Embedding DEVS Methodology in CBD Process for Development of War Game Simulators

Jung H. Kim*, Tag G., Kim* and JinLip. Jeong**
*Department of EECS, KAIST, Daejeon, Korea
** Rebublic of Korea Marine Corps, Pohang, Korea

Keywords: DEVS, CBD, War Game, Object-oriented, UML

Abstract

CBD (Component Based Development) methodology is a cost effective software development paradigm in which encapsulated software modules are assembled via interface to perform required functions. The CBD process directly applicable to general software development projects in conjunction with UML (Unified Modeling Language) based OOD (object-oriented development) methodology. However, special projects, such as war game simulators development, may not directly employ the OOD methodology within the process due to limitation in modeling power for war game components as discrete event systems. This paper presents employment of ***DEVS*** (Discrete Event Systems Specification) methodology for development of war game simulators using the CBD process. Embedment of the DEVS methodology is not replacement, but complement of the UML-based OOD methodology. Case study for development of a war game simulator using the DEVS embedded CBD process will be discussed.

1. Introduction

CBD methodology is a software development paradigm to assemble applications from reusable, executable software pieces, called ***Components***. Once developed, a component is repeatedly used in many projects via well-defined CBD interfaces, which greatly reduces the software development cost and increases the reliability. Usually, CBD for general software development projects is employed in conjunction with OO methodology using UML [1], as UML provides a descriptive power that is especially useful for object modeling.

Although it has worked in many commercial projects, UML-based CBD has one inherent problem. Since UML does not provide formal semantics of specification of an object as a discrete event model, UML-based CBD also lacks a modeling power of discrete event systems, which makes that the process cannot be directly applied to a kind of software like ***war game simulators***. War game simulators require specification of component models with discrete events, and therefore cannot employ conventional UML-based CBD processes like ***ADDMe*** [2], the Korean military standard CBD process. Kim [3] addressed this limitation of UML on discrete event systems, and proposed the Extended Sequence Diagrams that is enhanced from sequence diagrams in UML with DEVS. DEVS is a formalism that supports specification of discrete event system with sound semantics [4]. The DEVS formalism is known to be compatible with object-oriented world view for modeling systems in hierarchical, modular fashion, and has been widely applied to OO modeling and simulation of discrete event systems including war games.

This paper adopts the DEVS-extended UML, and applies it into a conventional UML-based CBD process of ADDMe to support war game simulators. Embedding DEVS methodology within ADDMe allows developers to employ sound semantics in DEVS for various phases of the process, ranging from analysis, design to testing. Embedment of DEVS methodology is not replacement of UML-based OOD methodology, for the methodology is still useful for development of some components in war game simulators such as database and user interface. Thus, the DEVS methodology is a complement of the UML-based OOD methodology in the development.

This paper is organized as follows. Section 2 gives brief overviews of CBD, UML, and ADDMe. Section 3 describes DEVS, and roles of DEVS in war game simulators. After discussing how to embed the DEVS methodology into ADDMe in Section 4, we detail the embedment with the case study of ChunJaBong Simulator, a war game simulator for Korean Marine Corps.

2. CBD Methodology and ADDMe process

In this section, we provide a brief overview of CBD and UML, and introduce ADDMe, the Korean military CBD methodology. Finally, this section describes the limitation of UML-based CBD on discrete event systems.

2.1. CBD Methodology

CBD (Component Based Development) methodology is a cost effective software development paradigm in which encapsulated software modules are assembled via interface to perform required functions. The methodology can be applicable to all phases of software development process which includes plan, analysis, design, implementation, testing and surrender. Different CBD methodologies are developed in different projects, some of which are Catalysis [5], RUP [6], Select Perspective [7], and ADDMe [2].

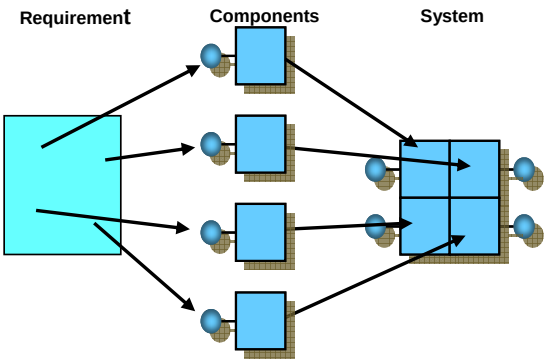


Figure 1. CBD Concept

Figure 1 exemplifies the concept of CBD. As shown, the application requirements are divided into several independent components, each of which may be newly implemented, or adopted from other projects [8]. Such modularity, and reusability of CBD over different projects greatly increases both time-to-market and reliability.

