EVENT-ORIENTED CONTROL FUNCTIONS FOR ENHANCING DEVELVOPMENT PROCESS OF WAR-GAME SIMULATORS

Se Jung Kwon[(a)], Changbeom Choi[(b)], Tag Gon Kim[(c)]

Dept. of Electrical Engineering, KAIST
Daejeon, KOREA

[(a)]sjkwon@smslab.kaist.ac.kr, [(b)]cbchoi@kaist.ac.kr, [(c)]tkim@ee.kaist.ac.kr

ABSTRACT
The development processes of discrete event simulation software may not be straightforward. The processes can be interrupted repeatedly by modified requirements. This paper proposes an approach for making the iterative processes efficient. In order to keep the development cost low, we seeks to avoid modifying the simulation model inside as much as possible. Instead, our proposed work enables the same simulators to generate the other execution results adding additional information. This approach can be enabled with the Event-based Simulation concept and the Event-oriented Control Functions that are mapped to concerned events. The simulation engine can handle the input/output level data by accessing events. In this way, users can control their simulation simply by describing the functions mapped to events, instead of modifying simulation models. This paper also includes case studies to support contributions, assuming that a war-game model has been developing.

Keywords: discrete event systems modeling and simulation, event-based simulation control, DEVS formalism, war-game simulator development

1. INTRODUCTION

A software development process (or life cycle) means a structure that is imposed on the development of a software product. Many studies and applications in the Software Engineering community have been published for efficient development and maintenance. Similarly, the development processes can be applied for discrete event simulation software, as well.

Among the discrete event simulators, the S/W development processes of the war-game simulators have unusual features in contrast to other simulators. Generally, it is hard for developers to understand the model behavior of military domains. Hence, the stakeholders should suggest the objectives of their ordered simulator and behaviors of real military systems. Stakeholders may be fully able to suggest what to do. However, not only do they propose ambiguous suggestions based on the real systems' behaviors but they also realize the need to change the requirements based on the results of simulators or other miscellaneous reasons

According to an exploration or alteration of requirements, simulation models must be redeveloped during the development process. The repetitions of model modification and redevelopment cause an increase in the development time and cost. Assuming that the iterative developments and changing requirements are inevitable, this paper attempts to enhance the iterative processes by reducing the cost of developments.

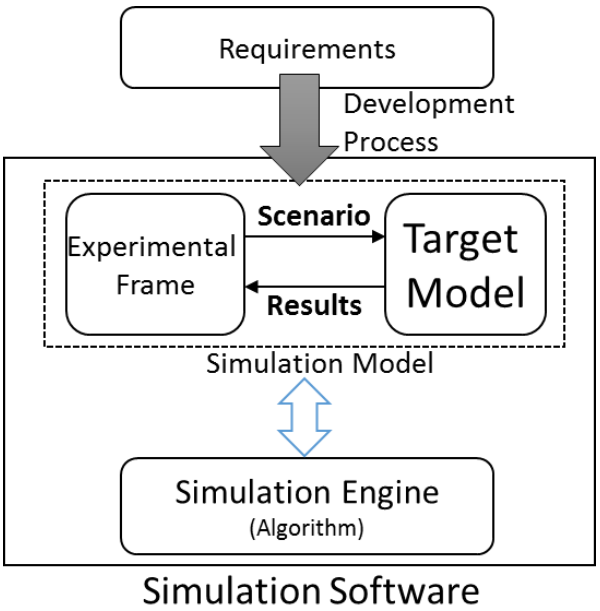


Figure 1: Compositions of Simulation Software

Assuming that the simulation model consists of an Experimental Frame (EF) and a target model, as depicted in Figure 1, an experimental frame generates scenarios for testing or analysis. As the requirements change, developers will try to modify the experimental frame at first and try to make it generate modified scenarios for proper results. However, in most cases, a specific experimental frame cannot deal with new scenarios, even though it would be a better modification approach due to the simplicity. To apply the new requirements right, the inside of target model should be

modified. The modification of the whole models has perfect modifiability, but it causes higher development costs as depicted in Figure 2.

