순차도의 추상 시나리오 기반의 UML 상태 머신 다이어그램 시뮬레이션 기법

(An Automatic Simulation Technique for UML State Machine Diagrams based on Abstract Scenarios in Sequence Diagrams)

곽 휘 † 이 우 진 ††
(Hui Guo) (Woo Jin Lee)

요 약 시스템 개발 초기 단계에 시스템의 기능적 요구사항이 제대로 반영되었는지를 검사하기 위해 시뮬레이션 기법을 이용한다. 일반적으로 시뮬레이션은 순차도에 나타난 추상적 시나리오를 바탕으로 상태 머신을 직접 또는 랜덤으로 수행하는 행태로 진행된다. 시뮬레이션은 분석자가 직접 수행해야 하므로 많은 시간과 노력이 소요된다. 이 논문에서는 순차도 기반의 상태 머신의 시뮬레이션의 자동화 기법을 제공한다. 일반적으로 순차도와 상태머신의 추상화 레벨이 달라서 순차도에서 상세 시뮬레이션 트레이스를 추출하기가 쉽지 않다. 이 연구에서는 상태 머신을 LTS 모델로 변환하여 합성적 분석, 트랜지션 축약 등의 분석 방법을 적용하여 순차도와 동일한 추상화 레벨로 변환한 다음, 시나리오 포함여부를 검사한 후 해당 시나리오의 상세 시뮬레이션 트레이스를 생성한다. 이러한 시뮬레이션 트레이스는 순차도에 기술된 시나리오를 기반으로 시뮬레이션을 자동으로 수행할 뿐만 아니라, 특정 시스템 상태까지 자동 시뮬레이션할 수 있으므로 시뮬레이션을 효율적으로 진행할 수 있다.

키워드 : 시뮬레이션, 순차도, 상태머신 다이어그램, LTS, UML

Abstract In an earlier development phase, the simulation technique is one of the key analysis methods for checking the correctness of system's functional requirements. In general, simulation is manually or randomly performed by executing state machine diagrams according to the abstract requirement scenarios. Therefore, simulation is one of the most effort-consuming tasks. In this paper, an automatic simulation technique of state machine diagrams is provided according to the abstract scenarios of the sequence diagrams. It is not easy to generate detailed simulation traces from sequence diagrams due to different abstraction levels between sequence diagrams and state machine diagrams. In order to adjust for different abstraction levels, state machine diagrams and sequence diagrams are transformed into LTS models and compositional analysis and transition reduction are performed. After checking behavior conformance between them, detailed simulation traces for the state machine diagrams are generated. These simulation traces are used not only for performing automatic simulation but also for assisting analyzers to reach a specific system state in order to guide further efficient simulation.

Key words : simulation, sequence diagram, state machine diagram, LTS, UML

· This work was partially supported by BK21 and Defense Acquisition Program Administration and Agency for Defense Development under the contract. (2009-SW-32-DJ-02)
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '시퀀스 다이어그램 기반 상태머신 다이어그램 시뮬레이션'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 경북대학교 전자전기컴퓨터학부
ghelic@gmail.com
†† 종신회원 : 경북대학교 전자전기컴퓨터학부 교수
woojin@knu.ac.kr
(Corresponding author임)

논문접수 : 2008년 8월 25일
심사완료 : 2009년 4월 2일

정보과학회논문지: 소프트웨어 및 응용 제36권 제6호(2009.6)

1. Introduction

During system development, how to validate that a developing system satisfies the user requirements and how to reduce the cost of system modification are very important issues. From the requirement validation viewpoint, simulation performing in the requirement phase is a low-cost alternative to a testing technique which should be performed after the actual implementation of a real system [1]. In the requirement phase by UML [2], the user requirements are described by sequence diagrams and system behaviors are represented by state machine diagrams. During the general simulation of the state machine diagrams, analyzers usually perform a random simulation or manually perform step-by-step execution according to the scenarios in sequence diagrams. However, it is inconvenient and time-consuming to manually perform simulation several times for a large number of user's requirements.