Generally, CBD process for general software development projects is employed in conjunction with OO methodology using UML. It is because the concept of components is very compatible with that of objects in OO and UML provides a strong descriptive power on object modeling. In UML-based CBD processes, analysis and design of components rely on UML models, which have successfully proven themselves in many projects.

For instance, there is a CBD process for Korean military systems development, called ADDMe. ADDMe has been developed at ADD (Agency of Defense Development) as a Koran military standard for systems development. ADDMe is also based on UML just like other CBD processes, and adopted in many Korean military projects.

2.1.1. UML : OO Modeling of General Software

The Unified Modeling Language (UML) [1] is one of the most exciting tools in the world of object-oriented system development today. UML enables system builders to create blueprints that capture their visions in a standardized, easy-to-understand way, and to communicate them to others. Basically, UML models a system in multiple views, each of which is represented by a corresponding graphical diagram.

As a visual modeling language, UML employs 8 graphical diagrams: use case diagram, class diagram, object diagram, state diagram, activity diagram, sequence diagram, collaboration diagram and component diagram.

2.1.2. ADDMe

ADDMe [2] is a CBD process that is developed by ADD for Korean military systems development. Currently, it is widely used in many military projects as Korean military standard. It is based on UML like other CBD processes, and composed of 4 steps of ***Analysis***, ***Design***, ***Implementation and Testing***, and ***Surrender***. The details of each step are as follows.

2.1.2.1. Analysis Step

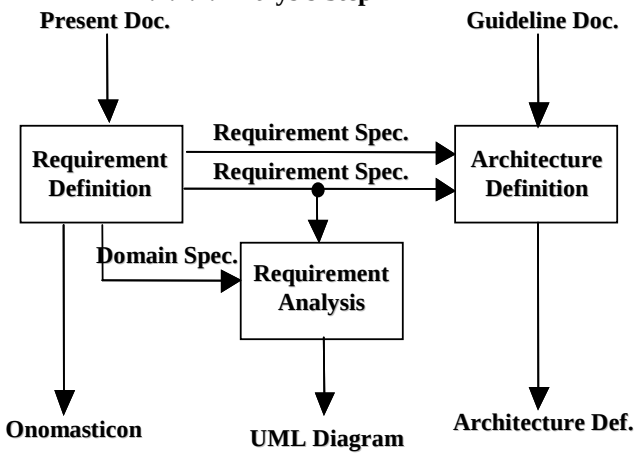


Figure 2. Analysis Step

The Analysis Step mainly identifies user's requirements. It clarifies the domain scope with use case diagrams and class diagrams from several guidelines, business model, and other user requirements. It also generates architectures for software and system.

2.1.2.2. Design Step

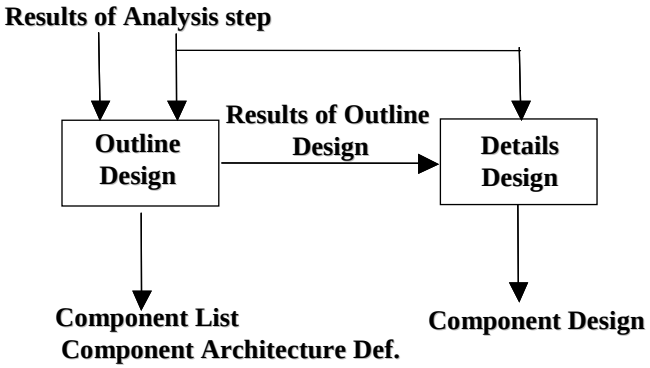


Figure 3. Design Step

The Design Step details the system based on use case diagrams, class diagrams and system architectures that have been defined in the previous Analysis Step. The Design Step is divided into Outline Design and Detail Design. The Outline Design stage identifies the required components and data models, and defines operations for each component and their interfaces, while the Details Design Step details the design of each component itself.