Hence, a key point of this paper is a model development method that has higher modifiability than modification of EF and lower development cost than modification of whole models, like the grey area of the Figure 2. By applying the approach to the existing iterative processes, the cost reduction of the development process can be achieved.

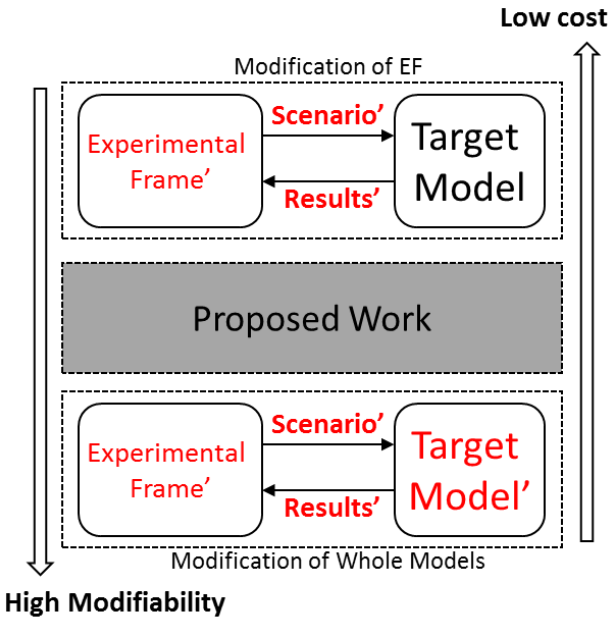


Figure 2: Existing Modification Approaches

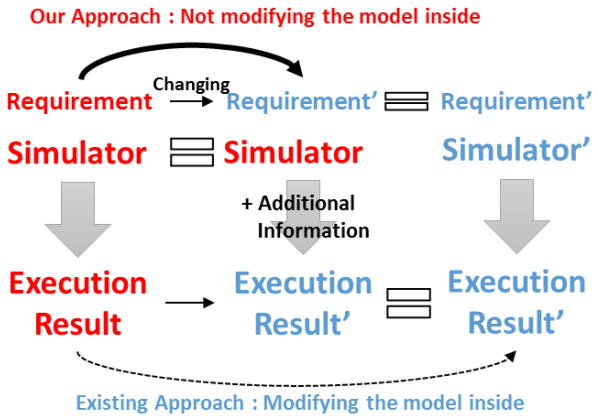


Figure 3: Distinction between Existing Approach and Our Approach as the Requirements Change

Figure 3 describes an approach of this paper, which does not modify the model inside as much as possible. Instead, our approach enables the same simulators to generate the other execution results, according to new requirements. In other words, our approach does not modify syntaxes of models, but it modifies semantics of simulation. This can be possible with described additional information, instead of modification.

For the application of the additional information to the simulation model, this paper adopts an event-based simulation concept, which is a basic execution method for discrete event simulations (Cota and Sargent 1992, Zeigler *et al*. 2000). An event, as a simulation unit of event-based simulation, is listed on an event list and executed in order to affect state transitions of a model. Although the objective models of the simple event-based simulation are event-oriented functions and global states, other object-oriented discrete event models can also be executed with additional interpreter algorithms. Each executed model generates time events and data events for the progression of their simulation. Time events schedule their next execution when the model should be executed, and data events are transmitted to other models with data (messages). The simulation engine cannot access the inside of object-oriented models (e.g. state variables). By accessing the generated events from models, the simulation engine can handle the input/output (I/O) level data. Although object-oriented reusable models can also reduce the development cost (Kim 1996), our approach concerns the I/O level data, not the model inside.

To handle the I/O level data, we define an *Event Control Model* (ECM), which consists of global states and *Event-oriented Control Functions* that are mapped to events. When some models should present different behaviors at the run-time, the same models with additional functions are executed through the event-based simulation engine. When the events are scheduled on the event list at the run-time, the mapped events are passed through the related control functions. For example, the variables of data events can be modulated by the mapped functions, and the behaviors of the models become different without modifying the model inside.

It is true that the model inside has to be modified at one time or another as the requirement is changed repeatedly. Nevertheless, there exist enough empirical evidences that a simulator shows the other behavior with the additional ECM without any modification of the model itself. They can be shown in Chapter 3.2 and Chapter 4.

