Parametric Behavior Modeling Framework for War Game Models Development Using OO Co-Modeling Methodology

Jae-Hyun Kim* and Tag Gon Kim**
Department of EECS
KAIST
373-1 Kusong-dong, Yusong-gu
Daejeon, Korea 305-701
Tel: +82-42-869-3454
Fax: +82-42-869-8054
jhkim@smslab.kaist.ac.kr*, tkim@ee.kaist.ac.kr**

Keywords: Parametric Behavior Modeling Framework, Co-Modeling Methodology, Automatic Synthesis of DEVS Simulator, War Game Models

The OO co-modeling methodology, previously introduced, employs a layered approach in OO war game models development in which an upper layer models abstract behavior of an object and a lower one models details of the object. Within the methodology military domain experts and M&S experts models an object at the upper and the lower layers, respectively in concurrent manner. This paper proposes a parametric behavior modeling framework for war game models development based on the methodology, which provides a means for military domain experts/users to change model's detailed behavior with no knowledge on modeling semantics. To do so it requires separate implementations between the upper and lower layer objects in which the lower layer object is managed in a dynamic linked library. The simulator would support new algorithms or strategies with minimal cost. To demonstrate the effectiveness of the proposed framework, a naval war game DEVS simulator is exemplified.

1 INTRODUCTION

Given modeling objectives modeling and simulation (M&S) experts should represent only abstract knowledge of the system to be modeled, which meets the objectives. For such abstraction M&S experts should have deep knowledge of the modeled system. However, this is not always the case. In fact, it is practically rare that domain experts - e.g. military officers in war game simulation – develop the simulation model; instead they may define functional requirements to meet modeling objectives, which eventually are transformed to a simulation model by M&S experts. Thus, modeling requires cooperative teamwork between domain experts and M&S experts in a whole modeling process.

There have been some efforts for communicating and co-operation between domain experts and M&S experts using Unified Modeling Language (UML) [1]. However, that approach is horizontal one. It means that domain experts first analyze the system specification using UML, develop several diagrams of UML, and then M&S experts transform the diagrams to DEVS models with additional information necessary for discrete event simulation. Thus, the modeling process in [1] seems to be sequential.

The object-oriented co-modeling methodology [2] employs a layered approach in model development in which the upper layer represents abstract behavior of an object and the lower layer represents details of the object. The model development is a co-modeling process in which M&S experts and domain experts work concurrently at different abstraction layers for identical objects in a co-operative manner.

In most cases, the specification or requirement clearly limits the functionality of the simulator to be developed. However, the developed simulator is often required to be upgraded to accommodate new algorithms or model behaviors during its maintenance. Or a simulator is specified as an open architecture accommodating various algorithms in order to compare performance between algorithms under test.

Parametric behavior modeling framework is proposed to easily adopt new behavioral requirements. Like we set a particular value to a parameter in order to control a program, model behavior itself is modifiable to the simulator in the framework.

The modifiable behavior is limited to the object modeling part in co-modeling methodology. The simulator that implements discrete event models calls associated behaviors or operations implemented in a separate dynamically linked library. After the simulator starts to run,

it searches associated dynamically linked library in its path, and loads the data and code into the memory space of the application process.

This paper is organized as follows. Section 2 discusses the co-modeling methodology. Section 3 describes an object-oriented modeling and introduces UML modeling and DEVS formalism. Section 4 presents the proposed parametric behavior modeling framework and automatic synthesis of DEVS simulators. In Section 5, an example of a navy war game model is developed within the proposed method. Conclusion is made in Section 6.

2 CO-MODELING METHODOLOGY

In this section, we briefly review the co-modeling methodology proposed in [2].

2.1 Modeling Process using UML and DEVS

The co-modeling methodology is analogous to HW/SW co-design in VLSI systems design. HW/SW co-design means the meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design. To do so, system specification is partitioned in hardware and software parts for concurrent job. After that, each part is implemented and then integrated for co-simulation [3] [4].