2.1.2.3. Implementation & Testing Step

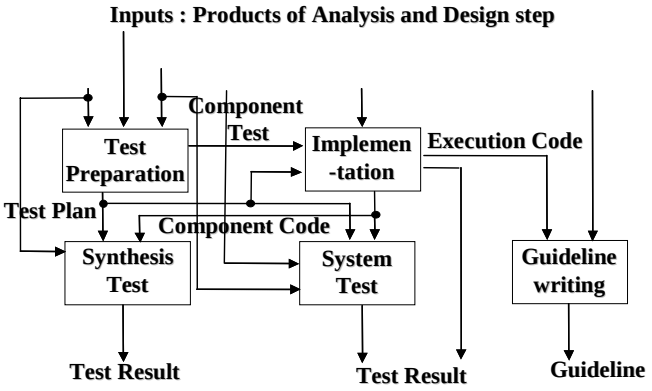


Figure 4. Implementation & Testing Step

The Implementation & Testing Step performs implementation of the designed components and orchestrates them to complete the target system, as well as the step generates various testing scenarios on components and systems and conducts validations and verifications with the scenarios.

2.1.2.4. Surrender Step

.

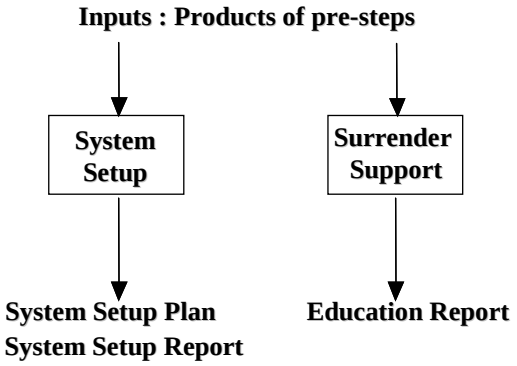


Figure 5. Surrender Step

The Surrender Step is the final step that surrenders the product to the customer with system setup guidelines, operation manuals, and maintenance manuals. The step also helps the users ready to adopt the new system

2.1.3. Limitation of UML-based ADDMe

Since UML does not support discrete event systems, it is hard to directly apply ADDMe to war game simulators, in which specification of component models requires semantics for modeling discrete event systems. In other words, the ordinary UML cannot represent time advancement of an object and an event occurrence at random time which are essential information in discrete event models. Recently, Kim addressed this limitation in UML, and extended UML with DEVS in [3]. The author demonstrated that the problem comes from the limited descriptive power of sequence diagrams, and proposed the Extended Sequence Diagrams in which representation of event occurrence at random time, Kim extended the sequence diagram notation of UML by adding independently ***Activation***[1] under the Activation on the ***Lifeline***[1] of sequence diagram when an event occurs at random time.

Therefore, to deal with war game simulators using the ADDMe process, we extend ADDMe with the DEVS formalism in Section 4.

3. DEVS and DEVS Modeling in War Game

Before explaining the enhancement in ADDMe, this section introduces the DEVS formalism, and presents DEVSimHLA, a tools kit for the DEVS formalism complying IEEE 1516 standard.

3.1. DEVS Formalism

DEVS formalism is a framework for modeling of discrete event systems in a hierarchical, modular manner [4]. Within the formalism, one specifies a discrete event system in two kinds of models: atomic model and coupled model. An atomic model is an unbreakable component of a system, which is represented as follows.

$M = < X, Y, S, d_{ext}, d_{int}, \lambda, ta >$

X : Input event set, finite set;
S : State set, finite set;
Y : Output event set, finite set;
$\delta_{int} : S \rightarrow S$: internal transition ;
$\delta_{ext} : Q \times X \rightarrow S$: external transition ;
$\lambda : S \rightarrow Y$: output function ;
$ta : S \rightarrow Real$

where $Q = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$: total state of M

The first three elements in the 7-tuple are the system's input, state set, and output, and the next four elements give the constraints among the three.

A coupled model is a compound component consisting of atomic models and/or coupled models. The coupled model can itself be employed as a component in a larger coupled model, thereby giving rise to the construction of complex models with hierarchical structures. A coupled model is represented as follows.