The contents on which this paper focuses are mainly related to the modulation of variables in data events. By utilizing the events modulation and reducing development costs, these control approaches can make rapid prototyping (Martin 1990) more valuable against the sudden (or planned) changes of scenarios. In particular, the ECM amplifies some development strategies including successive prototyping like the *Sawtooth* model (Rowen 1990).

The target of this paper is a *Discrete EVent systems Specification* (DEVS) model (Zeigler *et al*. 2000), which is one of the most frequently utilized system specifications to model discrete event systems in the real world. Since the event-based simulation algorithm is not limited to DEVS and can be applied to other discrete event simulations by designing proper

interpreter algorithms, this paper can also be extended to other discrete event simulations.

This paper is organized as follows: Section 2 presents the event-based simulation concept. Section 3 explains our proposed work, which contains the specifications for the event-oriented control function and its applications, and Section 4 illustrates case studies for the development process of war-game simulators. Finally, Section 5 concludes the paper.

2. EVENT-BASED SIMULATION

Event-based simulation is one of the most efficient and basic discrete event simulation strategies because of its simplicity. Event-based simulation works by prescheduling all events in an event list (Zeigler *et al.* 2000). In this view, Discrete Event Systems (DES) can be specified to the event-oriented functions mapped to events (or state transitions), which are as units in contrast with the object-oriented concepts. The event-oriented models consist of functions mapped to events and global variables modified by functions. The simulation engine executes a function mapped to an event by extracting it from the event list, and the executed functions insert newly generated events to the event list for scheduling. Since the event-oriented models, which are not passive models, have scheduling parts for events in contrast to object-oriented models, the event-oriented models are not separated from the execution algorithm. On the contrary, object-oriented models, e.g., DEVS models, are passive models and should be separated from the simulation engine. The following algorithm describes the event-based simulation concept.

```
T_global // current simulation time
EventList
// List of sorted event(time, target-function)

Simulation_Run()
while ( Event-List is not empty )
   first = top of EventList
   delete the top of EventList
   T_global = first.time
   execute first.mapped-function
 End while

Schedule_New_Event(time, target-function)
// called by functions mapped to events
  create an event with the pair( time, target-function )
  insert the event to Event-List
```

Algorithm 1: Simple Algorithm of Event Scheduling

Due to characteristics of object-oriented modeling theory, the models need interpreter algorithms in order to be executed by event-based simulation algorithm as depicted in Figure 4. For example, the execution of DEVS models with event-based simulation can be performed with a pre-process algorithm for flattening the hierarchical structure and a mediation algorithm between different interfaces (Kwon and Kim 2012) because the DEVS models are structured hierarchically and systematically. Although the interpretation causes a little degradation of simulation performance, executing the object-oriented models by the event-based simulation engine may cause improved speed-up.

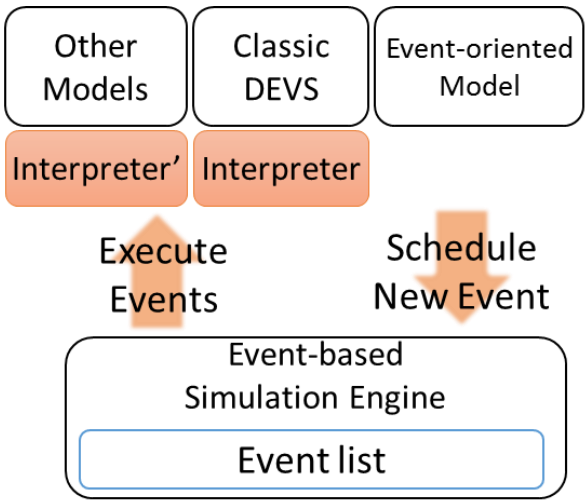


Figure 4: Event-based Simulation

3. EVENT-ORIENTED CONTROL FUNCTION AND ITS APPLICATION

This chapter describes the specifications of *Event Control Model* (ECM), including the *Event-oriented Control Function*. Its simulation and its applications are also covered.

