these results can be divided into class-based and method-based categories. For class-based, there are two stereotypes: <<view>> and <<controller>>. For method-based, there are two stereotypes as well: <<view_onDraw>> and <<button_onClick>>. Furthermore, applicable model transformation rules can be divided into five classifications: basic structure formation, view formation, view_onDraw formation, controller formation, and button_onClick formation. Table 3 below illustrates the independent/dependent model API Matrix.

Table 3. Independent/dependent model API Matrix

No	Type	Stereotype	Target Model		
			class	method	parameter
1	S	-	-	-	-
2	C	<<view>>	Context AttributeSet View <<view>>	- - - <<create>>	- - - c : Context , a: AttributeSet
3	M	<<view_ondDraw>>	Canvas <<view onDraw>>	- onDraw	- c : Canvas
4	C	<<controller>>	Activity	-	-
5	M	<<button_onClick>>	Bundle Button <<controller>> Button.OnClickListener <<button onClick>>	- - onCreate - onClick	- - s : Bundle - v : View

The API Matrix organizes categories formed during the platform analysis process to use for model transformation rule extraction. It consists of type, stereotype, and target model. Type refers to the current transformation type where S=structure, C=class, and M=method. Stereotype refers to the independent item extracted during the platform analysis phase and is vital to the UML model. Although 4 examples are presented herein, additional stereotypes can be formed. Lastly, target model refers to the information that must be transformed when combined with an independent model stereotype. The target model can further be classified into class, method, and parameter.

To illustrate, table 3 notes that example 2 is a class type that forms a context, attribute-set, and view when it encounters the <<view>> stereotype. In this case, the <<view>> class indicates its inclusion of view with parameters given as context and attribute-set. In this regard, the API Matrix essentially becomes the fundamental means by which model transformation rules may be written.

4. Writing model transformation rules

Model transformation rules can be written using the independent/dependent model API Matrix extracted through the platform analysis process. This paper presents 5 model transformation rules.

4.1. Rule 1: Formation of a basic structure

The formation of a basic structure in this case refers to the minimum structure required to execute an Android application. Since data types are fundamentally required for a UML class diagram to be formed, it is therefore necessary to first copy the data type class when forming a basic structure. The activity class which is fundamentally formed in Android executes control during this step. Since this procedure occurs during the formation of the controller itself, it is not a necessary provision of the basic structure formation process.

4.2. Rule 2: Transformation of <<view>>

<<View>> is used to process and present images on screen. Therefore, a view class is required in the Android platform. The view formation process involves creating a view class first, then forming a general relationship between the view and LinkView classes. In Android, the sub-classification class must include the former. Thus, the view class and relevant parameters are added to the LinkView class. From this, both context and attribute-set classes

related to the parameter are created as well. Figure 3 presents the expression of this rule. It is vital to note that only classes with the <<view>> stereotype are transformed for execution.

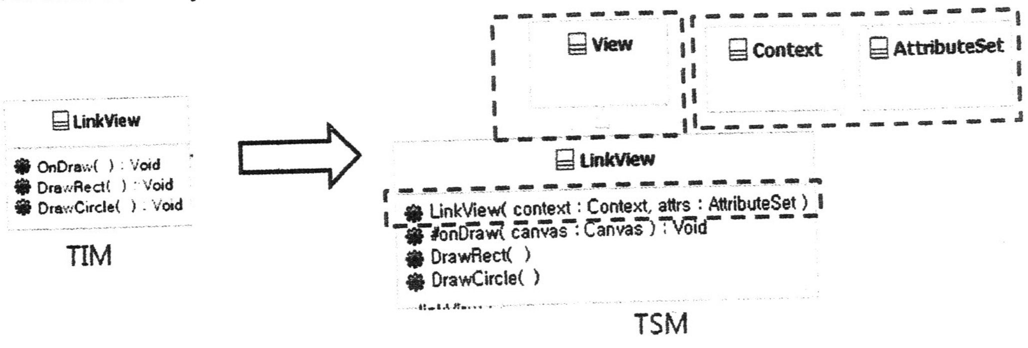


Fig. 3. Transformation of <<view>>

4.3. Rule 3: Transformation of <<view_ondraw>>

To form an image on screen, over-writing of the onDraw method, as previously defined in the view class, must be achieved. Also, a canvas class must be added as a parameter. Thus, to execute these dependent characteristics, first the canvas class should be formed, next the OnDraw method must be changed to the onDraw method, and finally a parameter is added to the OnDraw method. Figure 4 presents the expression of this rule. It is important to note that only the <<view_onDraw>> stereotype will be transformed.