$DN = < X,Y,M,EIC, EOC, IC>$

X : input event set , finite set ;
Y : output event set , finite set ;
M : internal model set, finite set ;

EIC: $EIC \subseteq DN.IN \times \cup M.IN$: External Input Coupling ;
EOC: $EOC \subseteq \cup M.OUT \times DN.OUT$: External Output Coupling ;
IC: $IC \subseteq \cup M.OUT \times \cup M.IN$: Internal Coupling ;

Atomic models and coupled models are consists of the total system model by hierarchical structure.

3.2. War Game Simulator and DEVS [9]

Models are specified and the each models are implemented by a scenario. For example, Figure 6 shows a general scenario of training model having four steps of movement, detection, engagement, and evaluation.

Since a war game simulation requires thousands of simulation objects, a war game simulator is usually developed in distributed manner. Therefore, the DEVS formalism also has been implemented with DEVSimHLA to comply IEEE 1516 Standard for Modeling and Simulation High Level Architecture. DEVSimHLA is a simulation tools kit based on DEVS, and includes DEVSim++, KHLAAdaptor, COMLib and ProTester. [10]

Figure 6. General scenario of training model[11]

DEVSim++ is an environment for development of HLA compliant DEVS models using C++. It also supports distributed simulation of DEVS models through RTI(Run Time Infrastructure). KHLAAdaptor is an interface between simulators and RTI software module. COMLib is an easy-to-use communication library based on the TCP/IP library. Finally, ProTester is a protocol tester which is useful for testing a user-defined protocol using COMLib. Fig 7 shows logical architecture for war game simulators developed using DEVSimHLA. The DEVSimHLA tools kit has been successfully employed to many military war game projects including ChungHae (Navy), ChangGong (Airforce), and ChunJaBong (Marine Corps).

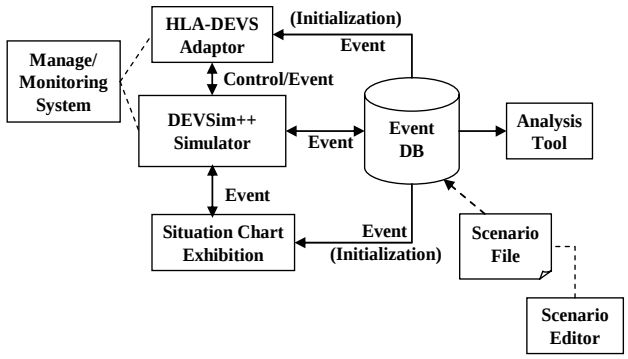


Figure 7. Architecture for War Game Simulators Using DEVSimHLA Tools Kit

4. Embedding DEVS in ADDMe for War game Model

4.1. DEVS embedding process in ADDMe

Among four steps in ADDMe, three steps except the last Surrender step, which is rather far from design and implementation, has to be modified to manage the DEVS formalism. The following section illustrates the embedding process of DEVS with ADDMe, and explains the additional requirements in each ADDMe step.

4.1.1. Analysis Step

As previously mentioned, the ordinary UML-based CBD lacks the descriptive power for discrete event systems, which is the reason that DEVS is to be embedded. Figure 8 describes three essential additional items to employ DEVS into ADDMe. The first is a ***Timing Information Specification***. It corresponds to a time advance in DEVS that means the necessary time for each action of an object. Therefore, for every and each action, time advance is defined according to a doctrine or domain experts. The second inserted item is the ***Extended Sequence Diagram*** that is explained in Section 2.1.3. The diagram supports time advancement of an object and an event occurrence at random time, which is hard for the conventional sequence diagram in UML. The last item is ***Simulator Architecture Definition***. As we employ the DEVS formalism, we also need a simulation architecture being able to manage DEVS atomic and coupled models.