Since the simulation techniques for UML diagrams such as the state machine diagram [3] have been studied for a long time, there are no recent researches on simulation itself. Recently, performance issues [4] or animation issues [5] for UML models are studied. Figure 1(a) shows the general approach of the simulation technique, which is time-consuming work. However, there have been few works for supporting simulation efficiency. In this paper, a way of automatically performing simulation is proposed based on user requirements, which is represented as abstract sequence diagrams. As shown in Figure 2(a), simulation traces for the state machine diagrams are automatically generated by referring the scenarios in a sequence diagram and then guide analyzers to perform simulation by using the generated traces. However, it is not easy to generate simulation traces from sequence diagrams due to different abstraction levels between state machine diagrams and sequence diagrams. Usually, from the users' viewpoint, abstract usage scenarios of a whole system are represented by sequence diagrams in the earlier requirement analysis phase, while the internal system behaviors of system components are described by state machine diagrams in the structural viewpoints.

In this paper, an automatic generation method of simulation traces for UML state machine diagrams is proposed according to the abstract scenarios of sequence diagrams. In order to solve the different abstraction level problem, sequence diagrams and state machine diagrams are transformed into labeled transition systems (LTS) models [6] and different abstraction levels between them are adjusted to the

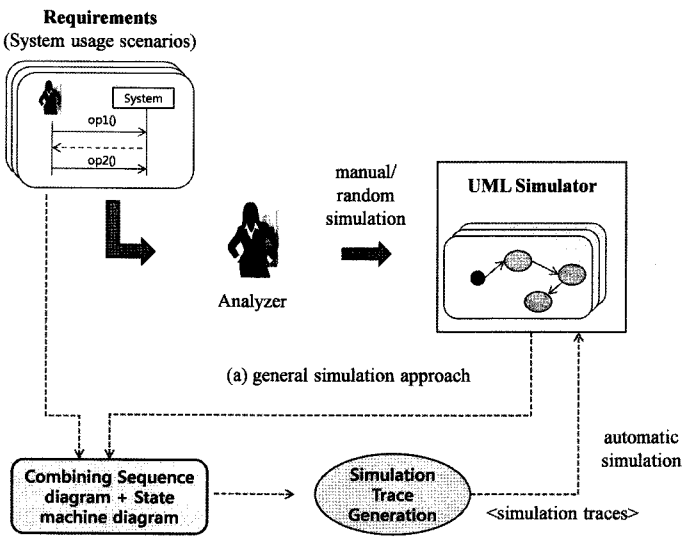

Figure 1 Difference between general simulation and our approach

same one by reducing the detailed information of state machine diagrams. At the same abstraction level, the behavioral conformance between them is checked. If it is satisfied, simulation traces for the state machine diagrams are generated by recovering the reduced transitions information. For automatic simulation, the generated simulation traces are inputted to the main simulator. The proposed simulation approach can be used not only for automatic simulation but also for efficient simulation by guiding to reach some system states. When analyzers want to perform simulation from some system states not from the initial state, this simulation approach can automatically proceed to the system state by performing the corresponding sequence diagram, which consists of a sequence of preceding events.

The remainder of the paper is organized as follows. Section 2 briefly introduces the state machine simulation approach based on sequence diagrams. Section 3 describes the transformation methods from sequence diagrams and state machine diagrams to LTS models. Section 4 describes how to generate the simulation traces. In Section 5, the overall design of the simulation trace generator is provided. Section 6 introduces the study case and verification issues. Finally, in section 7 the study is concluded.

