ISSN: 1738-9984

International Journal of Software Engineering and Its Applications

IJSEIA

Vol.7, No.4, July, 2013

SCIENCE & ENGINEERING
RESEARCH SUPPORT SOCIETY

International Journal of Software Engineering and Its Applications
Vol. 7, No. 4, July, 2013

Concretization of the Structural and Behavioral Models based on model Transformation Paradigm for Heterogeneous Mobile Software

Hyun Seung Son[1], Woo Yeol Kim[2] and Robert Young Chul Kim[1]

[1]*Dept. of CIC(Computer and Information Communication), Hongik University Sejong Campus, 339-701, Korea*
[2]*Dept. of Computer Education, Daegu National University of Education Daegu, 705-715, Korea*

[1] *{son, bob}@selab.hongik.ac.kr ,*[2] *john@dnue.ac.kr*

Abstract

Most model transformation approaches expressed to transform the static model structures of a system, but not involved with the behavioral model. Our previous model transformation [10] also focused on the structural model, especially class diagram, which was restricted to generate a detailed code. To solve this problem, we propose model transformation with both structural and behavioral models, that is, a message sequence diagram with a class diagram for developing heterogeneous software. This approach makes it possible to generate detailed codes through the static & behavioral expression of a system. This is also better to transform more specific design than the previous structure-based model transformation based on only static structure [11-15]. We show the application cases to perform model transformation on the three platforms—Android, iPhone, and Windows Phone.

Keywords: *Model Transformation, Heterogeneous Mobile Software, UML (Unified Modeling Language), MDD (Model Driven Development), Integrated Model*

1. Introduction

The platform-based development provides tools and API (Application Programming Interface) to make effective reuse of resources [1]. This method supports quickly to develop software on one platform. For example, one photo app. of smart phone software cannot use on any other platforms such as iPhone, Android, Windows Phone and something else. This means the smart phone software should be developed per various smart phone environment, and also performed on the basis of platforms [2, 3, 4, 5, 6]. This platform-based method serves as a barrier to the development of other platforms. The reason is that all platforms have their own different, or unique, characteristics instead of providing common methods of use [7].

The e-MDD (Embedded Model Driven Development) [8, 19] is a method for developing heterogeneous software on embedded platforms, which expanded with the original MDD [9]. Because the e-MDD approach also separates between target independent model (TIM) and target specific models (TSMs) at the stage of development, it can reduce to depend on a particular platform. Furthermore, it should absolutely use an automatic model transformation technique to solve the differences between two TIM and TSM models. This paper uses these strengths to conduct a study in order to apply to the different environments of smart phone development [10-15].

The existing model transformation focused on the model based on the class diagram, and provided various characteristics of the static structure on platforms. However, it is not likely to express related information of the dynamic structure of the system, and to create only skeleton codes. In the end, although the simple previous model transformation can help develop heterogeneous software per platform, but not create detailed codes without any behavioral models, that is, the dynamic structures of the system.

This paper applies the behavior model to compensate for the defects of the structure-based model transformation, which uses the message sequence diagram to express the behavior structure of a system. It applies both class diagram and message sequence diagram to the process of model transformation to develop heterogeneous software. One of the preconditions of model transformation should automatically express the possible characteristics of the model. This paper also uses a stereotype for their expression. It defines the rules of naming stereotypes as BNF. Through this stereotype, it is transformed from an independent model to a dependent model. For the purpose of transformation, the paper distinguishes the different characteristics between class diagram and message sequence diagram to apply into the process of model transformation on the platforms of Android, iPhone, and Windows Phone. As a result, it finds out that this model transformation expresses more characteristics of platforms than the existing model transformation.

This paper includes as follows: Chapter 2 mentions e-MDD as related research. Chapter 3 describes the method of model transformation for heterogeneous start phone platforms. Chapter 4 addresses the applied example cases. Finally, Chapter 5 makes the conclusion and works in the future.

2. Related Works

The original MDA (Model Driven Architecture) is architecture that makes it possible to construct a platform-independent model and thereby transforms it into a platform-dependent model and code [16]. On the other hand, MDD (Model Driven Development) [17] generally refers to various methodologies, tools and management methods that help new applications to construct through transforming a platform-independent model into a platform-dependent one, based on MDA. Model transformation is used along with a model transformation language, based on MOF (Meta Object Facility), a standardized meta-model [18, 19].

The existing embedded software development requires depending on each own development environment. However, the problem is difficult to re-use the embedded software byproduct in other environments. The e-MDD was come up with in order to solve this problem. The e-MDD is a method that allows simultaneous development of heterogeneous software during one development life cycle.