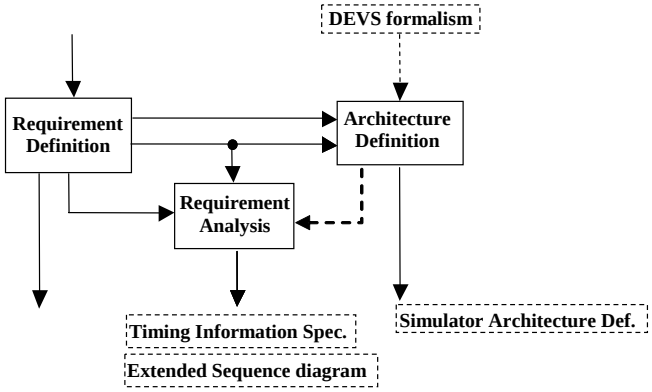


Figure 8. Analysis Step

4.1.2. Design step

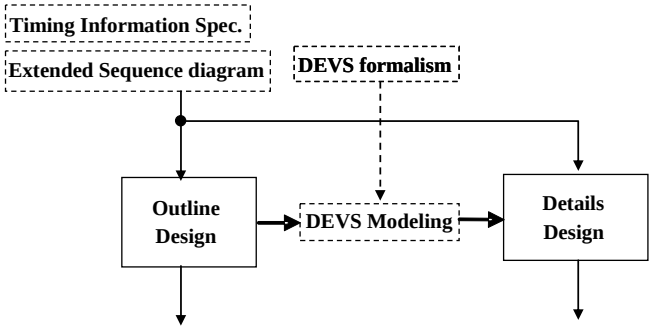


Figure 9. Details Design Step

The main difference in Design Step is the addition of

DEVS modeling. Based on Timing Information and Extended Sequence Diagram, DEVS modeling designs atomic and coupled DEVS component models.

4.1.3. Implementation & Testing step

In Implementation & Test Step, each component model in DEVS is implemented using DEVSim++, an environment for development of HLA compliant DEVS models using C++. Such DEVS implementation provides very useful methodology in the verification of timed I/O sequences. Since discrete event systems are characterized by random-time event occurrences, it is hard to generate all the sequences of inputs and desired outputs for verification from any given timing requirements. In [12], Hong introduced this verification problem, and proposed a systematic methodology to generate and verify timed input/output event sequences based on DEVS.

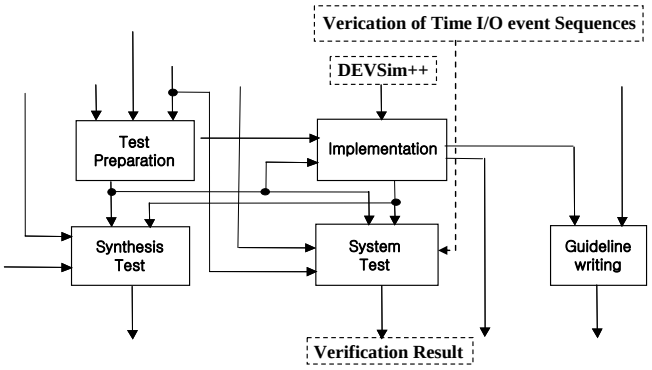


Figure 10. Implementation & Testing Step

4.2. Comparison of two methodologies

Table 1 summarizes the differences between ADDMe and the proposed DEVS-based ADDMe. The major differences are as follows.

The proposed methodology provides a template that can specify a Timing Information, and applies the DEVS formalism when it defines an architecture. The DEVS Modeling Step is added for DEVS specification at the Design Step. The Implementation Step employs the DEVSim++ and the Testing Step performs a verification by the timed Input/Output test vectors for verifying a DEVS model.

Table 1. Comparison of two Methodologies

Step	ADDMe	The proposed methodology
Analysis	▪ Requirement definition ▪ Architecture definition ▪ Requirement analysis	▪ DEVS for detailed analysis with timing information ▪ DEVS of Architecture definition ▪ Using the Extended Sequence Diagram
Design	▪ Outline design ▪ Details design	▪ DEVS for details design Atomic DEVS Coupled DEVS DEVS diagram
Implementation and Test	▪ Descript result After perform test based on Test design document	▪ DEVSimHLA with C++ for implementation ▪ DEVS for generation of timed I/O test vectors

5. Case Study: ChunJaBong Model

The ChunJaBong Simulator is a training simulator which is designed for training of Korean Marine Corps. The simulator has a sequence of three steps: Exercise Preparation Step, Exercise Execution Step and Analysis Step. The Preparation Step creates scenarios to be executed in the scenario database, which is executed by the Execution Step. The execution result is analyzed in the Analysis Step. Functions to be simulated include air battlefield environment, armed military force, battle support equipment, amphibious operation, seaside operation, islands defense operation and others.

