

**International Journal of Software
Engineering and Its Applications**

IJSEIA

Vol.7, No.3, May, 2013



Improving the Performances of Software for Rating Patent Technology: A Korean Case Study **343**

Youngkwan Kwon, Tae-Kyu Ryu and Jong Bok Park

Visualizing and Analyzing the Structure of AspectJ Software under the Eclipse Platform **353**

Sassi Bentradi and Djamel Meslati

Advance Convergence Characteristic Based on Recycling Buffer Structure in Adaptive Transversal Filter **377**

*Gwang Jun Kim, Chang Soo Jang, Chan Ho Yoon,
Seung Jin Jang and Jin Woo Lee*

Designing of Framework for Mobile Applications Assets Management **387**

Haeng-Kon Kim

Domain Specific Language for Collaborative Determination of Separation Minima between Aircrafts **399**

Sakon Sinlapakun and Yachai Limpiyakorn

MOF based Code Generation Method for Android Platform **415**

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

A Software Cost Model with Reliability Constraint under Two Operational Scenarios **427**

Satoru Ukimoto and Tadashi Dohi

Selected Block Size-Based Spectral Domain Scrambling **439**

Gwanggil Jeon

MOF based Code Generation Method for Android Platform

Hyun Seung Son¹, Woo Yeol Kim² and Robert Young Chul Kim¹

¹Dept. of CIC(Computer and Information Communication), Hongik University
Sejong Campus, 339-701, Korea

²Dept. of Computer Education, Daegu National University of Education
Daegu, 705-715, Korea

{son, bob}@selab.hongik.ac.kr, john@dnue.ac.kr

Abstract

The existing code generations methods focus on UML Class diagram, which easily represents code structure such as class, method, attribute, but just possibly generate a skeleton code. In this paper we describe to apply UML Message Sequence Diagram (MSD) for representing interactive behavior among objects, and generate more sophisticated Java Code for Android Platform. We also propose code generation method based on Meta Object Facility (MOF) using model transformation technique. And we show metamodel of MSD and model transformation rules written by Acceleo. Using proposed method, we can optimize Java code of Android platform, and increase more code generation rate than the previous approaches.

Keywords: Android, Smartphone, Code Generation, Meta Object Facility (MOF), UML Sequence Diagram, Model-to-Text Transformation

1. Introduction

Platform-based development supported the classes and methods are mostly used as one possible method to quickly develop software. But this platform dependent method gives us difficult to develop on other platform. For example, Android platform [1-3] based software cannot be executed on iPhone Platform [4]. To solve this problem, Model Driven Development approach is appeared by OMG for heterogeneous software [5].

MDD absolutely needs automatic tools and methods to transform platform-independent model into platform-dependent model based on metamodel. To completely follow the MDD approach, it is important to use model transformation technique to transform model into model. Model transformation consists of model-to-model and model-to-text [6]. First, Model-to-model mechanism transforms one dependent model (source model) into one independent model (target model). There exist the tools, ATL [7] and QVT [8], for this mechanism. Second, Model-to-text mechanism is able to create text (program code) from model (the dependent model or target model) through model-to-model step. This method tool as Acceleo [9] generates code with code template.

Some researchers have studied heterogeneous smartphone software development method based on the model transformation technique [10-13]. This previous researches just have focused model-to-model mechanism on UML Class Diagram (CD). Recently, we have studying model transformation using both UML Message Sequence Diagram (MSD) and Class Diagram [14]. Also, we have proposed code template to realize model-to-text [15] for Android (Java), iPhone (Objective-C), and Windows Phone(C#). Actually the previous

research method of code generation has limited and just expresses code structures with Class Diagram. We can generate only skeleton code from this problem.

In this paper, in order to solve this problem of the existing method, we propose one method to generate more dedicate fine codes with both MSD and CD, which can optimize Java code of Android platform through our proposed method, and also implement code generator based on Meta Object Facility (MOF) using Aceleo. This method possibly increases to generate more code rate than the previous approaches.