3.1. Event Control Model with Event-oriented Control Functions

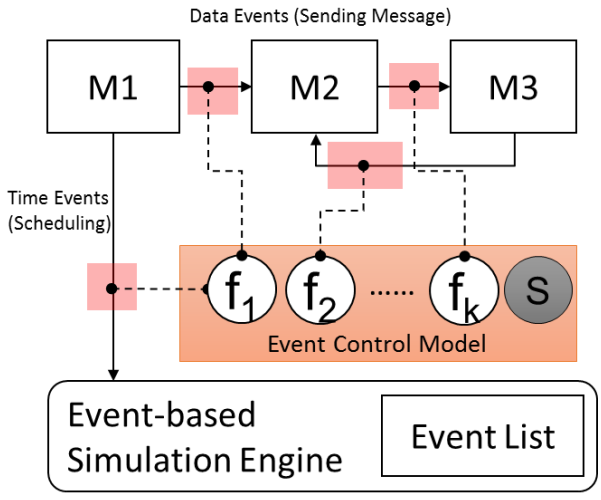


Figure 5: Accessible Points of ECM

The events, occurring state transitions, are listed on the event list of event-based simulation engine as mentioned above. The variables of events inside are public and accessible by the simulation engine as depicted in Figure 5. There are two types of state transitions, time events (E_{time}) and data events (E_{data}).

When the event occurs, the concerned event-oriented functions try to interfere with the occurred events inside.

We define *Event Control Model* (ECM) for the event-oriented control. The specifications of ECM are the following:

- $Ev = \{E_{time}, E_{data}\}$ is a set of events.

- $E_{time} = <T, t_N>$
  T is a target function (model).
  t_N is a next scheduled time.

- $E_{data} = <T, t_N, \{V_i\}>$
  T is a target function (model).
  t_N is a next scheduled time (Usually zero).
  $\{V_i\}$ are variables.

- $ECM = <\{f_i\}, S, SELECT>$
  f_i is an event-oriented control function.
  $: Ev \times S \rightarrow \{Ev \cup \emptyset\} \times S$
  S is set of global states.
  SELECT is a tie-breaking selection function.
  $: 2^{\{f_i\}} - \emptyset \rightarrow f_i$

The definitions of two events are a little different. A time event consists of a target model (a source model itself) and a next scheduled time. A data event consists of a target model (a destination model), a next scheduled time and variables. Though the next scheduled time of data events is usually zero, the user can handle the value to affect a time delay to the execution of events for the advance control.

The key point of ECM is the *Event-oriented Control Function*, f_i. A control function generates a modified event from a generated event of a model, or it can eliminate the event with a certain condition. For the cases that control functions need to store information, the global states are included in the specifications of ECM. The select function exists for resolving the priority problem because two or more functions, mapped to a same event, can be in conflict. The idea of the specifications is borrowed from the event-oriented models and DEVS formalism.

```
Schedule_New_Event(time, target-function(T) )
// called by functions mapped to events
  create an event(Ev) with the pair(time, T)
  for each mapped control functions fk
      Ev = ECM.fk( Ev )
  insert Ev to Event-List
```

Algorithm 2: Modified Algorithm for ECM

The algorithm 2 for executing the ECM is quite simple and has just two lines of added codes compared with the original *Schedule_New_Event* function in Algorithm 1. Before the generated events are inserted into the event list, the simulation engine calls the proper functions of the ECM.

Figure 6 shows the simulation environment of proposed work. The target model can be decomposed to events manually or automatically. Users can select proper events according to the control objectives, and designs the control functions and states. The implemented ECM from the design is executed with the original target model, and it interferes in the event scheduling. The algorithm 2 is embedded in the interpreter algorithm in the event-based simulation engine.

