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.603-613
ISSN 1343-4500

Revised Model Transformation for Model Convergence

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

At present, smartphones reply on a variety of different platforms such as Android, iPhone, and Windows phone. Since most software is developed for a specific platform, it is often difficult to reuse it on other platforms. In response to this concern, one model solution focuses on vertically transforming an independent model to a specific model; unfortunately, for smartphones, this technique has not had applied. In this paper, we suggest a revised model transformation method as a model convergence mechanism adaptable to the smartphone sector. This approach utilizes UML, which allows heterogeneous software to be created through model conversion. In the existing model conversion process, an independent or dependent model in platform A is horizontally converted into other independent or dependent models in platform B. By contrast, the proposed method integrates three stages of transformation: namely, abstraction, transformation, and code generation. In so doing, this model conversion method allows a given model to be partially reused and heterogeneous smartphone application software to be effectively developed.

Key Words: Model Transformation, Model Convergence, UML, Heterogeneous Smartphone Application, Cross Platform

1. Introduction

Smartphone development platforms such as Symbian, OpenC, iPhone, Android, Windows Phone, and Palm operating systems contain various technologies such as widget, Web runtimes, Python, Lazarus, Brew, Java Mobile Edition (ME), .NET Compact Framework (CF), and Flash Lite [1]. Although these provide diverse mobile contents such as audio, video, multimedia messaging, and Flash, they are limited to the platform for which are designed. For this reason, software developers generally prefer to use specific platform-based development method. Unfortunately, since the software developed is based on a specific platform, it cannot necessarily be reused in a heterogeneous manner.

Numerous studies addressing interoperability between platforms in various platform environments have recently been conducted [2]. And while they have examined the potential reuse of models generated during software development process, the results have not yet been

sufficiently applied to the smartphone sector. For this to occur, further research is required in the area of model convergence as a necessary step toward developing fully adaptable smartphone software for multiple platforms.

In this paper, to develop a heterogeneous software model-applicable to smartphone platforms, we adopt Model-Driven Development(MDD) [3] based approach. Using this method, we are able to convert one upper (independent) model to lower (specific) models. It should be noted that in the previous studies focusing on smart platform development with MDD, model convergence was not primary concern [4,5,6,7,8]. In contrast, our suggested method examines this concern first and foremost. The proposed method consists of three stages: first, the abstraction stage to abstract information from a dependent model on an existing platform; next, the transformation stage to convert an independent model into a target model based on the Model-to-Model method; and lastly the code generation stage to generate the code from the target model using the model-to-text method. In so doing, model convergence can occur when the converted model is added to another platform model. Moreover, the code generated is now based on the Model-to-Text Transformation method. To illustrate this process, we use ATLAS Transformation Language (ALT) [9] for Model-to-Model Transformation, as well as Acceleo [10] for Model-to-Text Transformation. It should be noted that such a process requires tools capable of conducting automatic model & text conversions with code templates and conversion rules, details of which will be examined later in this paper.

As a case study, a sample calculator application on the Android platform is transformed into a Windows Phone based platform. In so doing, approximately 90% of the Android based platform software can be reused through model convergence.

This paper is divided as follows: Chapter 2 describes previous model transformation approaches; Chapter 3 describes the model convergence process for heterogeneous platforms; Chapter 4 presents a case study illustrating model convergence; Finally, Chapter 5 provides concluding comments and suggestions for further research.

2. Related work

Most MDD tools only provide Model-to-Code conversions, which in turn generate codes using a Platform Specific Model(PSM) based on a Platform Independent Model(PIM). Unfortunately, these tools have not been sufficiently developed or applied to the smartphone sector [11,12]. This method also relies on a meta-model approach for the conversion of a source model into a target model. The purpose of the Model-to-Model conversion process is

to create an intermediate model when there is an abstract difference between a PIM and a PSM. For example, when materializing a class diagram into EJB with a tool like OptimalJ [13], an intermediate EJB component model is created which contains all the information necessary to make an actual Java code. This allows more modulations and maintainable conversions to be formed. Additionally, the PSM-to-PSM conversion (Model-to-Model Transformation) is able to generate different system model views [14].