This section instantiates the proposed DEVS-embedded ADDMe with the ChunJaBong project, and explains the DEVS embedding process in detail.

5.1. The process of Model Development

5.1.1. Analysis Step

As explained in 4.1.1, the newly proposed Analysis Step requires three additional items. Table 2, Figure 11, and Figure 12 respectively illustrate Timing Information, Simulator Architecture, and Extended Sequence Diagram. All tables and diagrams in this section are from ChunJaBong simulation model that describes a marine doing an engagement. Table 2 is the Timing Information table, and depicts time advances for all actions of a marine. For example, in Step 1-2, we can notice that unit positioning and enemy locating should be done in 10 game cycles, while such description on timing constraint is hard in the original UML-based ADDMe.

Table 2. Template of timing information

Step	Action	Time Advance (sec)
1-1	When receives the /engagement start/ signal, the unit updates target attack information.	5 (Message transmission)
1-2	During the minimum section (game cycle), the unit calculates next position, and locates enemies.	10 (Game cycle)
1-3	If the unit is within an enemy's detection range, the enemy is within the unit's attack range, and ammo is available, then the unit begins the engagement. (2-1)	120

1-4	Encountering any of the below situations, the engagement is stopped. (3-1) ⅰ.The enemy or the unit is destroyed ⅱ.The enemy is out of range ⅲ.Out of ammo	0
2-1	Based on the unit's weapon specification, damages on the enemy are calculated. When the enemy is destroyed, the destruction is informed to the operator, and ammo is reduced by a certain amount. After the engagement, if there still remains ammo, the unit detects and engages another enemy. (1-2)	0
2-2	Based on the unit's weapon specification, damages on the enemy are calculated. When the enemy is not destroyed, the non-destruction is informed to the operator, and ammo is reduced by a certain amount. After the engagement, if there still remains ammo, the unit detects and engages the enemy. (1-2)	0
3-1	If the engagement with enemy is stopped, the unit is switched to the standby mode, and the engagement stop is informed to the operator. After that, the unit is turned to the maneuver mode until receives the next commands from the operator.	5 (Message transmission)
4-1	When receives the target change command, the unit stops the current engagement, and updates the new target information.	5 (Message transmission)
5-1	If the unit receives the engagement stop command, the engagement is stopped.(3-1).	0

From the Timing Information in Table 2, we can obtain the Extended Sequence Diagram as shown in Figure 12.

Figure 11 shows the architecture for ChunJaBong Simulator. A fundamental DEVS model composition is performed by the Technique Control Tool. An atomic model, "Movement", is coupled with another atomic model that is classified by scale of a unit and a mission, a coupled model is constructed that the unit on what a scale can move and perform a mission that was granted. That is a fundamental architecture.