The framework of e-MDD consists of vertical model transformation to model (TIM to TSM) and horizontal model transformation from model to code (TSM to TDC) [20]. First, vertical model transformation is to transform from the target independent model to the target specific model based on separation between UML model and UI API Spec. model. That is, the UML model expresses algorithm and data-related parts as M/C (Model/Control) among MVC models. UI API Spec. model, V (view) of MVC, is composed of APIs related to the UI component displayed on the screen. The meta-model of the target model is necessary for these two model transformations. However, as for the existing UML meta-model, it is difficult to manage these different models. Furthermore, since it includes all UML diagrams, it is of complexity. The system of the meta-model is constructed, and the UML meta-model is re-defined to solve these

problems. And the model transformation language is come up with and the model transformation engine is designed so that the meta-model can work in this system.

Second, the horizontal model transformation is a method to create the dependent code using a code template from the target dependent model. It defines the code meta-model and uses the method of transforming the UML meta-model into the code meta-model. In particular, it is important to define the code template for Java, Objective-C, and C#, program languages of Android, iPhone, and Windows Phone, and simultaneously to create heterogeneous codes per each platform.

3. Model Transformation based on the Structural and Behavioral Models

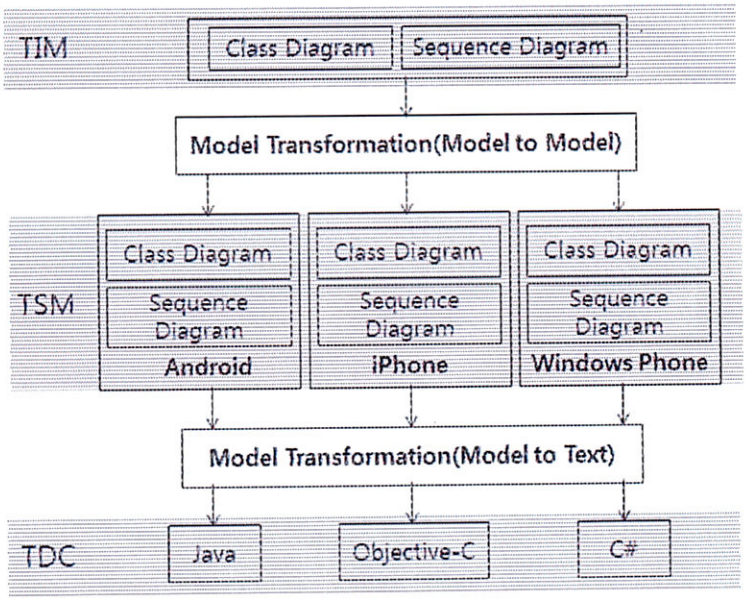


Figure 1. Model Transformation for Heterogeneous Smartphone

Figure 1 shows the structural- and behavioral-based model transformation for heterogeneous smart phones. We use the class diagram to describe a static structure, and have the message sequence diagram to express the behavioral structure of a system. Then, it uses model transformation (Model to Model) to transform two general diagrams on TIM stage into two detailed diagrams on TSM. The code is created with the detailed diagrams through Model Transformation (Model to Text) created. The method proposed by this paper uses the class diagram for expressing the static structure of a system, and needs sequential behavior information in the message sequence diagram. Our model transformation performs with both the class diagram and message sequence diagram simultaneously from TIM to TSM. The stereotype used for each diagram is defined to perform model transformation. The rules of model transformation are also defined for both the class and the message sequence diagram.

3.1. Definition of Stereotype

In order to automatically transform the TIM into the TSM, a particular identified marker is necessary for transformation. The stereotype as mechanism of UML expansion is used for expressing this particular identified one. There is not a special

rule of naming the previous stereotype. However, to classify classes, methods, and attributes, we define the stereotype as BNF as follows:

<SterotypeName> := <ClassName>|<ClassName> "_" <MethodName>|<ClassName> "_" <AtributeName>
<ClassName> := <Identifier>
<MethodName> := <Identifier>
<AtributeName> := <Identifier>

<Identifier> functions as a kind of identifier of programming languages. That is, English letters and the sign "_" should be placed first, followed by English letters, "_", and numbers. It is impossible to express a particular letter except for "_". To take an example of the rule of naming a stereotype, if the class name is 'View', and the method is 'OnDraw', then the stereotype is <<View_OnDraw>>. This paper shows model transformation rules for seven ones in total: <<View>>, <<View_ReDraw>>, <<View_OnDraw>>, <<Timer_Set>>, <<Timer_Refresh>>, <<Timer>>, <<Image_Load>>. It uses <<View>>, <<Timer>>, <<Image_Load>> as the class diagram, and <<View_ReDraw>>, << View_OnDraw>>, <<Timer_Set>>, <<Timer_Refresh>> as the message sequence diagram.