Co-modeling methodology is analogous to HW/SW co-design and the process is shown by Figure 1. At first, we design the simulator architecture from requirements and specification for a system to be simulated. If M&S experts do not need detailed knowledge on an object in atomic DEVS modeling, they define three sets and four functions according to DEVS formalism. On the hand, if atomic DEVS modeling needs detail knowledge on the object, domain experts defines such knowledge in forms of operations on the object using UML. Then, M&S experts employ the operations as services. Specifications of DEVS and UML models are implemented by DEVSim++ and C++, respectively.

2.2 Advantage

Within the proposed methodology M&S experts and domain experts design simulation models in a closely-coupled manner. The methodology should identify which objects are modeled only at the DES layer and which are modeled at both DES layer and the OM layer. Modeling at the DES layer does not requires specific knowledge of objects which only domain experts understand. However, if M&S experts do not understand detail operations of an object then the operations should be provided by domain experts. In this sense, DEVS modeling is viewed as software and UML modeling viewed as hardware in the HW/SW co-design methodology. Thus, the main advantage of the proposed methodology is a concurrent process in models development.

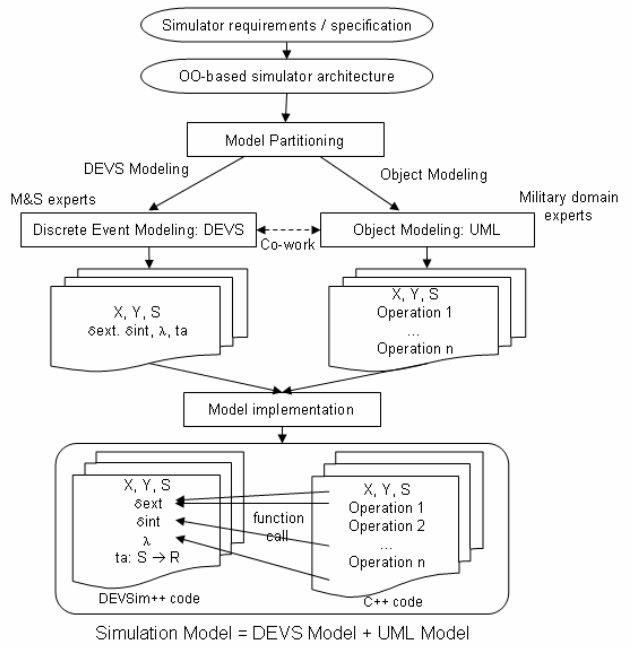

Figure 1 Co-modeling methodology using UML and DEVS

3 UML AND DEVS FORMALISM: BRIEF REVIEW

OO modeling provides a natural and powerful paradigm for representing the elements of a discrete event system and their behavior. In this section, we describe an instruction of UML as an OO modeling language and DEVS formalism for OO discrete event system modeling.

3.1 UML Modeling

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, and documenting the artifacts of an object-oriented system under development [5-7]. It simplifies the complex process of software design, making a blueprint for construction. This sub-section describes only three diagrams in UML, which are used for modeling of an object at the OM layer. They are use case diagram, class diagram, and sequence diagram.

3.1.1 Use Case Diagram

A use case diagram is a behavior diagram that defines a set of use cases and actors and relationships between them. As a behavioral classifier the diagram defines a sequence of actions, performed by one or more actors and a system, which results in an observable value to one or more actors. For system developers, this is a technique for gathering system requirements from a user's point of view.

3.1.2 Class Diagram

A class diagram is a structure diagram that shows a set of classes, interfaces, and/or collaborations and the relationships among these elements. A class includes name, attributes and operations. This diagram is a central modeling technique that runs through nearly all object-oriented methods and represents the static part of a system.

3.1.3 Sequence Diagram

A sequence diagram is an interaction diagram that focuses on the time-ordering of a message between objects. A sequence diagram depicts a sequence of actions that occur in a system which is a very useful tool to easily represent the dynamic behavior of the system. This diagram includes objects and messages in two-dimensional form in nature. On horizontal axis, it shows the life of objects that it represents, while on the vertical axis, it shows the sequence of the creation or invocation of these objects.

3.2 DEVS Formalism

The DEVS formalism specifies discrete event models in a hierarchical and modular form [8]. With this formalism, one can perform modeling more easily by decomposing a large system into smaller component models with coupling specification between them. There are two kinds of models: atomic model and coupled model.