The existing model conversion methods can largely be divided into five categories: namely, Direct-Manipulation [15], Relational [16], Graph-Transformation [17], Structure-Driven [18], and Hybrid [19] approaches. The Direct-Manipulation approach provides internal model conversion and controls API. The main benefit of this method is that there is no constraint on the conversion. However, its weakness is that all parts must be materialized for conversion to take place. The Relational approach defines constraints on the relationship between the source model and the target model. However, while there are various connection methods that can be used to map out rules, this approach is difficult to complete without a proper conversion language [20]. The Graph-Transformation approach uses graphs to clarify and specify the conversion process. However, most conversions generated are highly complex. The Structure-Driven approach provides meta-model definitions for each source and target model using model element structures. Unfortunately, this too is highly complex and different to adapt easily. The Hybrid approach is a combination of two or more approaches based on their respective strengths and weaknesses. The Hybrid approach enables various conversions to occur, but renders the conversion process even more complex and time consuming. In this paper, we examine an innovative method for heterogeneous smartphone platforms, model conversion, one that combines a hybrid approach based on ATL and a Structure-Driven approach based on Acceleo.

3. Model conversion method for heterogeneous platforms

The model conversion method for heterogeneous platforms integrates both model-to-model and model-to-text transformations as shown in Figure 1. To begin, an existing model 2A (i.e., the original model) is selected from platform A as shown on the left side of figure 1. For model-to-model transformation, model 2A is then converted into model 2B (i.e., the changed model) in platform B as shown on the right side of figure 1. In other words, the original model is abstracted into an input model as part of the model convergence process. The input model is then changed into an output model through Model Transformation (MT). Next, the

output model is inserted into Model 2B at a specific phase of within platform B. Lastly, the inserted model is converted codes into code template generated by the model-to-text transformation.