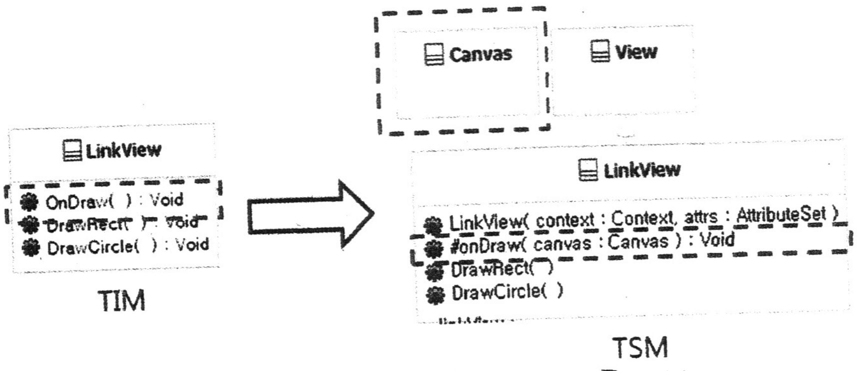


Fig. 4. Transformation of <<view_onDraw>>

4.4. Rule 4: Transformation of <<controller>>

While <<view>> refers to actions presented on screen, <<controller>> is used for processing generated events. In Android, one form of an activity class is required to achieve controller processing. Thus, the controller class must inherit a given activity class. Formation is achieved as an activity class appears, and a general relationship between the activity class and the controller is established. As this occurs, an onCreate method is added and a parameter-related bundle class is formed. Figure 5 presents the expression of this rule. It is important to note that only the <<controller>> stereotype will be transformed.

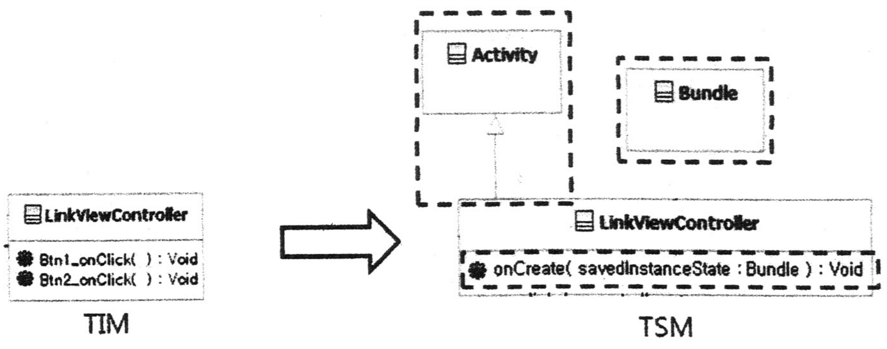


Fig. 5. Transformation of <<controller>>

4.5. Rule 5: Transformation of <<button_onclick>>

In Android, the button function uses a Java-based class to register the event handler while simultaneously writing a command. However, since this class is already present in the proposed method, it does not affect the overall class diagram. For this reason, an interface class can be used instead to create two button classes, thereby allowing certain problems in the design phase to be solved. Once the Button and Button.OnClickListener classes are formed, a correlation between the LinkView Controller class and the Button.OnClickListener class may be established. This in turn allows the Btn1_onClick and Button2_onClick classes to be created, providing 'substantiation' relationship to also be established between the Button.OnClickListener, Btn1_onClick, and Button2_onClick classes. Figure 6 shows the expression of this rule. The execution conditions include the formation and transformation of classes relevant to the <<button_onClick>> stereotype.

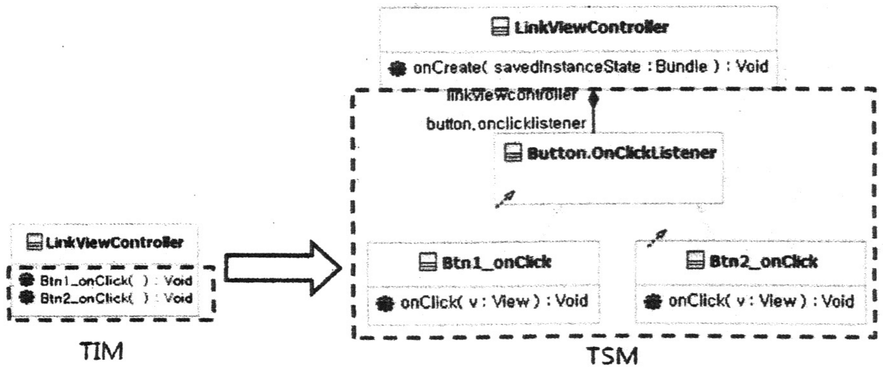


Fig. 6. Transformation of <<button_onClick>>

5. Conclusion

Model Driven Development (MDD) is an effective means in creating heterogeneous smart phone applications. The use of a model transformation technique is vital to this process as transformation rules, generated from an analysis of independent and dependent models, allow for the potential of competing platforms to be bridged.