3.2. Model Transformation Rules based on Class Diagram

On transforming model, the most effective expression part in the class diagram is to add or delete some features of class, method and attribute. <View> is a stereotype that is used to draw a picture or to express an image on the screen. Attaching <View> to a name of a class will allow the class to play a role of 'View'.

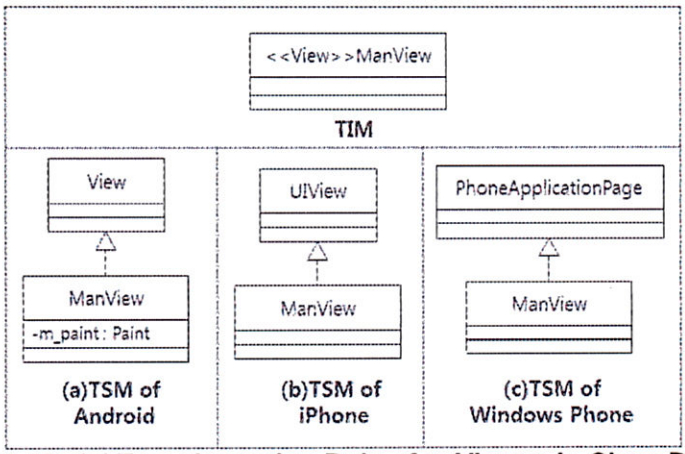


Figure 2. Model Transformation Rule of <<View>> in Class Diagram

Figure 2 shows model transformation of the class diagram when transforming each TSM from the stereotype <View> class of TIM performs model transformation. In Android platforms, it just inherits from class 'View' and has 'Paint' as an attribute. In the iPhone platforms, it inherits from class 'UIView'. In Windows Phone platforms, it does not have special View Class. Classes related to View class are all treated in a UI framework of XAML. There does not exist a callback function to draw a picture. So, we

just assign to inherit from the most similar class 'PhoneApplicationPage'. This way is the reason why this class can treat the events happened from XAML.

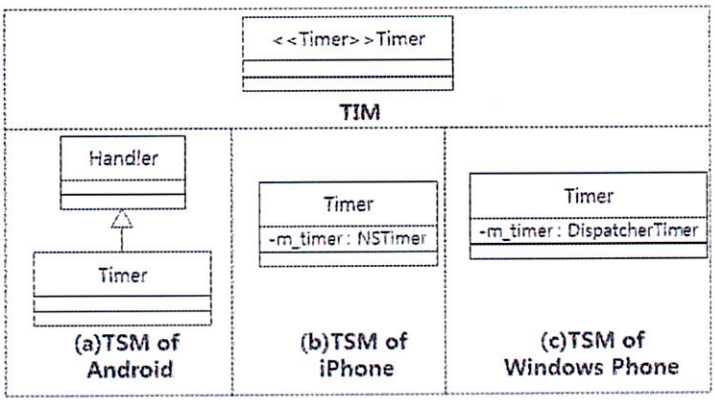


Figure 3. Model Transformation Rules of <<Timer>> in Class Diagram

<<Timer>> is a stereotype used when it is necessary to create time periodically. Figure 3 shows the model transformation of <<Time>> class. In Android platform, it inherits from class 'Handler' because of implementing the timer with 'Handler' class. In contrast, iPhone and Windows Phone just use an attribute without inheriting the timer-related class. iPhone uses class 'NSTimer', but Windows Phone uses class 'DispatcherTimer'. The reason with these different methods to create time is that it depends on whether a callback method is placed inside or outside the class 'Timer'. The inheritance of class 'Handler' like Android means to already exist a callback method in the parent class, which should redefine with overriding. However, if it designates a composition relation through attributes like iPhone and Windows Phone, it will transfer to parameter as the callback method to link with the internal callback method of Timer class. These differences have different rules from each other.