An atomic model is the basic model and has specifications for the dynamics of the model. Formally, a 7-tuple specifies an atomic model M as follows.

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

where

X: a set of input events;

Y: a set of output events;

S: a set of sequential states;

δ_{ext}: $Q \times X \rightarrow S$, an external transition function, where $Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$ is the total state set of M;

δ_{int}: $S \rightarrow S$, an internal transition function;

λ: $S \rightarrow Y$, an output function;

ta: $S \rightarrow R^{+}_{0,\infty}$ (non-negative real number), time advance function.

A coupled model provides the method of assembly of several atomic and/or coupled models to build complex systems hierarchically. Formally, a coupled model is defined as follows.

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

where

X: a set of input events;

Y: a set of output events;

M: a set of all component models;

$EIC \subseteq DN.X \times \cup M.X$: external input coupling;

$EOC \subseteq \cup M.Y \times DN.Y$: external output coupling;

$IC \subseteq \cup M.Y \times \cup M.X$: internal coupling;

SELECT: $2^M - \emptyset \rightarrow M$: tie-breaking selector.

An overall system consists of a set of component models, either atomic or coupled, thus being in hierarchical structure. Each DEVS model, either atomic or coupled model, has correspondence to an object in a real-world system to be modeled. Within the DEVS framework, model design may be performed in a top-down fashion; model implementation in a bottom-up manner.

4 PARAMETRIC BEHAVIOR AND AUTOMATIC SYNTHESIS OF DEVS SIMULATORS

4.1 Parametric Behavior Modeling

The term, parameter, in computer science refers that a value or object which is used to transfer information to or from subprograms or functions. The parameters passed to a function subroutine are more normally called *arguments*.

A parameter is a user-adjustable quantity that affects some behaviors of the program. The program has pre-defined behaviors or functions according to the parameter values. Users assign appropriate values to the input parameters in order to control the behavior of the application or to get calculated results from the input values.

Similarly, a set of parameters is associated with each instances of the *parameterized* model class in a simulator. A model instance of such a class has the same static and dynamic specification. However, the particular values assigned to the parameters of a model instance determine the nature of its specification.

The parameterized model is used to express generalized model behavior with a set of parameters. It is possible for a model to change its behavior within a range of parameter values. However, the parameterized model has limited number of pre-defined behaviors listed in the specification.

It is often required for a simulator to apply new algorithms or model behaviors during its maintenance. Or a simulator is specified as an open architecture accommodating various algorithms in order to compare performance between algorithms under test. In these cases, the parameterized model should be modified to support completely different algorithms, which will cost much.

Parametric behavior modeling framework has a point of view that behavior or algorithm itself becomes a particular value of a parameter. Like we change parameter values, we change behaviors of the simulator unlimitedly.

Co-modeling methodology clearly gives us how we can change behaviors of the model. In the co-modeling methodology, M&S experts and domain experts partition the target system into two parts; discrete event modeling and object modeling. Discrete event model deals with abstracted level of behaviors, mostly in state-transition level. In order to transit between states, decision making or calculation that involves domain-specific knowledge are necessary in most cases.

For example, there is a queue model. The queue buffers received inputs and delivers an input to the idle processors. We can model a queue with abstracted operations such as *enqueue* and *dequeue*. The discrete event queue model handles received inputs with *enqueue* operation in "receive-input" state, and send an item to the processor with *dequeue* operation if it is signaled that the processor is idle. The queue model works independently regardless of detailed algorithms used in *enqueue* or *dequeue* operations.

Implementation of these *enqueue* and *dequeue* operations requires specific knowledge about a queue. Depending on what algorithms are used in these operations,

the complete queue model becomes a First-In First-Out (FIFO) queue, Last-In First-Out (LIFO) queue or etc.

In the co-modeling methodology, M&S experts deal with the discrete event queue model, and "queue" experts implements *enqueue* and *dequeue* operations.

Parametric behavior modeling framework is to provide operations independently from the model. Sets of *enqueue* and *dequeue* operations, combined with the discrete event queue model, generate queue simulators with different behaviors. A particular set of operations is analogous to a particular value of a parameter. Assume that the queue model is developed as a parameterized form. In this case, the requirement of the simulator should clearly specify that the queue model works FIFO or LIFO according to the assigned parameter value. The developed queue model only supports FIFO and LIFO.