The paper is organized as follows: Chapter 2 mentions related works. Chapter 3 explains model transformation for heterogeneous smartphone platforms. Chapter 4 presents case studies using model transformation. Chapter 5 gives conclusion and future works.

2. Related Works

Metamodel is model to express model. In other words, this is mechanism for definition to express abstract model of actual worlds. Therefore, metamodel clearly describes necessary constructs and rules to organize specific models in concern domain. Metamodel is shown from three different perspectives: First, in order to build a model that is used building blocks and a set of rules, Second, model of concern domain, Third, instance of other model. Metamodel of Simulink and ECML represents to use Meta Object Facility (MOF) as expressive method of metamodeling [16, 17]. MOF that is establishing OMG standard consists of definition language of metamodel and framework for repository management of metadata. This MOF is used such as metamodel of UML, Common Warehouse Metamodel (CWM), Model Driven Architecture (MDA), and other metamodels. MOF ensures interoperability within the scope, which is defined metamodel on standard.

3. Code Generation from Message Sequence Diagram based on MOF

3.1. Metamodel of UML Message Sequence Diagram

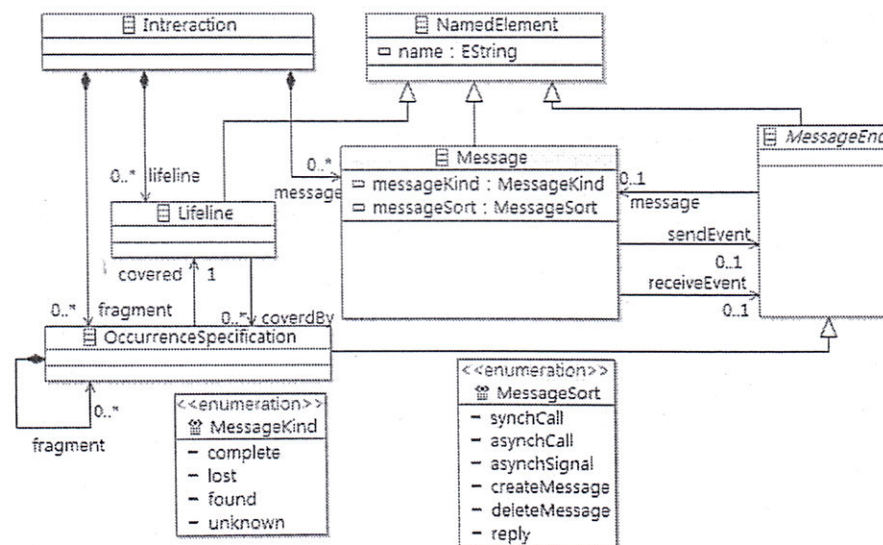


Figure 1. Simplified metamodel of UML Message Sequence Diagram

In order to execute model transformation, metamodel of input model is defined. But, original UML metamodel is a large scale. Therefore we represent metamodel of UML Message Sequence Diagram (MSD) effectively to show our research. We simply design metamodel of UML MSD such as Figure 1. This metamodel consists of Interaction, Lifeline, Message, and OccurrenceSpecification. Interaction is one scenario a number of elements such as Lifeline, Message, and OccurrenceSpecification.