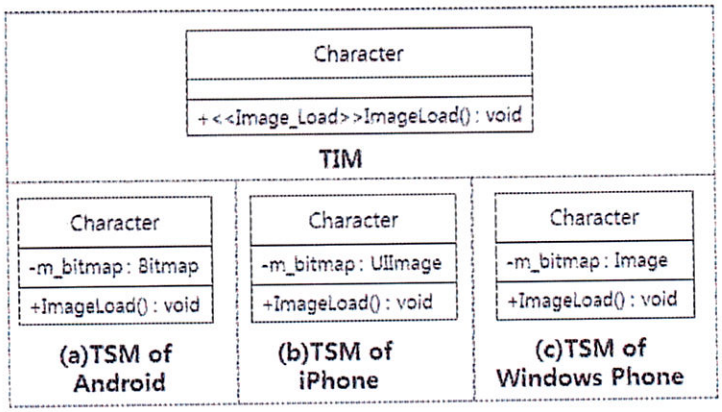


Figure 4. Model Transformation Rules of <<Image_Load> in Class Diagram

<<Image_Load>> is a stereotype used on upload image files in the memory. Figure 4 shows the model transformation of <<Image_Load>>. While the previous examples are all involved in classes, <<Image_Load>> is related with methods. <<Image_Load>> is a method that calls the existing image files to draw pictures on the screen. With the

same mechanism of reading images, Android uses 'Bitmap' class; iPhone 'UIImage' class; Windows phone class 'Image,' respectively. The <<Image_Load>> is not used in the message sequence diagram, but expressed in the class diagram due to adding attributes without time sequences of method calls.

In this manner, the class diagram can effectively depict the program structure. However, it can give no expression to the sequence of calling the methods. The class library of the platform does not only have the simple structure, but also includes this case to ask for a call of other methods before using a particular method.

For this reason, the structure-based model fails to give expression to the sequence of calling API provided by the platform. This paper uses the message sequence diagram to overcome these limitations.

3.3. Model Transformation Rules based on Message Sequence Diagram

The message sequence diagram can give effective expression to sequential behaviors. Therefore, it can express the method calls in temporal sequence. <<View_ReDraw>> is a stereotype to perform re-drawing in a method designated in the class 'View'. <<View_OnDraw>> is also a stereotype originally to draw something from an assigned event. The relationship between these two stereotypes is that after running 'Repaint()' method of <<View_ReDraw>>, it should absolutely execute 'OnDraw()' method of <<View_OnDraw>>. The class diagram is not enough to express this mechanism and can just change the name of the method.

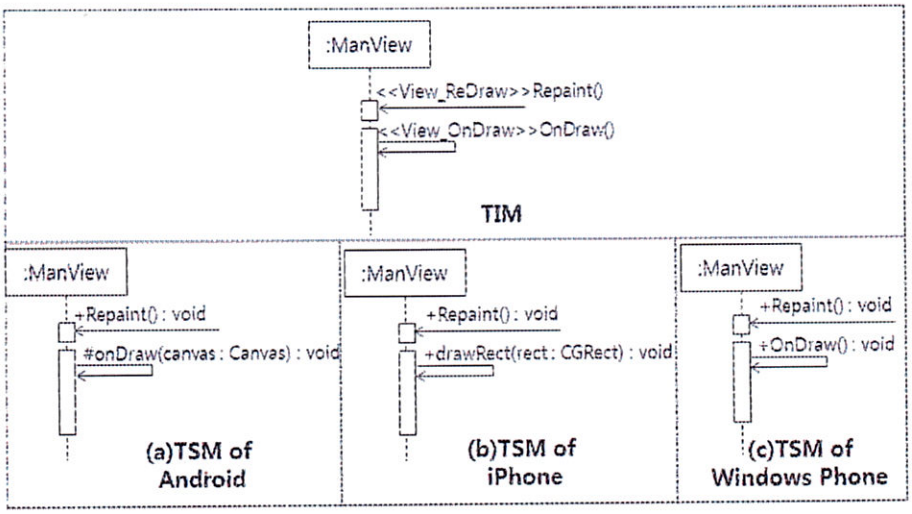


Figure 5. Model Transformation Rules of <<View_ReDraw>> and << View_OnDraw>> based on Message Sequence Diagram

Figure 5 shows the model transformation of <<View_ReDraw>> and <<View_OnDraw>>. On transforming the model based on this rule, it does not change the Repaint method of each platform because of abstracting one method with the behavior of re-drawing. For Android, the method called when draws a picture is changed into 'onDraw(canvas:Canvas)'; for Windows Phone, it is changed into 'drawRect(rect:CGRect)'. Because of overriding class 'View' of both platforms, it should follow the method rule of the parent class. But Windows Phone does not change its method because it does not have a callback method for drawing.