Technically, the switch of functions in the application at run-time or load-time becomes possible due to dynamically linked library technology, which is described in the next section.

4.2 Dynamically Linked Library [9]

In computer science, a library is a collection of subprograms used to develop software. A library contains code and data, which provide services to programs that links the library. Executable program and libraries make reference known as *links* to each other through the process known as *linking*, which is typically done by a *linker*. The code and data in a library is copied into a new executable at link time, which is called *static* linking.

Dynamic linking means that the linker only records what libraries the executable needs and the index names or numbers, and the data in the library remains in a separate file on disk. The coping of code and data in the library into the process's memory space is done when the application is loaded or during the execution of the process. This kind of libraries is called a dynamically linked library. The term, Dynamic Link Library (DLL), specifically indicates dynamic libraries in Microsoft Windows environments.

Dynamic loading means that a dynamically linked library loads and unloads at run-time on request. The request to load such a dynamically linked library may be made implicitly at compile-time, or explicitly by the application at run-time.

Traditional static linking binds a program and libraries together at compile-time, and produces one executable code that includes all data and code. In this technology, the program should be re-compiled in order to replace a function. However, dynamically linked library that contains functions to be replaced, is loaded after the application starts. If we implement functions as each separate dynamically linked library, the simulator is able to execute user-supplied functions in the library without re-compiling the simulator program.

4.3 Automatic Synthesize of Simulators

In co-modeling methodology, M&S experts develop DEVS models that represent system behaviors in state-transition level. Domain experts are responsible for developing specific algorithms as individual functions. These algorithms typically require domain-specific knowledge.

Majority of co-work between M&S experts and domain experts in co-modeling is to specify function prototypes. The functions should be designed independently and self-contained as possible.

DEVS models call associated functions inside each state transition functions. The contents of the function, or specifically the level of abstraction of equations used to implement the function, does not affect the model behavior as long as the I/O types of the function is consistent.

Domain experts are able to supply set of behavior functions in the form of a dynamically linked library. Combined with implemented DEVS models, a complete simulator is synthesized by selecting supplied dynamically linked libraries. This selection can be done at run-time. Therefore simulator users easily compare different combination of algorithms. Figure 2 describes these procedures.

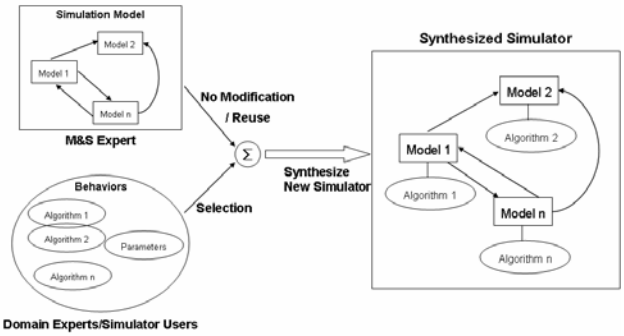

Figure 2 Automatic Synthesis of DEVS Simulator

5 CASE STUDY: WAR GAME SIMULATOR

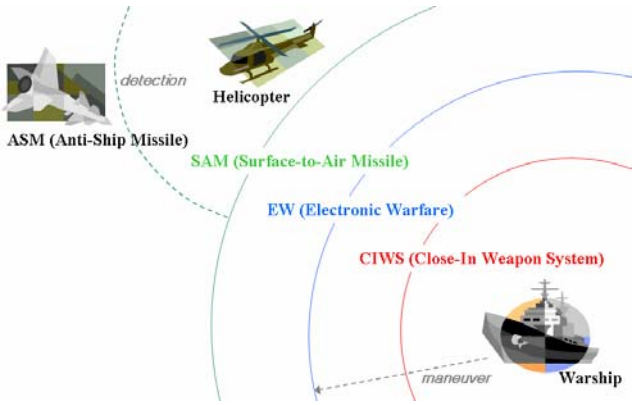

Figure 3 Defense of a Naval Ship Against Anti-Ship Missiles

The proposed parametric behavior modeling methodology is applied to the development process of a war game simulator depicted in Figure 3. The objective of the target simulator is to calculate survivability of a warship and to evaluate performance of AAW (Anti-Air Warfare) and EW (Electronic Warfare) systems in the ship against multiple ASMs(Anti-Ship Missile). The simulator is useful to increase the survivability of the ship by testing various combinations of defense strategies.