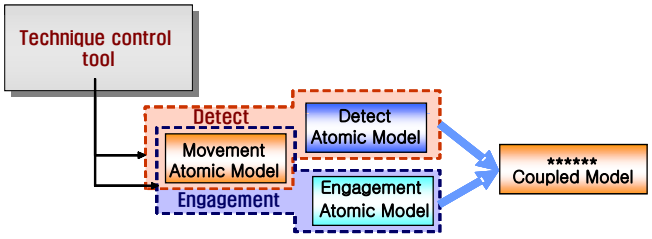


Figure 11. Architecture for ChunJaBong Simulator

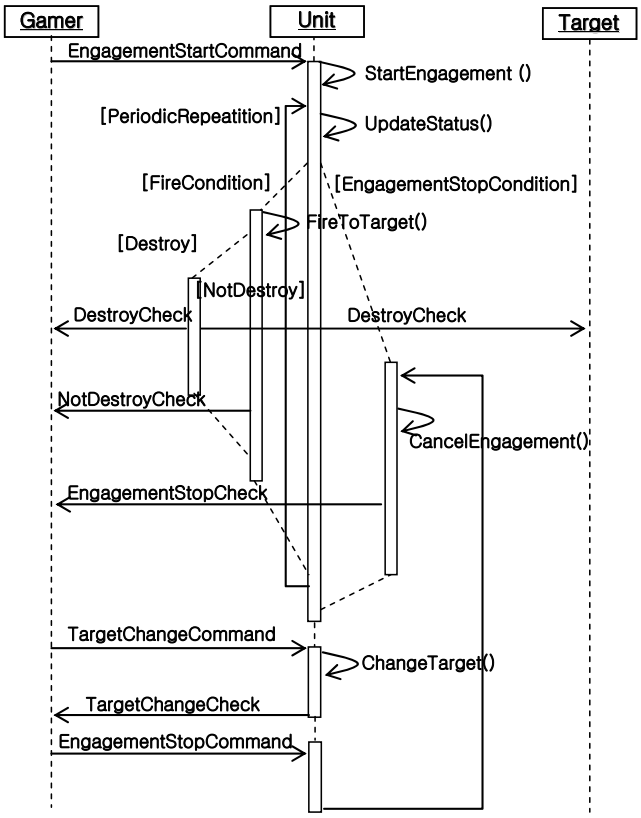
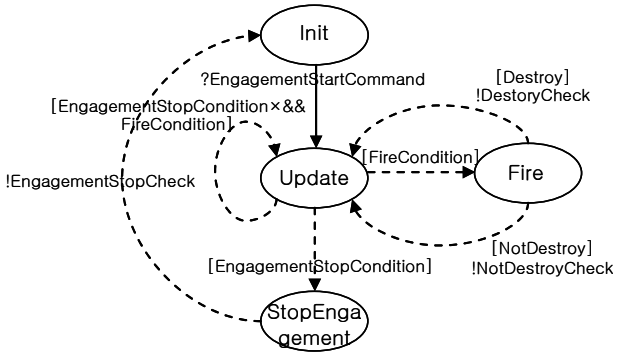


Figure 12. Extended Sequence Diagram

5.1.2. Design Step

In the Design Step, DEVS Modeling Step has been newly proposed as explained in Section 4.1.2. Components are modeled with DEVS atomic and coupled models and Figure 13 details the DEVS model for marine engagement. In the figure, a circle means a single state. A state transition for an input is denoted with a solid arrow, while a state transition with an output is represented by a dotted arrow. For simplicity, we separated a single DEVS graph into 5 graphs in the figure.



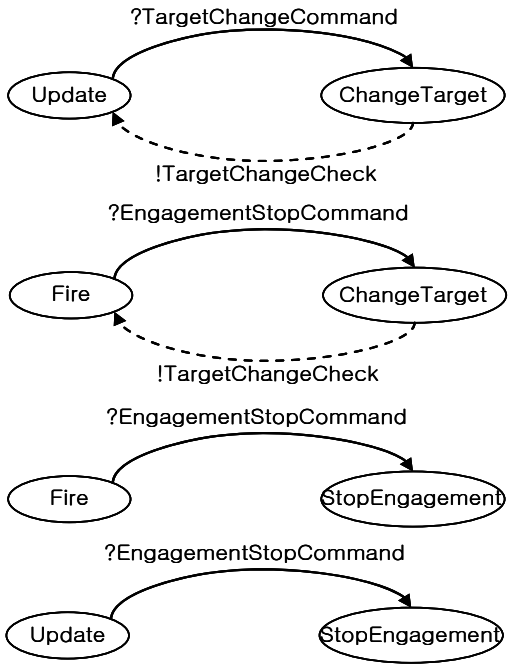


Figure 13. DEVS graph

5.1.3. Implementation and Test step

To implement the DEVS models designed in the previous step, the Implementation Sep uses DEVSimHLA. When implementation completes, the verification methodology introduced in Section 4.1.3 generates all valid input and output event sequences from the DEVS models and from the Extended Sequence Diagrams. The methodology compares two independently generated event sequences to verify the implementation. Table 3 and 4 respectively represent event sequences obtained from Extended Sequence Diagram in Figure 11 and DEVS model in Figure 13. Two identical tables mean that the DEVS model is the valid implementation for the Sequence Diagram.