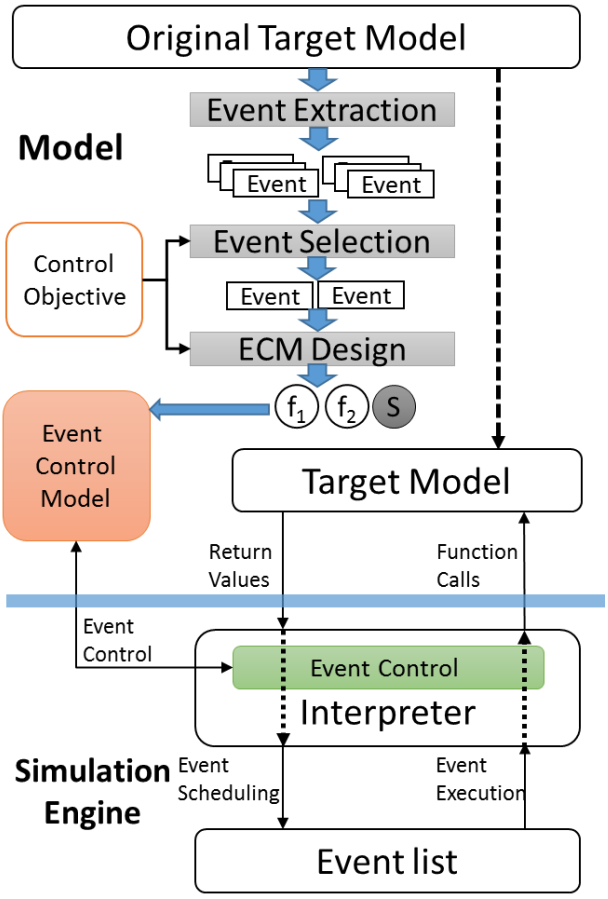


Figure 6: Design Process of the ECM and Proposed Simulation Environment

3.2. Applications

A primary usage is modifying variables of messages. Even if the model inside or state variables are unknown or fixed, the variables of I/O messages should be opened to the simulation engine. By describing event-oriented control functions, users can modulate the variables inside events. In the military domain, to simulate the real operations of equipment in detail, the model behavior usually includes the random variables due to the existence of errors of real operations. Though a certain model parameter was classified as non-prime value and became fixed value in the early stage of development, the value can be changed to the randomized value with a simple control function, which generates a modulated event with a random variable from an original event.

The simplest example is found in a Single Server Queuing Server model (known as a GBP model). A control function with a probability function can

modulate the processing time of a job from the Generator (G). This type of usages can be applied to various situations, e.g., fault injections to messages, adding probability functions, and so on. One of the detailed examples is described in Chapter 4.1.

If a simulation model consists of many homogenous events that have to be modulated, the ECM can be applied to not only an event, but also events in a lump. The representative applications of such cases involve the environment variables. The entities, including combat entities of the battlefield, are influenced by environment variables, e.g. natural environment (ex. temperature, humid, wind, etc.) or a certain global effect to the simulation area (ex. communication noises). Instead of an environment model and I/O coupling for environment variables, just a few event functions can simulate the environmental effects. It is simpler than ECM to model environmental characteristics by using global variables. However, if the initial model does not cover the environment variables the first time, the ECM can be much better solutions for adding the variables to the whole or most of the models.

Extending the usages mentioned above, users could reduce the developing costs of new. As the requirements are changing, an unplanned model may need to be newly developed. Perhaps users have no confidence that the model needs to be developed or not. In this case, it can be efficient that they check the results in advance by describing an ECM. It may be no matter that the target model has to save some information into state variables because the proposed ECM has global variables for states. The detailed example is described in Chapter 4.2.

Applying to the development processes, there are mainly two cases using the ECM. One is rapid prototyping for the refinement of ambiguous requirements in the beginning of development. While the prototyping for the software mainly handles requirements for GUI or representation, the prototyping for simulators mainly handles the behaviors of target models. After developers have made the initial model at the early stage, they make several temporal prototypes using ECM quickly and provide them to the stakeholders for acquiring their opinion iteratively. The iterative process can reduce ambiguity of requirement, and the proposed ECM may reduce the cost of iteration.

The other is the unexpected modification of requirements during the development. Similar to the above case, developers can explore whether the modified requirement is feasible or not as preceding researches. The difference is that the ECM in this case may not be discarded and can be succeeded as continuing the development process.

4. CASE STUDY

This chapter will show the applied examples of ECM during development of a war game, which had been developed for the Korean military actually (Seo *et al.* 2011). The brief scenario is illustrated in Figure 7. This war-game simulation model is developed for analysis and acquisition of underwater warfare. There are four types of combat entities: a submarine, a surface ship, decoys, and a torpedo. The attacking platform is a submarine, and the target platform is a surface ship. The surface ship launches the decoys as counter measures according to stored strategies against the torpedo's possible paths. While the torpedo traces decoys, the surface ship can evade the opponents.