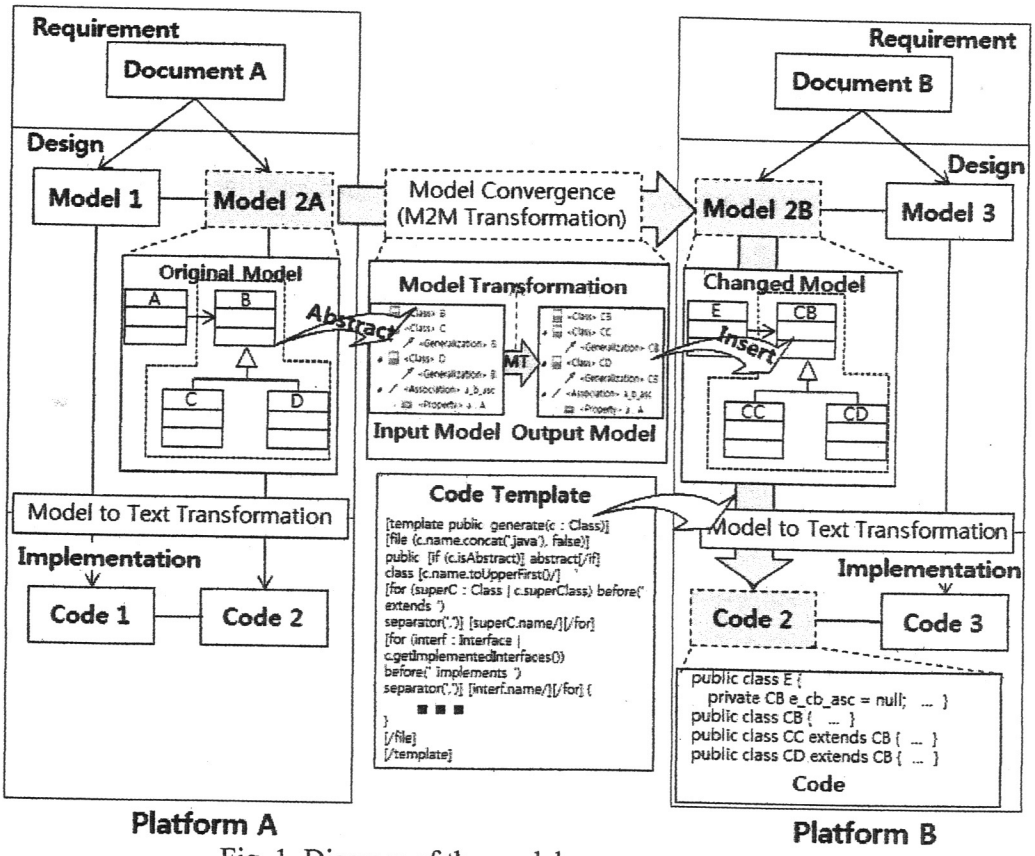


Fig. 1. Diagram of the model convergence process

3.1 Model to model transformation

For model convergence to take place, the proposed method suggests abstracting a model for conversion rather than conversing all of the models at one time. Thus, this technique relies on selective model-to-model conversion for model convergence. Specifically, an original model is selected from an existing model on a given platform. This model is then abstracted into an input model, that is, abstractization. Next, the input model is changed into an output model which allows the platform-dependent model of platform A to be transformed into a platform-dependent model on platform B. In so doing, the abstracted output model can be successfully converted into an adaptable model for platform B. This model-to-model transformation process uses conversion rules generated from ATL. Whereby this output model is correctly inserted into the corresponding position through model convergence.

A rule template to generate correlations with ATL can be created as well. This class-creation template is a UML based Metamodel as shown in Figure 2. It should be noted that new classes may also be generated by revising the "{Name}" in this template.

```
rule TransformationClassTemplateRule {
  from
    o : InputModel!"uml::Class"
  to
    mm : TargetModel!"uml::Class"(
        name <- '{ClassName}', generalization <- generalClass,
        ownedAttribute <- Sequence{ mAttribute }, ownedOperation <- Sequence{ mOperation } ),
    generalClass: TargetModel!"uml::Generalization" (
        general<-thisModule.getRef('{GeneralizationClass}')),
    mAttribute : TargetModel!"uml::Property" (
        name <- '{AttributeName}', visibility <- o.visibility, type <- thisModule.getRef('{TypeClass}') )
    mOperation : TargetModel!"uml::Operation" (
        name <- '{OperationName}', upper <- 1, visibility <- #public, isStatic <- false,
        ownedParameter <- Sequence{parameter} ),
    parameter : TargetModel!"uml::Parameter" (
        name <- '{ParameterName}', type <- thisModule.getRef('{ParameterType}') ), }
```

Fig. 2. Rule Template for Class Creation with ATL

The terms used in the template are provided in Table 1.

Table 1. Terms used for class creation rules

Name	Comment
{ClassName}	Class name
{GeneralizationClass}	Name of class to be inherited
{AttributeName}	Attribute name
{TypeClass}	Attribute type
{OperationName}	Method name
{ParameterName}	Method parameter name
{ParameterType}	Method parameter type

Figure 3 also demotes a rule template to express an association with or the relationship between existing and abstracted classes. That is, a correlation may be generated based on this rule template. A given rule can also be used as a function when expressed in a program language such ATL. For example, if "Transformation AssocationRule ('ClassA','ClassB')" is executed, the correlation between Class A and Class B can be determined.

```
rule TransformationAssociationRule (ref1 : OclAny, ref2 : OclAny) {
    to
    mAssociation : TargetModel!"uml::Association" (
        name <- OclUndefined, ownedEnd <- Set{association1,association2} ),
    association1 : TargetModel!"uml::Property" (
        name <- ref1.name.toLower(), association <- mAssociation,
        upperValue <- ass1_Upper, lowerValue <- ass1_Lower, type <- ref1 ),
    ass1_Upper : TargetModel!"uml::LiteralUnlimitedNatural" (value <- 1 ),
    ass1_Lower: TargetModel!"uml::LiteralInteger"(value<-1),
    association2 : TargetModel!"uml::Property" (
        name <- ref2.name.toLower(), association <- mAssociation, aggregation <- #composite,
        upperValue <- ass2_Upper, lowerValue <- ass2_Lower,type <- ref2 ),
    ass2_Upper : TargetModel!"uml::LiteralUnlimitedNatural" ( value <- 1 ),
    ass2_Lower : TargetModel!"uml::LiteralInteger"(value<-1)
    do { mAssociation; }
}
```