2. Simulation based on Sequence Diagrams

A state machine diagram expresses the internal behaviors of each system component, but a sequence diagram represents the abstract interactions between users and a system or the interaction among system components. Since state machine diagrams and sequence diagrams may have different abstraction levels, it is difficult to combine them directly. To solve this problem, an adjusting method for state machine diagrams and sequence diagrams to the same abstract level is used. It is complicated to analyze one system through state machine diagrams with different hierarchy, while it is relatively easy to analyze LTS models with the same level. Therefore, state machine diagrams to LTS models are transformed. In order to reduce the search space in the analysis, LTS models are combined and reduced in a hierarchical manner. So in the process of abstraction, there are three steps included: transformation, reduction and composition. Sequence diagrams should be referenced in reduction step, and lots of information in the state machine will be removed in this process. For generating detailed simulation traces, however, reduced information should be recovered. Figure 2 shows the overall approach for generating simulation traces.

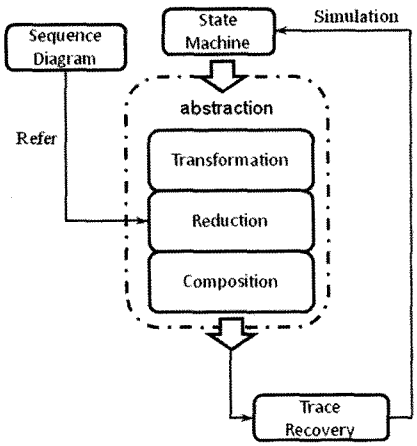

Figure 2 Overall approach of simulation

An LTS has been widely used in the process of analysis. To perform analysis based on LTS, it is necessary to construct the whole behavior model abstracted from the state machine diagrams. For example, consider that a state machine diagram consists of several regions which will be specified by LTS_1, LTS_2, ···, and LTS_n. The whole behavior of all state machine diagrams is described by one composite LTS, which can be constructed by composing the LTS_1, LTS_2, ···, and LTS_n. This approach is generally known as reachability analysis. A major problem with reachability analysis is that the search space involved can grow exponentially with the increase in the number of concurrent processes.

To cope with this problem, reduction techniques have been proposed by minimizing the search space. These reduction techniques can be categorized into two classes; reduction by partial ordering [7] and reduction by compositional minimization [8,9]. In the partial ordering approach, the search space is reduced by excluding the paths formed by the interleaving of the same set of transitions. In the compositional minimization approach, also known as compositional reachability analysis, the search space is reduced

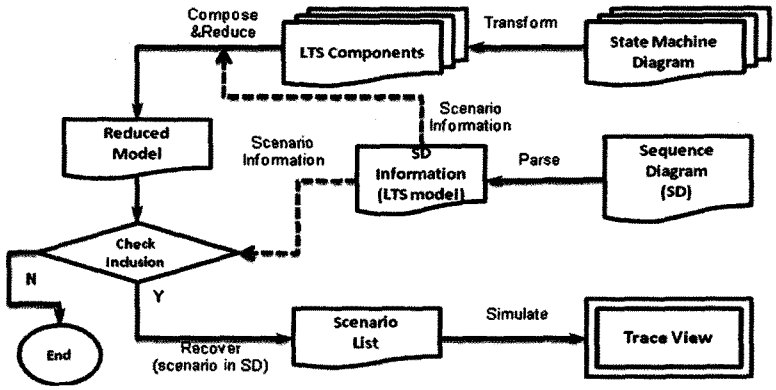

Figure 3 Abstraction and recovery procedure

by compositionally constructing the composite LTS where globally observable actions are abstracted out. Reduction and compositional analysis methods are described in detail later. Figure 3 shows the abstraction and recovery procedure of the proposed approach.

3. Transforming UML models to LTS models

3.1 Transforming state machine diagrams to LTS models

In system modeling analysis, it is complicated to analyze state machine diagrams, since there are different abstraction levels even in a state machine diagram. In contrast, LTS models have a simple structure at the same level, which is easy to analyze. A state machine diagram has many construction elements such as regions, states (simple state and composite state), transitions, while the LTS model consists of states, transitions, and labels. So transformation rules from state machine diagrams to LTS models are defined. LTS is defined as follows.