Since model transformation rules are essential to achieving proper model transformation techniques in MDD, this paper proposed adopting the following steps when writing model transformation rules: first, conducting a platform analysis; second, constructing an API Matrix; and third, writing transformation rules for a heterogeneous application. To illustrate, this paper examined the Android application structure during the platform analysis phase. Each characteristic was extracted and then categorized into either independent or dependent model categories. Next, these extracted categories were used to create a transformed API relationship as part of the API Matrix construction phase. Once completed, the API Matrix was used to specify the intended model transformation rules, effectively acting as a model transformation language.

The model transformation rules created can be divided into two types: class formation-based and method formation-based. Within these types, five detailed rules were generated using the model transformation language ATL. These five rules include <<basic structure>> transformation, <<view>> transformation, <<view_onDraw>> transformation, <<controller>> transformation, and <<button_onClick>> transformation. Also, it should be noted that transformations from the independent model to the dependent model were verified by executing the written ATL details in eclipse. Furthermore, a successful application of the model transformation technique to a smart phone environment was provided.

Although this paper focused on the Android platform, additional applications of the model transformation technique may be achievable for iPhone and Windows Mobile based platforms. Further research is therefore suggested as the demand for heterogeneous smart applications continues to increase.

6. Acknowledgments

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)(NIPA-2012-(H0301-12-3004)) and the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea(NRF) through the Human Resource Training Project for Regional Innovation.

References

[1] Axel Jantsch, Modeling Embedded System and SOCs. *Mogan Kaufmann*, 2004.

[2] Woo Yeol Kim, Hyun S. Son, Young B. Park, Byung H. Park, C. R. Carlson, R. Young Chul Kim, The Automatic MDA (Model Driven Architecture)Transformations for

Heterogeneous Embedded Systems. *Proceedings of The 2008 International Conference on Software Engineering Research and Practice*, Vol. 2 (2008), pp. 409-414.

[3] Woo Yeol Kim, R. Young Chul Kim, A Study on Modeling Heterogeneous Embedded S/W Components based on Model Driven Architecture with Extended xUML. *The KIPS Transactions*, Vol. 14-D, No. 1 (2007), pp. 83-88.

[4] Woo Yeol Kim, Hyun Seung Son, R. Young Chul Kim, C. R. Carlson, MDD based CASE Tool for Modeling Heterogeneous Multi-Jointed Robots. *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 7 (2009), pp. 775-779.

[5] Wikipedia, ATL, http://en.wikipedia.org/wiki/ATLAS_Transformation_Language

[6] B. Selic, The Pragmatics of Model-Driven Development. *IEEE Software special issue on Model-Driven Architecture*, 2003.

[7] K. Czarnecki, S. Helsen, Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, Vol. 45, No. 3 (2006), pp. 621-64.

[8] M. Karanam, A. Rao Akepogu, A Framework for Visualizing Model-Driven Software Evolution – Its Evaluation. *International Journal of Software Engineering and Its Applications*, Vol. 5 No. 2 (2011), pp.135-148.

[9] W. Alouini, O. Guedhami, S. Hammoudi, M.Gammoudi, D. Lopes, Semi-Automatic Generation of Transformation Rules in Model Driven Engineering : The Challenge and First Steps. *International Journal of Software Engineering and Its Applications*, Vol. 5 No. 1 (2011), pp. 73- 88.

[10] S. M. Ghosh, H. R. Sharma, V. Mohabay, Analysis and Modeling of Change Management Process Model. *International Journal of Software Engineering and Its Applications*. Vol. 5 No. 2 (2011), pp. 123-134.

[11] Roy Grønmo and Jon Oldevik, An Empirical Study of the UML Model Transformation Tool(UMT). *In The First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA)*, Geneva, Switzerland, 2005.

[12] D. Vojtisek and J.-M. Je´ze´quel, MTL and Umlaut NG : Engine and Framework for Model Transformation. http://www.ercim.org/publication/Ercim_News/enw58/vojtisek.html

[13] OMG, Documents associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation, Version 1.0, 2008.

*Corresponding author: Robert Young Chul Kim, Prof.

Department of Computer and Information Communication,

Hongik University Sejong Campus,

300 Jochiwon-eup, Yeonki-gun, Choongchungnam-do 339-701, Korea

E-mail: bob@hongik.ac.kr