Fig. 3. Rule template for association between classes with ATL

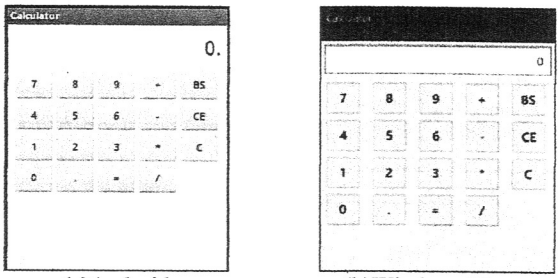
3.2. Model to text transformation

As part of this conversion process, we can generate the code needed for a model in the target platform by using the Model-to-Text Transformation. In the code-generation method, a

code template is used to generate code. In this study, the changed model in platform B is thus converted into code based on the code template presented in figure 1 which was prepared using Acceleo and C#, which mentioned in a previous study [5].

4. Case study for transforming an Android to a Window based platform application

This section provides a sample of the model convergence process for a calculator application (Figure 4) developed for Android platform and converted into a Windows Phone based platform.



(a)Android (b)Windows Phone

Fig. 4. Calculators on heterogeneous platforms

4.1. Android platform model

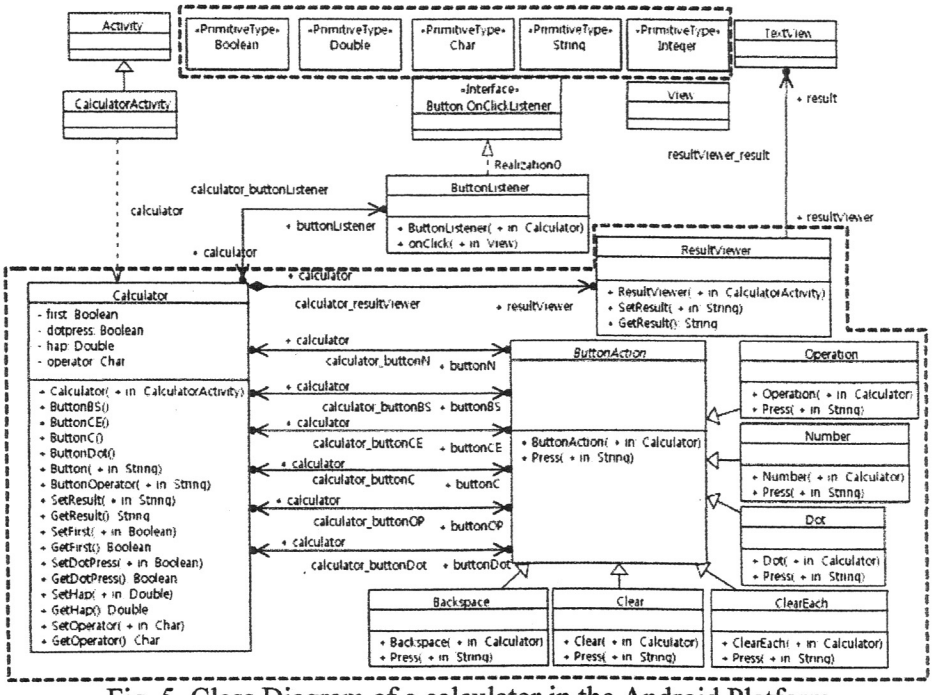


Fig. 5. Class Diagram of a calculator in the Android Platform

The application model for the Android platform, as shown in Figure 5, is presented as a class diagram. The calculator class represents the main class or role of the calculators. The Listener class directs the event processing in the Android platform. The ButtonListener class is used to process many buttons at one time. The ResultViewer class shows the calculated

result. When the calculator button is pressed, the ButtonAction class is activated. Its roles are divided into Backspace, Clear, ClearEach, Dot, Number, and Operation classes. Backspace is a class to delete one letter when the "BS" button is pressed. Clear is a class to delete all data when the "C" button is pressed. ClearEach is a class to delete a formula when the "CE" button is pressed. Dot is a class to feed decimal points when the "." button is pressed. Number is a class carrying out processing when number keys are pressed. Operation is a class applied when "+,-,*,/" keys are pressed.