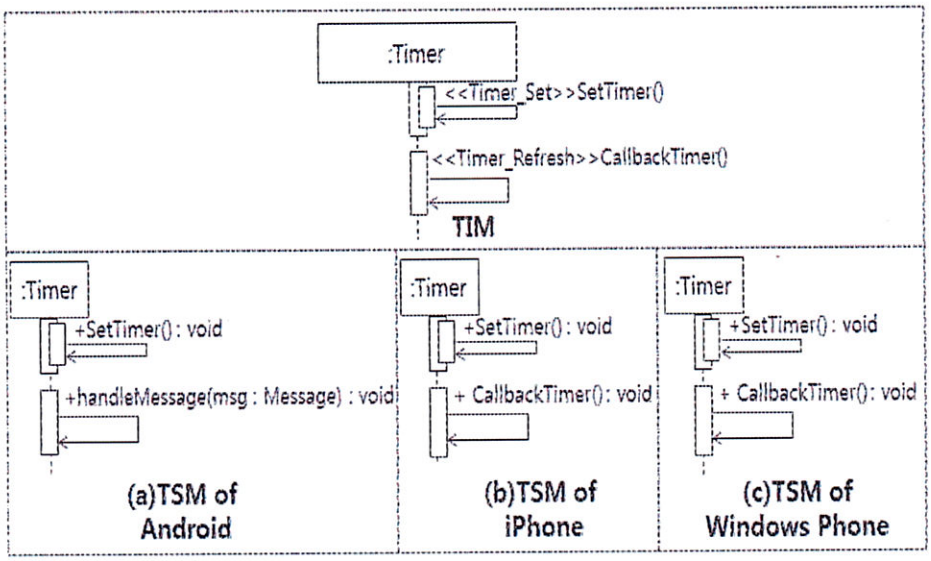


Figure 6. Model Transformation Rules of <<Timer_Set>> and <<Timer_Refresh>> based on Message Sequence Diagram

<<Timer_Set>> is a stereotype for designating time. <<Timer_Refresh>> is a stereotype that is automatically invoked a callback method when it reaches a particular time. Both these methods are associated with class 'Time'. The stereotypes <<Timer_Set>> and <<Timer_Refresh>> always works on together. After executing <<Timer-Set>> method, then a designated method <<Timer-Refresh>> must be called after the designated time. Figure 6 shows the model transformation rule of <<Timer_Set>> and <<Timer_Refresh>>. The Android platform alone has changed the name of its method. This reason is that it uses this method with overriding the class Handler without registering the callback method. iPhone and Windows Phone use the attribute defined within the class diagram that the method of <<Timer_Refresh>> is defined in the method of <<Timer_Set>>.

4. Case Study

This case study shows an application that makes it possible to load and a particular picture on the screen on each platform. In order to develop this application, it is necessary to load a picture and use a timer to move coordinates on the picture. With this simple example, it can transform from target independent model (TIM) to target specific model (TSM) on each platform.

Figure 7 is model transformation based on the class diagram. Three classes are necessary to load a picture and then move it on the screen. View is a class that shows the screen; Timer is a class that calls a method periodically; Character is a class that includes a picture. It shows them as stereotypes to express what to transform in these classes. The rules of <<View>>, <<Timer>>, and <<Image_Load>> defined in Chapter 3 are all performed. Figure 7 (a) is one general model, that is, target independent model. Figure 7(b) shows to transform TIM to TSM of Android; Figure 7(c) shows to transform TIM to TSM of iPhone; Figure 7(d) shows to transform TIM to TSM of Windows Phone.