Definition 1 A labeled Transition System (LTS)
An LTS is denoted by a 4-tuple (Q, Σ, δ, q_0) *where*

- Q *is a set of states,*
- Σ *corresponds to the set of event labels of the LTS, which represents internal events or communicating labels,*
- $\delta \subseteq Q \times \Sigma \times Q$, *denotes a transition relation that maps from a state and an event onto another state,*
- q_0 *is an initial state.*

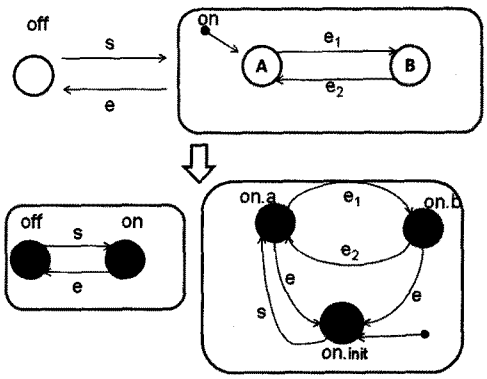

Figure 4 A composite transformation

Following cases should be considered in transformation: 1) a simple transformation describes the simple conversion from state machine to LTS model; 2) a concurrent transformation describes the conversion about transitions fired concurrently; 3) a composite transformation describes the conversion from a state machine, which includes composite states. Figure 4 shows the composite transformation of a state machine diagram.

3.2 Transforming sequence diagrams to LTS models

A sequence diagram shows how processes operate one with another and in what order with exchanging messages. The order of messages in a sequence diagram is mapped to an intermediate state and transitions in the LTS model. Following cases will be considered in transformation: 1) a simple transformation describes the conversion from sequence diagram without combinedFragment; 2) an alterna-

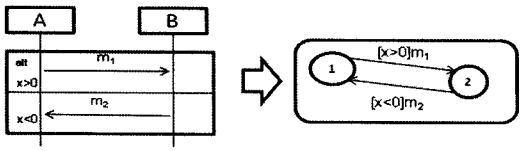

Figure 5 An alternative transformation

tive transformation describes the conversion from sequence diagram with interactionOperator - alt; 3) a parallel transformation describes the conversion from sequence diagram with interactionOperator - par. Figure 5 shows the alternative transformation in a sequence diagram.

4. Generating simulation traces

4.1 State reduction and composition

For effective analysis, it is important to minimize the state space of a system model by localizing and reducing transitions of the LTS model. During the making of a reduced model by the compositional approach, local transitions except for the referenced transitions in the LTS model are abstracted by the λ-elimination rules [10]. Local transitions will be transformed into λ transitions. And they are eliminated by λ-elimination rules such as λ-loop elimination and λ-transition reduction. Figure 6 shows the reduction rules.

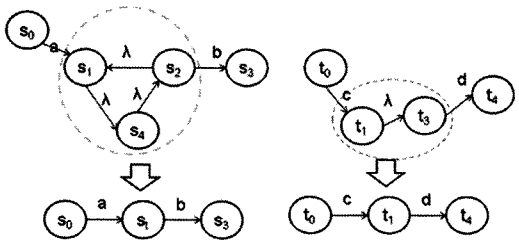

Figure 6 λ-elimination rules

In order to easily check behavior conformance between sequence diagrams and state machine diagrams, it is necessary to build one composite LTS model. Since the composition can be applicable to two LTS models, the composition process is applied to a set of LTS models in a repeated and hierarchical manner. For minimizing the size of intermediately-composed model, the reduction rules are also applied to them repeatedly. In the case of a set of LTS models, compositional hierarchy shows the order of composing base LTS models.

4.2 Inclusion checking

In this section, the reduced system model (LTS_1) and sequence diagram information (LTS_2) will be compared and the expected trace list will result. For checking inclusion between two LTS models, an algorithm based on the weak bi-simulation concept of Milner [6] is provided. Consistency of the models is checked by comparing the reachable trace sets of two LTS models. Reachable traces are defined as follows.