In the class diagram of an Android calculator, the red dotted box indicates the independent components of the platform. The remaining components are Android platform-dependent. It is important to note that the platform-dependent portions should be separated when designing the software design.

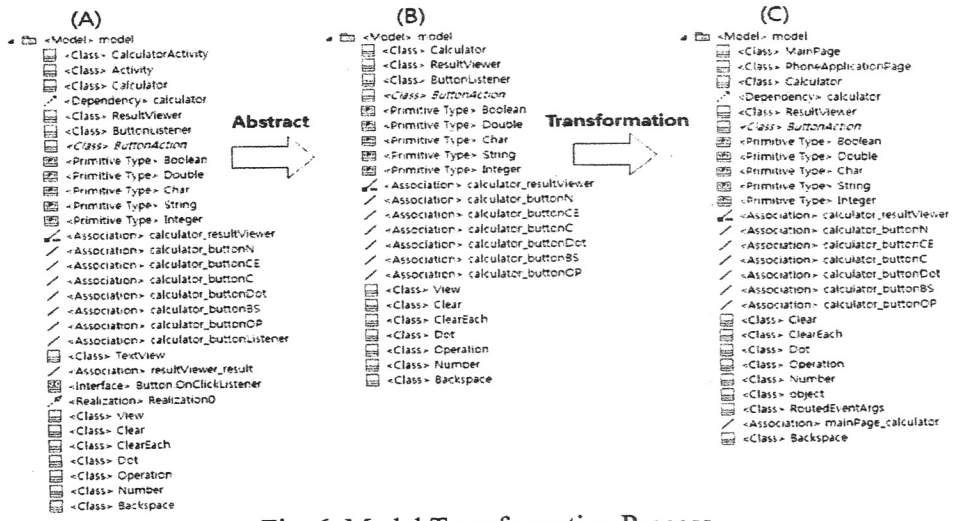


Fig. 6. Model Transformation Process

The results of the model conversion process applied to the Android based platform calculator are shown in Figure 6. The features (A) in the figure 6 are available when the Android class diagram is conducted through the eclipse UML model editor. It can also be confirmed that all components of the Android class are included. (B) is the result of abstractization based on an independent model for reuse. As a result, the platform-dependent models are removed and only the independent model components are included. (C) indicates a model generated through the model conversion process. Once generated, these classes are inserted into the Window platform-dependent models.

Figure 7 shows the results of this model transformation. Based on the model conversion process, most of the independent model structures can be reused. Also, when the MainPage and PhoneApplicationPage classes are generated, they become dependent on the Windows Phone platform as well. As for commands in the Windows Phone platform, MainPage can

now directly receive an event even though there is no ButtonListener class. To increase the reuse of models upon convergence, it is important to separate platform-independent models from platform-dependent ones upon model design.