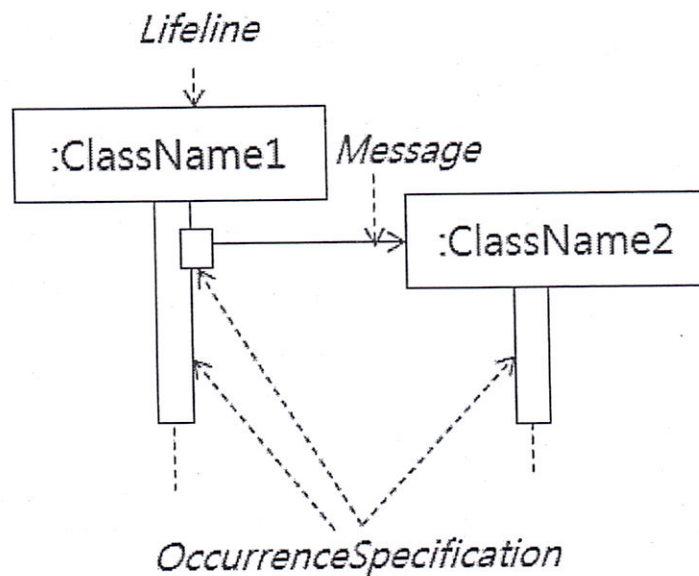


Figure 2. The relationship between model and metamodel

Lifeline, Message, and OccurrenceSpecification show as Figure 2 that metamodel and model have a close relationship. In order to understand, we simply express necessary model for java code generation in Android platform.

3.2. The rules of model transformation for code generation

Code generation method consists of total six rules such as Object Creation (OC), Call Own Method from a Constructor (COMC), Call a Method from Other Object (CMOO), Call the Own Method after Call a Method from Other Object (COM_CMOO), Call Other Object after Call the Own Method (COO_COM), and Call another object after Call a Method from Object (CAO_CMO).

We define six rules of each case as follows:

Case 1: Object Creation (OC)

OC uses stereotype as <<create>> at one step related to object creation. The code generation method consists of three steps as Figure 3. First step add a property definition for object creation in class name "ClassName1", second step add constructor for "ClassName1", and third step add object creation code for "ClassName2" in constructor name "ClassName1". A box on lifeline contacts a box on OccurrenceSpecification in SD, which is related to constructor.

Model (MSD)	
XMI	<pre> <simpleumlsd:Interaction> <lifeline name=":ClassName1" coveredBy="//@fragment.0/@fragment.0"/> <lifeline name=":ClassName2" coveredBy="//@fragment.1"/> <message name="&lt;&lt;create>>" messageSort="createMessage" sendEvent="//@fragment.0/@fragment.0" receiveEvent="//@fragment.1"/> <fragment name="send" covered="//@lifeline.0"> <fragment name="sub-send" message="//@message.0" covered="//@lifeline.0"/> </fragment> <fragment name="recive" message="//@message.0" covered="//@lifeline.1"/> </simpleumlsd:Interaction> </pre>
Rule (Acceleo)	<pre> [template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] / { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.messageSort.toString() = 'createMessage' and aMsg.sendEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let tarClassName : String = aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered.name.replace(':', '.')] private [tarClassName /] [tarClassName.toLowerCase() /]; public [tarClassName.concat('()') /] { [tarClassName.toLowerCase() /] = new [tarClassName.concat('()') /]; } [/let] [/if] [/for] } [/file] [/template] </pre>
Result (java)	<pre> class ClassName1 { private ClassName2 className2; public ClassName1() { className2 = new ClassName2(); } } </pre>

Figure 3. A Rule of Object Creation

Case 2: Call Own Method from a Constructor (COMC)

COMC calls own method from a constructor. The code generation method consists of three steps as Figure 4. First step, sets a constructor definition in class name "ClassName1", second step adds method definition of calling self, and third step adds code invoking own method.

Model (MSD)	
XMI	<pre> <simpleumlsd:Interaction> <lifeline name=":ClassName1" coveredBy="//@fragment.0 //@fragment.0/@fragment.0"/> <message name="method1():void" sendEvent="//@fragment.0/@fragment.0" receiveEvent="//@fragment.0/@fragment.0"/> <fragment name="send" covered="//@lifeline.0"> <fragment name="sub-send" message="//@message.0" covered="//@lifeline.0"/> </fragment> </simpleumlsd:Interaction> </pre>
Rule (Acceleo)	<pre> [template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] / { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.sendEvent = aMsg.receiveEvent)] public [aLifeline.name.replace(':', '.').concat('()')] / { [aMsg.name.strtok(':', 0) /]; } [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype/] [fname /] { } [/let][let] [/if] [/for] } [/file] [/template] </pre>
Result (java)	<pre> class ClassName1 { public ClassName1() { method1(); } public void mehtod1() { } } </pre>

Figure 4. A Rule of Call Own Method from a Constructor

Case 3: Call a Method from Other Object (CMOO)

CMOO is invoked by other object. The code generation method consists of one step like Figure 5. This step adds method definition in class name "ClassName1".