Definition 2 A Reachable Trace

A reachable trace is composed of a sequence of consequent transitions which may be terminated or have an inner loop to a previous transition.

For checking the inclusion of two models, $tr(A) \subseteq tr(B)$, each reachable trace of model A should appear in the model B. Appearance of a reachable trace is easily checked by traversing the trace in the model B. Behavioral equivalence between the A and B models can be checked by performing inclusion checking of both $tr(A) \subseteq tr(B)$ and $tr(B) \subseteq tr(A)$. So if $LTS_2 \subseteq LTS_1$, it means user requirements represented by sequence diagram are satisfied in system design.

4.3 Scenario recovery

Although the scenario of a sequence diagram is satisfied in the state machine diagram, the scenario is too abstract to perform a simulation. The abstract scenario in a sequence diagram should be recovered to detailed scenarios according to LTS models. The information keeping method is adopted before the process of reduction. Information of the original LTS model is held in information containers. And the reduced information is also kept using a disjoint-set, which is attached to information containers.

An actual simulation trace should start from the initial state. So in the process of recovery, the previous transitions, which should be fired from the initial state to the starting state in the abstract scenario, are found. And intermediate transitions between the abstract consequent transitions are also found. But the transitions which are fired after the final transition are not to be considered since they are out of consideration. Figure 7 shows the reco-

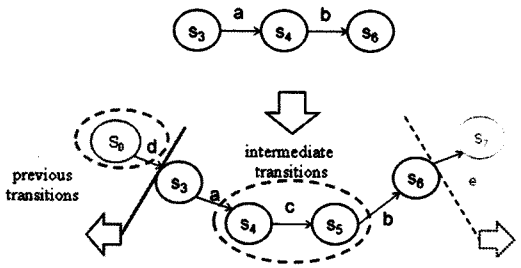

Figure 7 A recovery concept

very process from the abstract scenario a → b. The transition d is the previous transition from the initial state S_0, while the transition c is the intermediate transition between the a and b transitions.

5. Simulation Trace Generator

The simulation trace generator is not an independent simulator but a simulation supporting module. Therefore, it must be connected by main simulator and graphical modeling tool such as a state machine diagram editor and a sequence diagram editor. Figure 8 shows the dependency among the simulation trace generator and existing tools. The simulation trace generator is composed of six major components such as the LTS transformer, the transition reducer, the LTS composer, the sequence diagram parser, the inclusion checker, and the scenario recovery module.

In the simulation trace generator, the information of the state machine diagrams and sequence diagram will be given in xml files, which are saved in the state machine diagram editor and the sequence diagram editor, respectively. The transformation from the state machine diagrams to the LTS models will be done by the LTS transformer. Compositional minimization will be solved in the transition reducer and the LTS composer. Checking inclusion of the reduced system model and the scenario in a sequence diagram is processed in the inclusion checker. The sequence diagram parser finishes the work of representation of sequence diagram information. The scenario recovery will recover the reduced information of each scenario, and the final detailed scenario will be provided to the main simulator as an input trace.

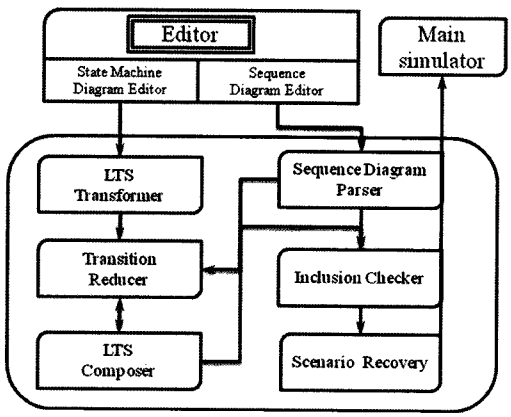

Figure 8 The structure of the simulation trace generator

6. Case Study and Verification

The Multiple Integrated Laser Engagement System (MILES) is a training system that provides a realistic battlefield environment for soldiers involved in training exercises. MILES provides tactical engagement simulation for direct fire force-on-force training using eye-safe laser "bullets". Each individual and each vehicle in the training exercise has a detection system to sense laser hits and to perform casualty assessment. Laser transmitters are attached to each individual and vehicle weapon system and accurately replicate actual ranges and lethality of the specific weapon systems. MILES training has been proven to dramatically increase the combat readiness and fighting effectiveness of military forces [11]. Among huge components of MILES, some parts of a rifle and personal devices are described such as laser launcher, laser sensors, personal displayer, sounder and data transceiver. Figure 9 shows a snapshot of some state machine diagrams of MILES systems.