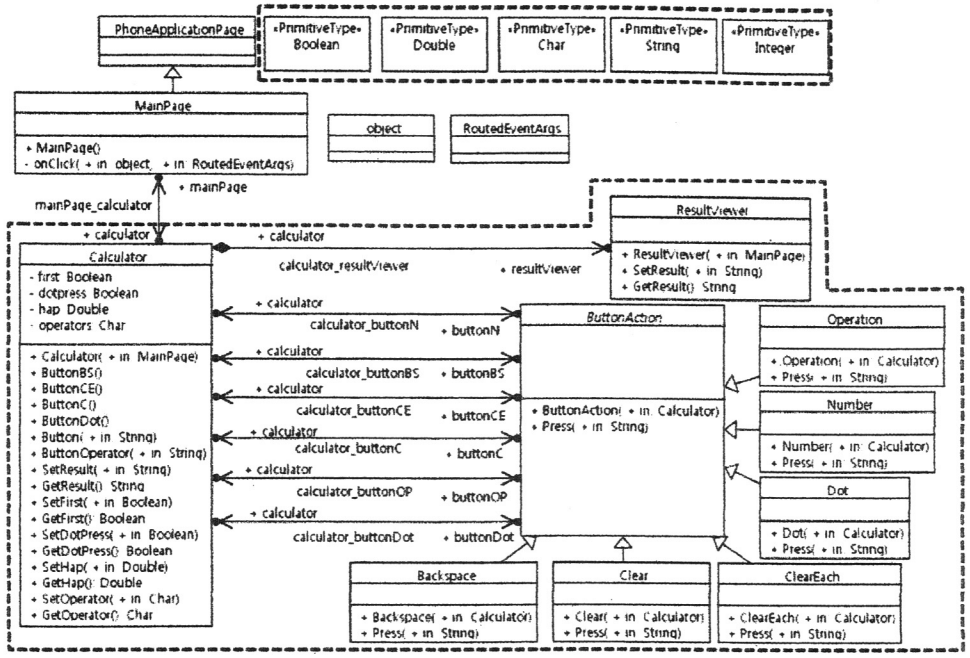


Fig. 7. Class Diagram of a calculator in the Windows Phone Platform

When the model is generated through the model conversion process, codes are then generated through model-to-text transformation. Figure 8 shows a comparison between Android platform and Windows Phone platform code segments. Based on this comparison, we can see that a greater number of codes can be reused when the platform-dependent areas are removed.

```
public class Operation extends ButtonAction {
  public Operation(Calculator c) { super(c); }
  @Override public void Press(String value) {
  if(calculator.GetFirst() && !value.equals("=")){
    calculator.SetOperator(value.charAt(0));return; }
  double susu = Double.parseDouble(calculator.GetResult());
  switch(calculator.GetOperator()) {
  case '+': calculator.SetHap(calculator.GetHap() + susu);
  break;
  case '-': calculator.SetHap(calculator.GetHap() - susu);
  break;
  case '*': calculator.SetHap(calculator.GetHap() * susu);
  break;
  case '/': calculator.SetHap(calculator.GetHap() / susu);
  break; }
   if(!value.equals("=")) {
     calculator.SetOperator(value.charAt(0)); }
  double ii = calculator.GetHap()-(int)calculator.GetHap();
  if(ii == 0) { calculator.SetResult(
        String.valueOf((int)calculator.GetHap()) + ".");
   } else {
  calculator.SetResult(String.valueOf(calculator.GetHap()));
   } if(value.equals("=")) {
       calculator.SetHap(0.0);
       calculator.SetOperator('+'); }
   calculator.SetFirst(true);
   calculator.SetDotPress(false); } }
```

(a) Android's code

```
public class Operation : ButtonAction {
  public Operation(Calculator c) : base(c) {}
  public override void Press(string value) {
   if(calculator.GetFirst() && !value.Equals("=")){
     calculator.SetOperator(value[0]); return; }
   double susu = Double.Parse(calculator.GetResult());
   switch(calculator.GetOperator()) {
   case '+': calculator.SetHap(calculator.GetHap() + susu);
   break;
   case '-': calculator.SetHap(calculator.GetHap() - susu);
   break;
   case '*': calculator.SetHap(calculator.GetHap() * susu);
   break;
   case '/': calculator.SetHap(calculator.GetHap() / susu);
   break; }
     if (!value.Equals("=")) {
       calculator.SetOperator(value[0]); }
     double ii = calculator.GetHap()-(int)calculator.GetHap();
     if(ii == 0) {       calculator.SetResult(
             ((int)calculator.GetHap()).ToString() + ".");
     } else {
       calculator.SetResult(calculator.GetHap().ToString());
     } if (value.Equals("=")) {
         calculator.SetHap(0.0);
         calculator.SetOperator('+'); }
     calculator.SetFirst(true);
     calculator.SetDotPress(false); } }
```

(b) Windows Phone's code

Fig. 8. Code comparison between Android and Window phone platform

5. Conclusion

Presently, efficient model convergence methods for heterogeneous platforms are limited in terms of reliability and adaptability, most notably in the area of smartphone applications. In response to this concern, this paper examined first the Model Driven Development (MDD) method, as a possible convergence approach for smartphone heterogeneous platform environments. This approach automates the process of a software design to software materialization, and enables the conversion of one upper model into other lower models. Although the original MDD method is appropriate for model conversion, it is not capable of horizontal movement between heterogeneous models. Subsequently, MDD is limited and not advantageous for model convergence. Therefore, an alternative model transformation method for model convergence was presented, combining Hybrid Development and Structure-Driven Approaches.