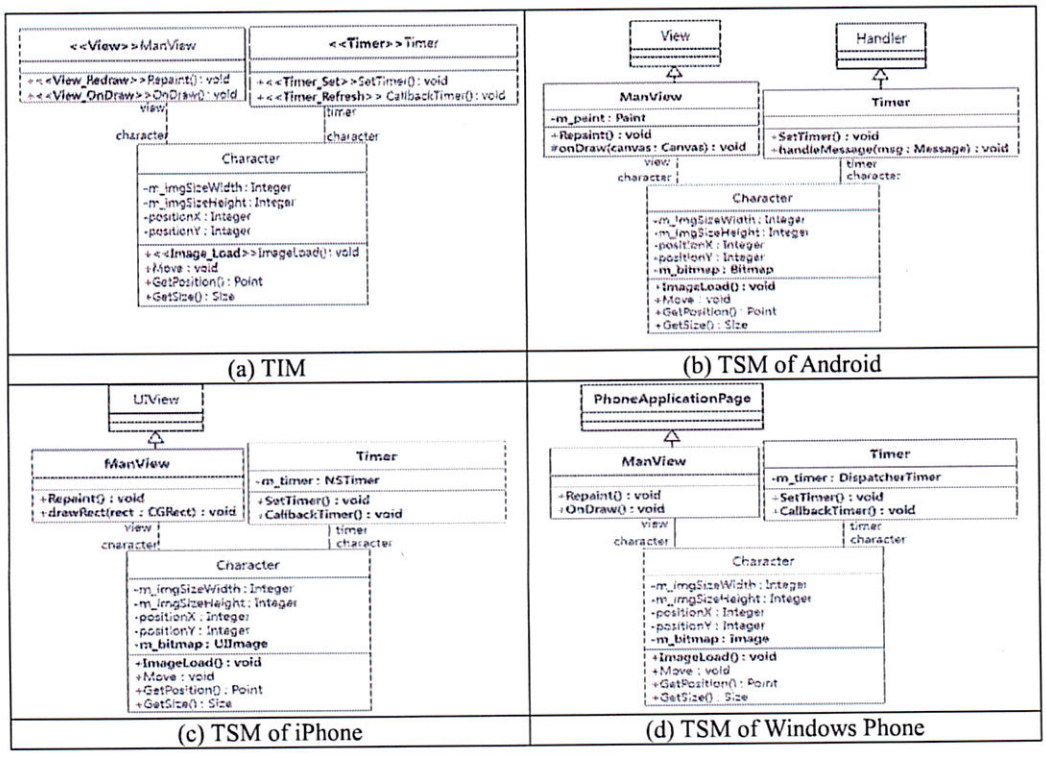


Figure 7. Model Transformation of Class Diagram on each Platform

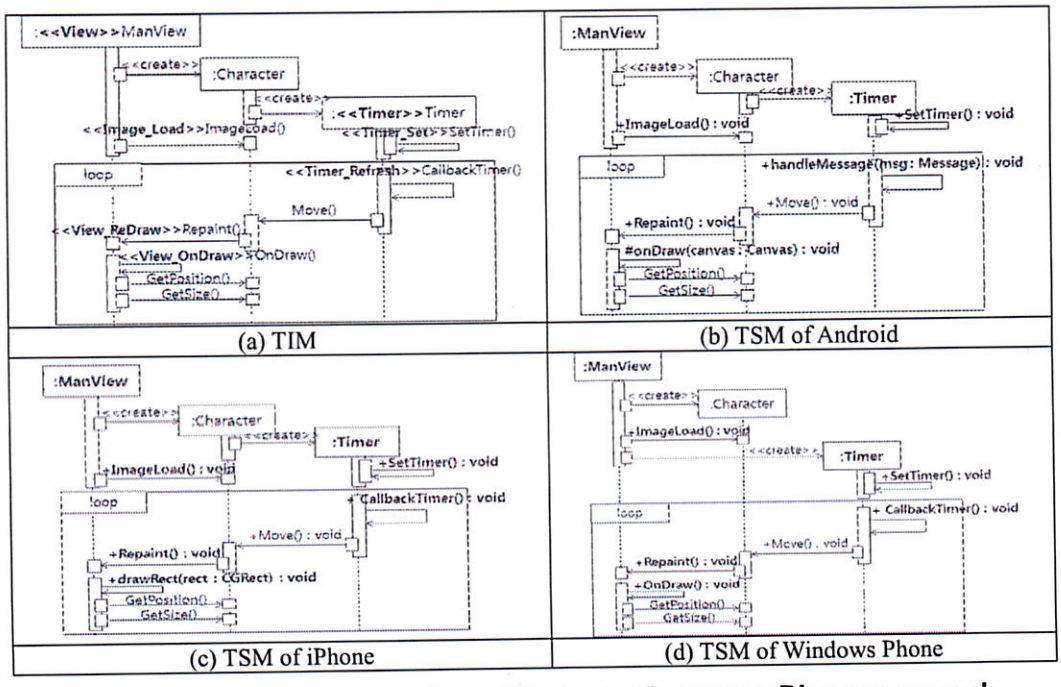


Figure 8. Model Transformation of Message Sequence Diagram on each Platform

Figure 8 is model transformation used message sequence diagram. This diagram shows to indicate when an object be created, and which a method is called in the sequential order. The rules of <<View_ReDraw>>, << View_OnDraw>>, <<Timer_Set>> and <<Timer_Refresh>> defined in Chapter 3 are all performed.