For checking correctness of the simulation trace generator, the following method is adopted, as shown in Figure 10.

1) Perform the simulation on state machine diagrams using the main simulator several times, and get a set of original simulation traces.
2) For each simulation $trace_x$, abstract it and then get a set of abstract traces.
3) For each abstract $trace_a$ from the abstract trace set, perform the simulation trace generator and

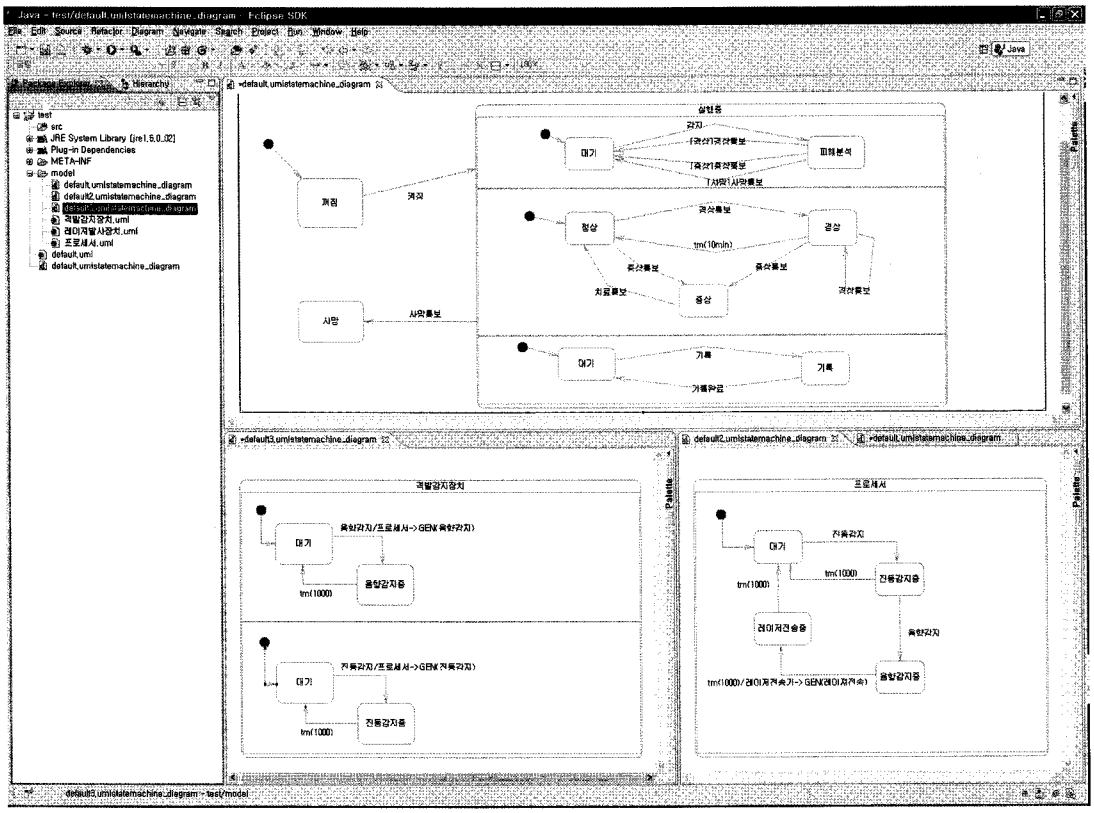

Figure 9 Snapshot of some state machine diagrams of MILES

get the recovered scenario set.