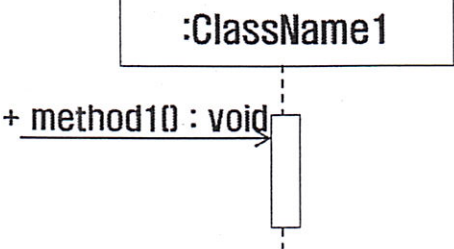
Model (MSD)	
XMI	<pre><simpleumlsd:Interaction> <lifeline name=":ClassName1" coveredBy="//@fragment.0"/> <message name="method1():void" receiveEvent="//@fragment.0"/> <fragment name="recive" message="//@message.0" covered="//@lifeline.0"/> </simpleumlsd:Interaction></pre>
Rule (Acceleo)	<pre>[template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype /] [fname /] { } [/let][/let] [/if] [/for] } [/file] [/template]</pre>
Result (java)	<pre>class ClassName1 { public void method1() { } }</pre>

Figure 5. A Rule of Call a Method from Other object

Case 4: Call the Own Method after Call a Method from Other Object (COM_CMOO)

COM_CMOO is invoked by other object and invoking own method. The code generation consists of three steps like Figure 6. First step, adds invoked method definition by other object in class name "ClassName1", second step inserts method definition of calling self, and third step inserts code invoking own method.

Model (MSD)	
XMI	<pre> <simpleumlsd:Interaction> <lifeline name=":ClassName1" coveredBy="//@fragment.0 //@fragment.0/@fragment.0"/> <message name="method1():void" receiveEvent="//@fragment.0"/> <message name="method2():void" sendEvent="//@fragment.0/@fragment.0" receiveEvent="//@fragment.0/@fragment.0"/> <fragment name="recive" message="//@message.0" covered="//@lifeline.0"> <fragment name="sub-self" message="//@message.1" covered="//@lifeline.0"/> </fragment> </simpleumlsd:Interaction> </pre>
Rule (Acceleo)	<pre> [template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] / { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype] / [fname] / { [if (aMsg.receiveEvent.oclAsType(OccurrenceSpecification).fragment <> null)] [aMsg.receiveEvent.oclAsType(OccurrenceSpecification).fragment.message.name.strtok(':', 1).concat(';') /] [/if] } [/let]/[/let] [else] [if (aMsg.sendEvent = aMsg.receiveEvent)] public [aLifeline.name.replace(':', '.').concat('.') /] { [aMsg.name.strtok(':', 0) /]; } [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype/] [fname] / { } [/let]/[/let] [/if] [/if] [/for] } [/file] [/template] </pre>
Result (java)	<pre> class ClassName1 { public void method1() { method2(); } public void mehtod2() { } } </pre>

Figure 6. A Rule of Call the Own Method after Call a Method from Other object

Case 5: Call Other Object after Call the Own Method (COO_COM)

COO_COM is calling own method and then also calling a method of other object. The code generation method consists of four steps as Figure 7. First step inserts property definition for "ClassName2" in class name "ClassName1", second step inserts method definition of calling self, third step inserts method definition in class name "ClassName2", and forth step inserts code invoking method of other class.