The scenario of the war game simulation is as follows:

1. A warship is moving to a certain destination point.
2. The warship is detecting any hostile activities.
3. One or multiple ASMs are fired to the warship one by one with specified delay time. Users are able to define start location of the missiles by relative position from the ship.
4. A warship has three types of defense systems; SAM(Surface-to-Air Missile), EW(Electronic Warfare), and CIWS(Close-In Weapon System). The systems have different effective ranges. The warship tries to shoot down the missiles with the defense systems. The ship also starts to make an evasion maneuver as soon as the missile is detected.
5. If an ASM hits the warship, or all ASMs are shot down, the simulation ends.
6. An optional helicopter may escort the warship and detect hostile activities. The presence of the helicopter expands the range of detection.

The requirement includes behaviors that need specific military knowledge.

1. Maneuver
   A. Warship, Helicopter, SAM, ASM
   B. Warship's Evasion Maneuver
2. Detection
   A. Warship, Helicopter
3. Engagement
   A. Weapon Assignment
   B. SAM-to-ASM
   C. EW-to-ASM
   D. CIWS-to-ASM
4. Damage Analysis
   A. SAM-to-ASM
   B. EW-to-ASM
   C. CIWS-to-ASM
   D. ASM-to-Warship

Each behavior can be partitioned into two parts. One is related to advance simulation time, while the other is instant decision or calculation. After the behavior is partitioned, the part related to the simulation time advance is modeled as a discrete event system. The other part is pure calculation or decision making that specific domain knowledge is probably required to implement. The latter is assigned to domain experts and they are able to model this behavior as a simple function or set of functions.