Table 3. Verification of Sequence Diagram

N	Sequences	V
1	EngagementStartCommand→StartEngagement()→[PeriodicRepeatition]→UpdateStatus()→[FireCondition]→FireToTarget()→[Destroy]→DestroyCheck→[PeriodicRepeatitio]	Pass1
2	EngagementStartCommand→StartEngagement()→[PeriodicRepeatition]→UpdateStatus()→[FireCondition]→FireToTarget()→[NotDestroy]→NotDestroyCheck→[PeriodicRepeatition]	Pass2
3	EngagementStartCommand→StartEngagement()→[PeriodicRepeatition]→UpdateStatus()→[EngagementStopCondition]→CancelEngagement()→EngagementStopCheck	Pass3
4	EngagementStartCommand→StartEngagement()→[PeriodicRepeatition]→UpdateStatus()→[EngagementStopCondition X ,FireCondition X]→[Periodi cRepeatition]	Pass4
5	TargetChangeCommand→ChangeTarget()→TargetChangeCheck	Pass5

Table 4. Verification of DEVS Diagram

N	DEVS-Sequences	V
1	Init→?EngagementStartCommand→Update→[FireCondition]→Fire→[Destroy]!DestroyCheck→Update	Pass1
2	Init→?EngagementStartCommand→Update→[FireCondition]→Fire→[NotDestroy]!NotDestroyCheck→Update	Pass2
3	Init→?EngagementStartCommand→Update→[EngagementStopCondition]→StopEngagement→!EngagementCh eck→Init	Pass3
4	Init→?EngagementStartCommand→Update→[EngagementStopCondition X &&FireCondition X]→Update	Pass4
5	Update→?TagetChangeCommand→ChangeTargetFromUpdate→!TargetChangeCheck→Update	Pass5
6	Fire→?TargetChangeCommand→ChangeTargetFromFire→!TargetChangeC heck→Fire	Pass5

6. Conclusion

This paper embedded the DEVS formalism within ADDMe, a UML-based CBD methodology, to deal with a war game simulator requiring discrete event systems modeling. Embedment of the DEVS methodology is not replacement, but complement of the UML-based OOD methodology. To employ the DEVS' modeling power of discrete event systems, we enhanced the Analysis Step with Timing Information, Extended Sequence Diagram, and Simulator Architecture. Using those newly introduced information, DEVS component models are defined in the Design Step. DEVS modeling also enables the use of the well-defined verification tool between Sequence Diagram and DEVS model. The DEVS adaptation in ADDMe has been case-studied with the currently working ChunJaBong simulator project.

References

[1] M. Priestley, *Practical Object-Oriented Design with UML*, McGraw-Hill, 1996.

[2] "ADDMe v1.1", Agency of Defense Development, 2005.

[3] Do Hyong Kim, "Method and Implementation for Consistency Verification of DEVS Model against User Requirement," Master's Thesis, Dept of EECS, KAIST, 2005.

[4] Bernard P. Zeigler; Herbert Praehofer; Tag Gon

Kim. 2000. *Theory of Modeling and Simulation.* ACADEMIC PRESS.

[5] D. D'sousz and A. Wills, Objects, *Components, and Frameworks with UML: The Catalysis Approach*, Addison-Wesley, 1998.

[6] Kruchten, P.B., *Rational Unified Process, The: An Introduction* (2ed), Addison-Wesley, 2000.

[7] Allen, P. and Frost, S., *Component-Based Development for Enterprise Systems: Applying The SELECT Perspective*, Cambridge University Press – SIGS Publications, 1998.

[8] NamKyu Cho, "Building System from Pieces with Component-Based Software Engineering", KAIST Business School Seminar, 2001.

[9] Tag Gon Kim, "Methodology for Design and Realization of Intelligent Autonomous Engagement Simulation System", Final Report, Systems Modeling and Simulation Lab, Dept of EECS, KAIST, 2006.

[10] Tag Gon Kim and Jae Hyun Kim, "DEVS Framework and Toolkits for Simulators Interoperation Using HLA/RTI", *in Prc. of Asia Simulation Conference/the 6th International Conference on System Simulation and Scientific Computing, pp. 16-21*, October 24-27, 2005 (Invited Paper).

[11] Tag Gon Kim, "Electromagnetic waves Signal environment simulation of helicopter battlefield", Final report, Systems Modeling and Simulation Lab, Dept of EECS, KAIST, 2004.

[12] Ki J. Hong and Tag G. Kim, "Timed I/O Test Sequences for Discrete Event Model Verification." LNAI 3397, pp. 275-284, 2005.