Model (MSD)	<pre> sequenceDiagram participant C1 as :ClassName1 participant C2 as :ClassName2 activate C1 C1->>C1: + method10 : void deactivate C1 activate C1 C1->>C2: + method20 : void activate C2 deactivate C2 deactivate C1 </pre>
XMI	<pre> <simpleumlsd:Interaction> <lifeline name=":ClassName1" coveredBy="//@fragment.0 //@fragment.0/@fragment.0"/> <lifeline name=":ClassName2" coveredBy="//@fragment.1"/> <message name="method1():void" sendEvent="//@fragment.0" receiveEvent="//@fragment.0"/> <message name="method2():void" sendEvent="//@fragment.0/@fragment.0" receiveEvent="//@fragment.1"/> <fragment name="self" message="//@message.0" covered="//@lifeline.0"> <fragment name="sub-send" message="//@message.1" covered="//@lifeline.0"/> </fragment> <fragment name="recive" message="//@message.1" covered="//@lifeline.1"/> </simpleumlsd:Interaction> </pre>
Rule (Acceleo)	<pre> [template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.sendEvent = aMsg.receiveEvent and aMsg.sendEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype/] [fname/] { [/let] [/let] [elseif (aMsg.sendEvent.oclAsType(OccurrenceSpecification).fragment <> null and aMsg.sendEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let tarClassName : String = aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered.name.replace(':', '.')] [tarClassName.toLowerCase() /].[aMsg.name.strtok(':', 1).concat(';')/] } private [tarClassName/] [tarClassName.toLowerCase() /]; [/let] [elseif (aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype/] [fname/] { } [/let][let] [/if] [/for] } [/file] [/template] </pre>
Result (java)	<pre> class ClassName1 { public void method1() { className2.method2(); } private ClassName2 className2; } class ClassName2 { public void method2() {} } </pre>

Figure 7. A Rule of Call Other Object after Call the Own Method

Case 6: Call Another Object after Call a Method from Object (CAO_CMO)

Model (MSD)	<pre> sequenceDiagram participant C1 as :ClassName1 participant C2 as :ClassName2 C1->>C1: + method1() : void C1->>C2: + method2() : void </pre>
XMI	<pre> <simpleumlsd:Interaction> <lifeline name=":ClassName1" coverdBy="//@fragment.0 //@fragment.0/@fragment.0"/> <lifeline name=":ClassName2" coverdBy="//@fragment.1"/> <message name="method1():void" receiveEvent="//@fragment.0"/> <message name="method2():void" sendEvent="//@fragment.0/@fragment.0" receiveEvent="//@fragment.1"/> <fragment name="recv1" message="//@message.0" covered="//@lifeline.0"> <fragment name="sub-send" message="//@message.1" covered="//@lifeline.0"/> </fragment> <fragment name="recv" message="//@message.1" covered="//@lifeline.1"/> </simpleumlsd:Interaction> </pre>
Rule (Acceleo)	<pre> [template public generateElement(aLifeline : Lifeline)] [file (aLifeline.name.replace(':', '.').concat('.java'), false)] public class [aLifeline.name.replace(':', '.')] { [for (aMsg : Message aLifeline.ancestors(Interaction).message)] [if (aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let fname : String = aMsg.name.strtok(':', 0)] [let ftype : String = aMsg.name.strtok(':', 1)] public [ftype/] [fname /] { [/let] [/let] [if (aMsg.sendEvent <> null)] } [/if] [elseif (aMsg.sendEvent.oclAsType(OccurrenceSpecification).fragment <> null and aMsg.sendEvent.oclAsType(OccurrenceSpecification).covered = aLifeline)] [let tarClassName : String = aMsg.receiveEvent.oclAsType(OccurrenceSpecification).covered.name.replace(':', '.')] [tarClassName.toLowerCase() /].[aMsg.name.strtok(':', 1).concat(';') /] } private [tarClassName /] [tarClassName.toLowerCase() /]; [/let] [/if] [/for] } [/file] [/template] </pre>
Result (java)	<pre> class ClassName1 { public void method1() { className2.method2(); } private ClassName2 className2; } class ClassName2 { public void method2() {} } </pre>

Figure 8. Rule of case 6 named call another object after call a method from object

CAO_CMO is invoked method by other object and invoking method from another object. The code generation method consists of four steps like Figure 8. First step adds property definition for "ClassName2" in class name "ClassName1", second step adds method definition by called other object, third step adds method definition in class name "ClassName2", and forth step adds code invoking method of other class.