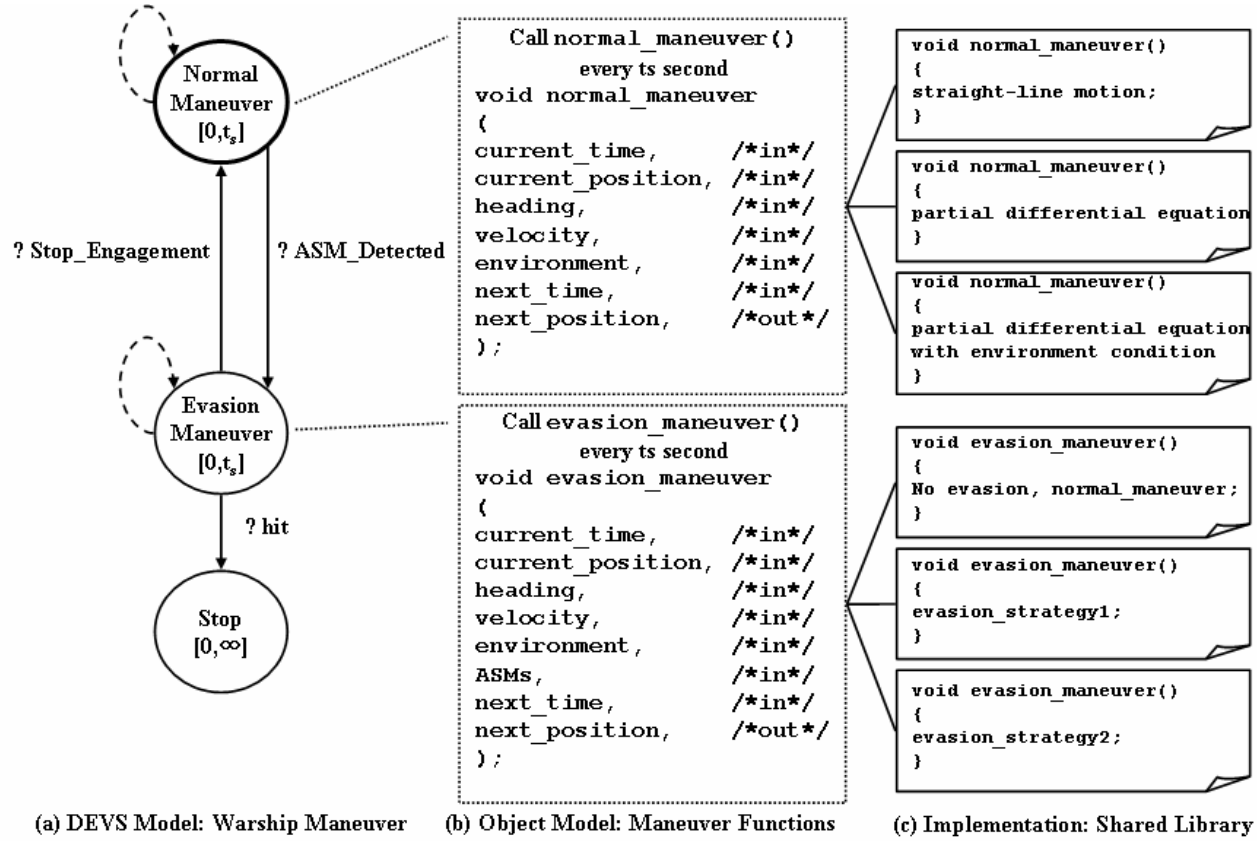

Figure 4 Co-modeling of Warship's maneuver

For example, a maneuver of a platform this is originally a continuous system, can be modeled as a discrete time system in this discrete event simulation. The maneuver model periodically advances certain amount of time (or time-step) and it calculates new position of the platform with new time value. Periodical advancement of simulation time is assigned to DEVS model because it manages simulation time. Calculation of next position is assigned to OM modeling because it requires specific knowledge about platform maneuver, and it does not advance simulation time.

Similarly, detection can be modeled with periodic time advance and testing at each time step whether missiles are detected or not. Engagement behaviors have more complex discrete event models because each weapon system has different control steps that consume different time. Weapon assignment or various damage analysis are decision making or calculation.

Figure 4 (a) shows an example discrete event model that represents maneuver behavior of a warship. The warship has three maneuver states; *normal maneuver*, *evasion maneuver*, and *stop*. *Normal maneuver* state indicates that the ship is not under attack, and it is heading to a certain destination. After a missile is detected, the model receives *ASM_Detected* input event, and the state transits to *evasion maneuver* indicating that the ship starts to make an evasion maneuver against the detected missile. If the missile hits the ship in spite of all defense efforts, the model receives *hit* event that makes the model stop.

Figure 4 (b) presents function prototypes that are used to calculate next position of the ship in normal or evasion maneuver states. The prototype shows input or output arguments of the functions. The prototype should be defined by co-work of M&S and domain experts when the model is partitioned or under development.

Figure 4 (c) displays possible alternatives of maneuver functions. The maneuver function may assume that the ship moves piece-wise straight line for each time step. The maneuver function may use partial differential equations that represent the motion of the ship, or more precisely, the function may also consider the weather or geographical conditions. If the candidates of normal maneuver show different abstraction of same behavior, those of evasion maneuver lists different strategies. In the example, there are three strategies; no evasion, evasion strategy 1 and strategy 2. Unlike pre-defined parameterized model, the number of strategies that the simulator can evaluate is not only 3, but unlimited.

Each function is developed as a separate dynamically linked library. The simulator that includes the DEVS maneuver model of Figure 4 (a) will selectively use one of candidate libraries. The users of the simulator explicitly select the libraries at run-time. The simulation results will be different according to what library it uses. By this way, users can test various combinations of defense strategies without recompiling the simulator itself. Also, users are able to add new strategies easily by making a function with pre-defined prototype as a dynamically linked library.

The example simulator is developed under Windows XP and Microsoft Visual Studio. We use Dynamic Link Library (DLL) technology. Figure 5 shows the screen shot of the developed simulator. You can find lines of texts and buttons below objects and weapon systems in the picture. The texts are lists of parameters and descriptions of the selected algorithm for the platform or weapons. The button is used to select a DLL file that contains behaviors for the platform or weapons. The simulation results are automatically displayed with tables and graphs by Excel automation (See Figure 6).