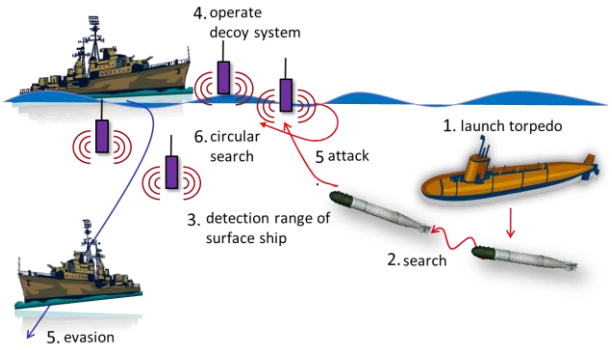


Figure 7: Brief Scenario
(Kwon *et al.* 2011, Seo *et al.* 2011)

The objective of this simulation model is to evaluate the counter-measure tactics against the torpedo system. From the results, we can determine how various factors, such as tactics and the performance of underwater weapons, influence the effectiveness of the system. Experimental results can support assessment of anti-torpedo countermeasure effectiveness.

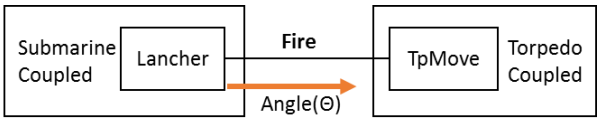
We assume two cases of requirement changing during development process of the simulation model. One is a simple changing of a model parameter. The other is that stakeholders demand development of a new model for an added entity.

Since the original model was based on DEVS formalism, a little knowledge is needed to understand the example, e.g., the hierarchical structure of DEVS models (*Coupled models* and *Atomic models*). The simulation engine for the war-game model is E-DEVSim++ (Kwon and Kim 2012), which was implemented for executing DEVSim++ (Kim *et al.* 2011) models with event-based simulation. The E-DEVSim++ has been extended for this proposed work. The extended E-DEVSim++ provides mapping API between events and event functions. Users can add event-oriented control functions to mapped events with the original models. The functions should take an object point of an event as function parameters and return whether the event is eliminated or not. Otherwise, there is no limitation of implementation for states of ECM.

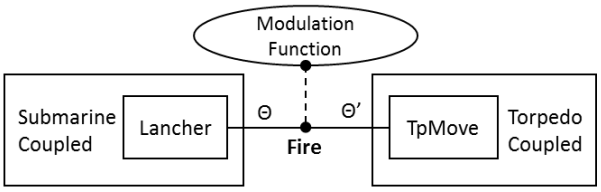
4.1. Example 1: Modulating Model Parameters

The target parameter cited in this chapter is the angle of fire in an attack command, which is generated by the C2 (Command and Control) model of submarines and transmitted to the torpedo model as a message. We assume that the angle of fire was a fixed value in the

initial model structure, as depicted in Figure **8**.(a), and should become randomized with a certain probability function for representing the real error of equipment.



(a) Initially developed model structure



(b) Modifying the parameter with ECM

Figure 8: Modulating the angle of fire

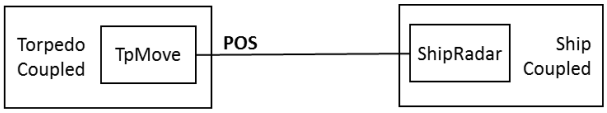
Without modifying the submarine model, adding a control model can handle the angle variable such as the below function. The data event, *Fire*, has a variable for the angle of fire, named as 'Angle'. The control function reads out the value from the *Fire* event and stores the randomized value with the exponential distribution function to the inputted event. The control function is embedded in a C++ class for ECM without any states. It is only the user's job that they register the class to the simulation engine (E-DEVSim++) without any modification of simulation model.

```
bool angleControlFunction( CEvent *ev ){
  double v
     = genExponential(ev->GetValue("Angle"));
  ev->SetValue("Angle", v);
}
```