Our model transformation performs with Class Diagram and Message Sequence Diagram on three platforms, which shows to have partially complementary relationship with each diagram. The class diagram can reflect the static characteristics of class, method and attribute; the message sequence diagram can reflect the sequential calling order of methods.

5. Conclusion

The existing model transformations focused on the class diagram does not express the behavior-centered information, which creates only structured skeleton codes. In the end, they can make heterogeneous models with the finally skeleton codes. This paper mentions concretization of the class diagram and message sequence diagram based on model transformation. And it shows the results for the Android, iPhone, and Windows Phone platform. The result indicates that the class diagram can reflect the characteristics of class, method and attribute, while the message sequence diagram can reflect only the order of methods.

For this purpose, it defined the stereotype as BNF and made the rule of model transformation for both class diagram and message sequence diagram. The model transformation rule of class diagram and message sequence diagram defines the 7 most widely used stereotypes: <<View>>, <<View_ReDraw>>, <<View_OnDraw>>, <<Timer_Set>>, <<Timer_Refresh>>, <<Timer>>, and <<Image_Load>>. And it verified the result by applying them to the platforms of Android, iPhone, and Windows Phone.

The result shows that the class diagram can reflect the static characteristics of class, method and attribute, and the message sequence diagram can reflect the calling order of methods. In addition, because it is possible to depend on the message sequence diagram to express the order of methods, it is expected the creation of the objects, and the time of performance. Furthermore, our model transformation can express more specific and dynamic characteristics of the platforms than the existing model transformations.

Our research still is studying on a detailed code creation based on the class diagram and the message sequence diagram by expanding model transformation, and also anticipating high-level code creations such as Java, C++, and C with code template creation through our model transformation. And it will be possible to develop heterogeneous start phones through automation model transformation (from model to model and code).

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0001845) 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 and D. Economou, "Development of Platforms for Mobile Applications: Status and Trends", Software, IEEE, vol. 28, no. 1, **(2011)**, pp. 77-86.
[2] Android, http://developer.android.com/.
[3] iPohne, https://developer.apple.com/.
[4] Windows Phone, http://developer.windowsphone.com/.
[5] J. Yim, "Implementation of Building Recognition Android App.", International Journal of Multimedia and Ubiquitous Engineering, vol. 7, no. 2, **(2012)**, pp. 37-52.
[6] J. H. Yap, Y. -H. Noh and D. -U. Jeong, "The Deployment of Novel Techniques for Mobile ECG Monitoring", International Journal of Smart Home, vol. 6, no. 4, **(2012)**, pp. 1-14.
[7] A. Jantsch, "Modeling Embedded System and SOCs", Morgan Kaufmann, San Francisco, **(2004)**.
[8] W. Y. Kim, H. S. Son, R. Y. C. Kim and C. R. Carlson, "Semi-Automatic Software Development based on MDD for Heterogeneous Multi-Joint Robots", 2009 World Congress on Computer Science and Information Engineering, vol. 7, **(2009)**, pp. 775-779.
[9] B. Selic, "The pragmatics of model-driven development", Software, IEEE, vol. 20, no. 5, **(2003)**, pp. 19-25.
[10] W. Y. Kim, H. S. Son and R. Y. C. Kim, "A Study of UML Model convergence Using Model Transformation Technique for Heterogeneous SmartPhone Application", Software Engineering, Business Continuity, and Education, CCIS 257, **(2011)**, pp. 292-297.
[11] W. Y. Kim, H. S. Son, J. S. Kim and R. Y. C. 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.
[12] W. Y. Kim, H. S. Son and R. Y. C. Kim, "Design of Code Template for Automatic Code Generation of Heterogeneous Smartphone Application", Advanced Communication and Networking, CCIS, vol. 199, **(2011)**, pp. 292-297.
[13] W. Y. Kim, H. S. Son, J. S. Kim and R. Y. C. Kim, "Adapting Model Transformation Approach for Android Smartphone Application", Advanced Communication and Networking", CCIS, vol. 199, **(2011)**, pp. 421-429.
[14] W. Y. Kim, H. Son, J. Yoo, Y. B. Park and R. Y. Kim, "A Study of Target Model Generation for Smartphone Applications using Model Transformation Technique", International Conference on Internet (ICONI) 2010, vol. 2, **(2010)**, pp. 557-558.
[15] H. S. Son, W. Y. Kim, W. S. Jang and R. Y. C. Kim, "Development of Android Application using Model Transformation", Joint Workshop on Software engineering Technology 2010, vol. 8, no. 1, **(2010)**, pp. 64-67.
[16] OMG, MDA Guide Version 1.0.1., omg/2003-06-01, **(2003)**.
[17] K. Czarnecki and S. Helsen, "Feature-Based Survey of Model Transformation Approaches", IBM Systems Journal, vol. 45, no. 3, **(2006)**, pp. 621-64.
[18] OMG, Documents associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation, Version 1.0, **(2008)**.
[19] M. A. Isa, D. N. A. Jawawi and M. Z. M. Zaki, "A Formal Semantic for Scenario-Based Model Using Algebraic Semantics Framework for MOF", International Journal of Software Engineering and Its Applications, vol. 7, no. 1, **(2013)**, pp. 107-122.
[20] W. Y. Kim, "Model Transformation Framework for Heterogeneous Mobile Embedded Platforms", Hongik University thesis, **(2011)**.