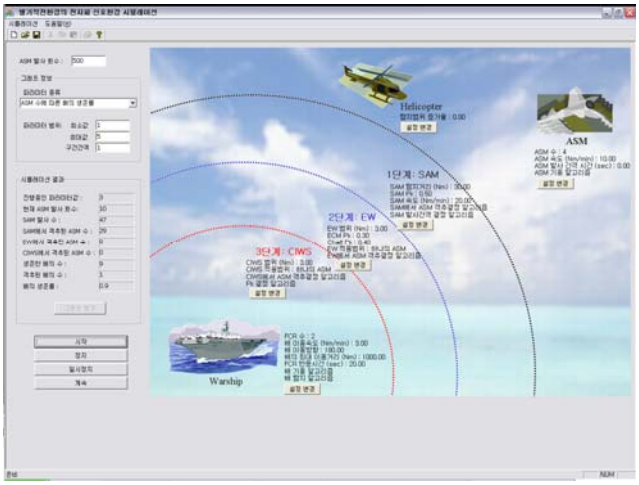

Figure 5 Screen shot of the Simulator (Korean)

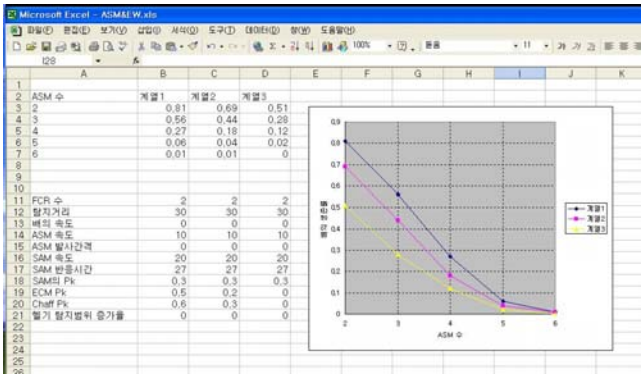

Figure 6 Simulation Result (Excel Automation, Korean)

6 CONCLUSION

This paper proposes a parametric behavior modeling methodology using co-modeling methodology. The main objective of the framework is to provide a flexible ways to change model behaviors to simulator users.

The co-modeling methodology partitions model specifications into two modeling parts; discrete event modeling and object modeling. The M&S experts define time associate output and transition function using DEVS formalism by use of operations defined in the object modeling. The discrete event models are specified regardless of the algorithms used to implement object modeling operations.

The object modeling operations are implemented as a dynamically linked library which is dynamically loaded after the simulator application starts. The methodology enables for simulator users to alter model behaviors or algorithms without recompiling the simulator.

A case study demonstrated effectiveness of the proposed approach. This approach may be applied to more complex war game modeling projects.

7 REFERENCES

[1] Su-Youn Hong and Tag Gon Kim, "Embedding UML Subset into Object-oriented DEVS Modeling Process," in *Proceedings of the Summer Computer Simulation Conference*, San Jose, California, USA, July 2004, pp. 161-166.

[2] Chang Ho Sung, Su-Youn Hong, and Tag Gon Kim, "Layered Approach to Development of OO War Game Models Using DEVS Framework," in *Proceedings of the Summer Computer simulation Conference*, 2005

[3] Jay K. Adams and Donald E. Thomas, "The Design of Mixed hardware/software systems," in *Proceedings of the 33rd annual conference on Design automation*, Las Vegas, Nevada, USA, June 1996, pp. 515-520.

[4] David W. Franke and Martin K. Purvis, "Hardware/Software CoDesign: A Perspective," in *Proceedings of the 13t international conference on Software engineering*, Austin, Texas, USA, 1991, pp. 344-352.

[5] Booch, Rumbaugh, and Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesely, Reading, Massachusetts, 1998.

[6] Rumbaugh, *Unified Modeling Language Reference Manual*, Addison-Wesely, Reading, Massachusetts, 1999.

[7] Mark Priestley, *Practical Object-Oriented Design with UML*, The McGraw-Hill Companies, 1996.

[8] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation*, Academic Press, 2000.

[9] http://en.wikipedia.org/wiki/Library_(computer_science)