4) Check inclusion of the $trace_x$ and the recovered scenario set. If $trace_x \subseteq$ the recovered scenario set, the simulation trace generator does work.

According to the verification process above, the experiment was based on the MILES system. In the first iteration of 100 times experiments, six errors occurred. After fixing problems, in the second iteration of 100 times experiments, all of them were successful.

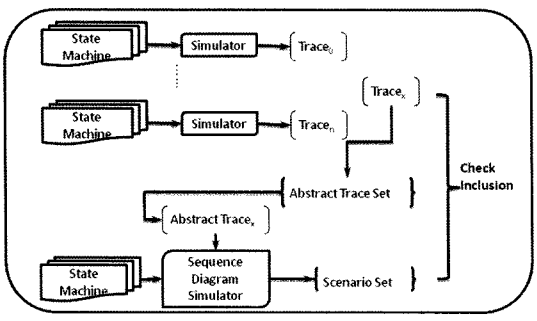

Figure 10 Verification of the simulation trace generator

7. Conclusion

UML state machine simulation makes it possible to check the correctness of system functionality and is widely recognized as a valuable analysis method. In this paper, we provided a new approach of automatic simulation based on sequence diagrams and also developed its supporting tool, the simulation trace generator, which provides the simulation traces into the main simulator. By this tool, in the earlier development phase, the errors of system model can easily be checked and the modification effort for the system model can be minimized. The simulator trace generator can also find the prefix simulation trace, which users can manually simulate from some interesting states. That is, the scenario which appeared in the sequence diagram can be used not only for the whole simulation trace, but also for

finding an interesting state, which is the final state in the sequence diagram.

References

[1] Alexander Egyed and Dave Wile, "Statechart Simulator for Modeling Architectural Dynamics," *Proc. of the 2nd International Working Conference on Software Architecture (WICSA)*, Aug. 2001.

[2] OMG, *Unified Modeling Language : Superstructure*, Version 2.2, http://www.omg.org, 2009.

[3] Enrique V. Kortright, "Modeling and Simulation with UML and Java," *Proc. of 30th Annual Simulation Symposium*, pp. 43-48, Apr. 1997.

[4] M. Marzolla, S. Balsamo, "UML-PSI: the UML performance simulator," *Proc. of First International Conference on the Quantitative Evaluation of Systems*, pp. 340-341, Sep. 2004.

[5] C. Ermel, K. Holscher, S. Kuske, and P. Ziemann, "Animated simulation of integrated UML behavioral models based on graph transformation," *Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 125-133, Sept. 2005.

[6] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[7] G. Holzmann, et al., "Coverage preserving Reduction Strategies for Reachability Analysis," *Proc. of the PSTV*, pp. 349-364, Jun. 1992.

[8] W. J. Yeh, M. Young, "Compositional Reachability Analysis using Process Algebra," *Proc. of ACM SIGSOFT*, pp. 49-59, Oct. 1991.

[9] K. C. Tai, P. V. Koppol, "An Incremental Approach to Reachability Analysis of Distributed programs," *Proc. of the 7th international workshop on software specification and design*, pp. 141-150, Dec. 1993.

[10] P. J. Denning, et al., *Machines, Languages, and Computation*, Prentice-Hall, 1978.

[11] Miles, http://www.fas.org/man/dod-101/sys/land/miles.htm

Hui Guo is working in FIH Technology Korea. He graduated from Qingdao University, China, in 2002. He received MS degrees from Kyungpook National University, Korea, in 2008. His research interests include software engineering, embedded system design, and software testing.

Woo Jin Lee is currently an associate professor in the school of Electrical Engineering and Computer Science at Kyungpook National University, Korea. He received the Ph.D. and M.S. degrees in Computer Science from KAIST in 1999 and 1994, respectively. His main research interests include formal methods, requirements engineering, embedded systems, and component-based software development.