4. Case Study

Case study is showing one example of moving "stick man" to the right like Figure 9. We developed software for Android using three classes such as ManView, Timer, Character [13]. We use the previous study [13] to show how to generate code with UML Message Sequence.

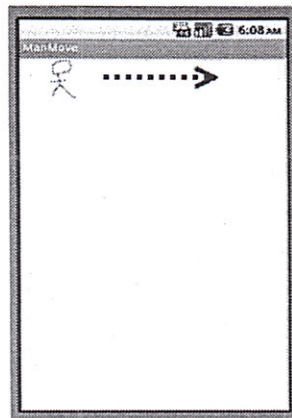


Figure 9. Example of "Stick Man" Android Platform

In Table 1, it shows the generated result with the proposed method to transform code based on MSD. The result increases more 50% code generation rate than using only class diagram.

Table 1. Result of Model-to-Text Transformation

Classification		ManView		Timer		Character	
		Add	Update	Add	Update	Add	Update
Android	Class Diagram	8	2	7	2	17	2
	Sequence Diagram	7	0	6	0	20	0
	Total	15	2	13	2	37	2

5. Conclusion

It is necessary to exploit model transformation technique for developing heterogeneous smartphone applications. The existing code generations methods focus on UML Class diagram, which easily represents code structure such as class, method, attribute, but just possibly generate a skeleton code. This paper applies UML Message Sequence Diagram for representing interactive behavior among objects, and generates more sophisticated Java Code for Android Platform. The proposed method generate code using six rules such as Object Creation (OC), Call Own Method from a Constructor (COMC), Call a Method from Other Object (CMOO), Call the Own Method after Call a Method from Other Object (COM_CMOO), Call Other Object after Call the

Own Method (COO_COM), and Call another object after Call a Method from Object (CAO_CMO). This method can optimize Java code for Android platform, and increase more code generation rate than the previous approaches.

Further research will be extending this work such as iPhone and Windows Phone in the future, which is not dealt in this study.

Acknowledgements

This work was supported by the IT R&D Program of MKE/KEIT [10035708, "The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"] 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] Android, <http://developer.android.com/>.
- [2] J. Yim, "Implementation of Building Recognition Android App.", *International Journal of Multimedia and Ubiquitous Engineering*, vol. 7, no. 2, (2012), pp. 37-52.
- [3] 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.
- [4] iPhone, <https://developer.apple.com/>.
- [5] B. Selic, "The pragmatics of model-driven development", *Software, IEEE*, vol. 20, no. 5, (2003), pp. 19-25.
- [6] K. Czarnecki and S. Helsen, "Feature-Based Survey of Model Transformation Approaches", *IBM Systems Journal*, vol. 45, no. 3, (2006), pp. 621-645.
- [7] Wikipedia, ATL, http://en.wikipedia.org/wiki/ATLAS_Transformation_Language.
- [8] Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Corporation, MOF Query/Views /Transformations. Revised Submission. OMG Document, (2003).
- [9] Obeo, Acceleo User Guide, <http://www.acceleo.org/>.
- [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*, vol. 25, (2011), pp. 292-29.
- [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, 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.
- [13] W. Y. Kim, H. S. Son, J. Yoo, Y. B. Park and R. Y. C. 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.
- [14] 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.
- [15] H. S. Son, W. Y. Kim, J. S. Kim and R. Y. C. Kim, "Concretization of UML Models based on Model Transformation for Windows Phone Application", *Information Science and Technology (IST)*, (2012), pp. 288-291.
- [16] H. S. Son, W. Y. Kim, R. Y. C. Kim and H. -G. Min, "Metamodel Design for Model Transformation from Simulink to ECML in Cyber Physical Systems", *Computer Applications for Graphics, Grid Computing, and Industrial Environment, CCIS*, vol. 351, (2012), pp. 56-60.
- [17] M. A. Isa, Dayang 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.

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