Authors

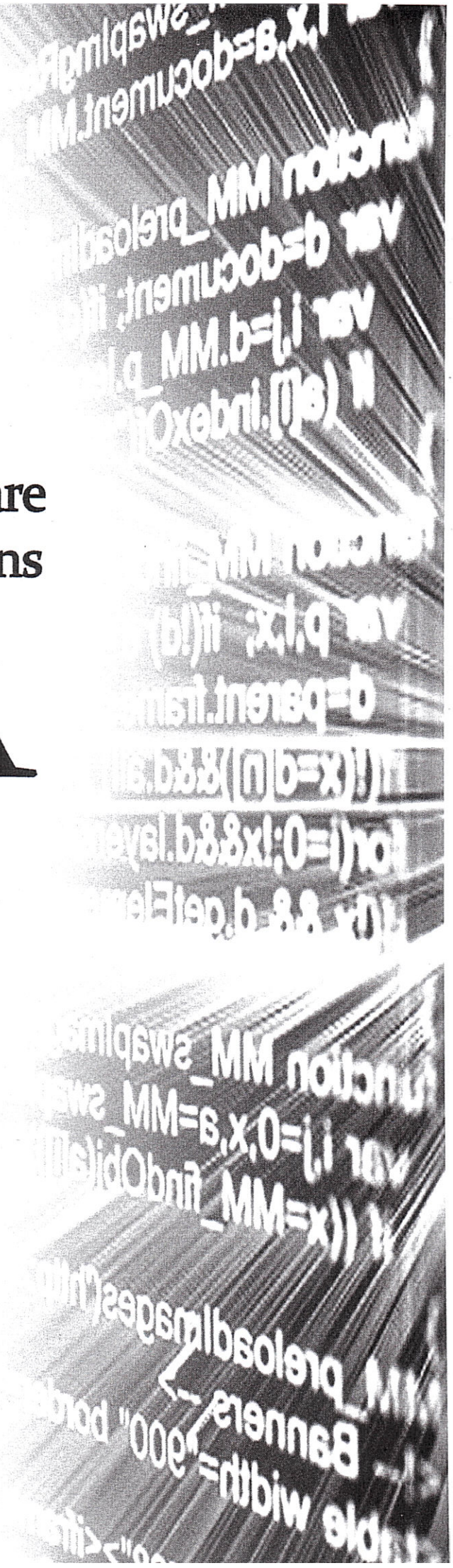
Hyun Seung Son received his B.S. and M.S. degree in Software Engineering from Hongik University, Korea in 2009. He is currently a Ph.D. candidate in Hongik University. His research interests are in the areas of Automation Tool Development in Embedded Software, Real Time Operation System Development, Metamodel design, and Model Transformation, Model Verification & Validation Method.

Woo Yeol Kim received the M.S. and Ph.D. degree in Software Engineering from Hongik University, Korea in 2011. He is currently a professor in Daegu National University of Education. His research interests are in the areas of Interoperability, Embedded Software Development Methodology, Component Testing, Component Valuation, and Refactoring.

Robert Young Chul Kim received the B.S. degree in Computer Science from Hongik University, Korea in 1985, and the Ph.D. degree in Software Engineering from the department of Computer Science, Illinois Institute of Technology (IIT), USA in 2000. He is currently a professor in Hongik University. His research interests are in the areas of Test Maturity Model, Embedded Software Development Methodology, Model Based Testing, Metamodel, Business Process Model and User Behavior Analysis Methodology.

International Journal of Software Engineering and Its Applications

IJSEIA



SCIENCE & ENGINEERING
RESEARCH SUPPORT SOCIETY