The suggested method incorporates the Model Transformation with MDD based mechanisms and conducts convergence of an existing model with a target model. The first stage is abstractization, which involves separating platform-dependent models from platform-independent models. The second stage is model-to-model transformation, which converts a given model into an abstracted model, then reinserts the latter into its corresponding position. At this stage, class-creation templates are used, as well as correlation generation methods using the conversion language, ALT. The third stage involves Model-to-Text Transformation that converts the model into codes based on a code template generated from the conversion language, Acceleo. Based on this process, partial automation of model convergence occurred and heterogeneous platform interoperability became partially achievable.

To illustrate this process, we presented a case study involving the transformation of a calculator application from the Android platform into the Windows platform. In so doing, nearly 90% of the Android based platform software structures could be re-used. Thus, by using the suggested method, the model may be partially re-used and effectively applied to the development of heterogeneous smartphone application software.

At present, complete automation of the model convergence process is not yet possible and significant manual operation is still required. Details of this problem could be seen in the Class Diagram presented in this study.

Further research should therefore be conducted to address this concern as well as limitation of the class diagramming process.

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] D. Gavalas, D. Economou, Development Platforms for Mobile Applications: Status and Trends. *Software, IEEE*, Vol. 28, Issue 1 (2011), pp. 77 – 86.

[2] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches. *IBM Systems Journal*, Vol. 45, Issue 3 (2006), pp. 621 – 645.

[3] B. Selic, The pragmatics of model-driven development. *Software, IEEE*, Vol. 20, Issue 5 (2003) , pp. 19 – 25.

[4] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, R. Young Chul Kim, Development of Windows Mobile Applications using Model Transformation techniques. *Journal of KIISE : Computing Practices and Letters*, Vol. 16, No. 11 (2010), pp. 1091-1095.

[5] Woo Yeol Kim, Hyun Seung Son, R. Young Chul Kim, Design of Code Template for Automatic Code Generation of Heterogeneous Smartphone Application. *Advanced Communication and Networking*, CCIS 199 (2011), pp. 292–297.

[6] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, R. Young Chul Kim, Adapting Model Transformation Approach for Android Smartphone Application. *Advanced Communication and Networking*, CCIS 199 (2011), pp. 421–429.

[7] Wooyeol Kim, Hyunseung Son, Junbeom Yoo, Young B. Park, R. Youngchul Kim, A Study on Target Model Generation for Smartphone Applications using Model Transformation Technique. *International Conference on Internet (ICONI) 2010*, Vol. 2 (2010), pp. 557-558.

[8] Hyun Seung Son, Woo Yeol Kim, Woo Sung Jang, R. Young Chul Kim, Development Android Application using Model Transformation. *Joint Workshop on Software engineering Technology 2010*, Vol. 8, No. 1 (2010), pp. 64-67.

[9] Wikipedia, ATL, http://en.wikipedia.org/wiki/ATLAS_Transformation _Language

[10] Obeo, Acceleo User Guide, http://www.acceleo.org/

[11] 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.

[12] 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.

[13] OptimalJ, http://en.wikipedia.org/wiki/OptimalJ

[14] K. Czarnecki, S. Helsen, Classification of model transformation approaches. *OOPSLA 03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[15] Jamda, The Java Model Driven Architecture 0.2. http://sourceforge.net/projects/jamda/.

[16] D. H. Akehurst, S. Kent. A., Relational Approach to Defining Transformations in a Metamodel. *UML 2002 - The Unified Modeling Language 5th International Conference*, Dresden, Germany, LNCS 2460 (2002), pp. 243-258.

[17] F. Marschall and P. Braun, Model Transformations for the MDA with BOTL. *Model Driven Architecture: Foundations and Applications*, 2003, pp. 25-36.

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

[19] Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Corporation, et al., MOF Query/Views/Transformations. *Revised Submission. OMG Document*, 2003.

[20] I. Bouzouita, S. Elloumi, Generic Associative Classification Rules: A Comparative Study. *International Journal of Advanced Science and Technology*, Vol. 33 (2011), pp. 69-84.

*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