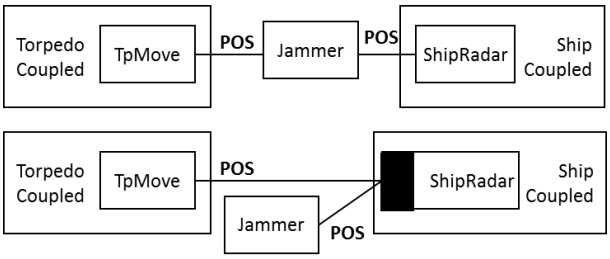
4.2. Example 2: Substitution of a new model development

This example is about the extension of the anti-torpedo simulation model by adding a jammer model, which generates air bubbles or noise to prevent the sonar of torpedoes from detecting our forces. Since the original simulation model included only decoy systems as counter measures against a torpedo, there was actually a demand for additional counter measures of the surface-ship model (Kwon *et al*. 2011).

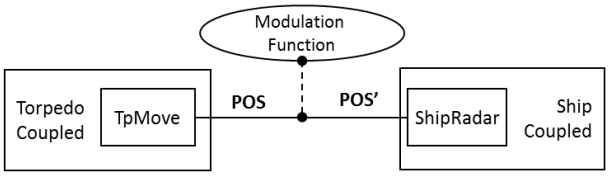
Many instances represent the behavior of the jammer from the initial structure of the simulation model, as depicted in Figure 9.(a) and (b). One of them is that the jammer model is placed for interfering with the position messages between the torpedo model and the radar model of the ship. The other is that radar model gathers the position of the torpedo and the jammer for deciding whether the position information is eliminated or not. The structure of the actual model was developed like the latter. A launched jammer generates its position and sends the message to the radar model. By calculating the distance between two positions, the simulation model decides the success of jamming noise.



(a) Initially developed model structure



(b) Previous solution by adding a new model



(c) Modifying with only ECM

Figure 9: Interfering the position messages

Applying the ECM to this case, the model structure becomes like Figure 9.(c). The event function is attached to the position message, and it decides the elimination of position information instead of the jammer model. Accordingly, users do not have to add a new model or modify the model inside for the extension of simulation model. From the model's pre-development stage, users and stakeholders can know whether their changed requirements are feasible or not. If the model with ECM is enough to be passed for the next step (deployment or analysis), the ECM will be kept. If not, developers may hold the ECM models and redevelop them from the original models. In this time, the model with ECM can be used for testing of the developed model.

```
Location *POS;     // x, y, z;
double operatedTime = 0.0;
double lifeTime  = 100.0;    // Life time of jammer;
int State;              // O: OFF, 1: ON, 2: END
double jamRange; // Operating range

// attached to the fire-order messages of decoys
bool jammerON( CEvent *ev ){
   State = 1;
   operatedTime = ev->GetTime();
   POS  = (*Location)ev->GetValue( "POS" );
}

// Attached to the POS messages from the ship
bool jammerControlFunction( CEvent *ev ){
```

```
 if( State == 1 ){
  if( ev->GetTime() - operatedTime > lifeTime ){
   State = 2; // The life time is over.
   return true;
  }
 }
 if(GetDist(ev->GetValue("POS"),POS) < jamRange )
  return false; // The event should be eliminated.
 return true;
}
```

The actual implemented C++ codes for substituting the jammer model are shown above. To substitute the jammer model, two control functions are needed. The first function substitutes for launching the jammer by attached to the fire-order messages of decoys, which will be operated at the same time with the jammer. When the ship launches the jammer model, the jammer function reads out the operation time and location information by interfering with the data events. At this time, the 'State' is changed from zero to one. The second function is attached to the POS messages from the ship model to the torpedo model. The function will be operating against all the position messages from the ship model, but it does not operate until the 'State' is changed. When the 'State' is changed and is not two, the jammer control function decides whether the location information is eliminated or not by calculating the distance between the jammer and the ship. When the control functions return 'false', the simulation engine knows that the events should be eliminated and throws out the events.

5. CONCLUSION

This paper proposes an event-oriented specification for simulation control, *Event Control Model*, enabled by the event-based simulation. Specifically, it can help the development of war-game simulators. The two case studies in Chapter 4 show how that can be possible. The characteristics of war-game simulators have been mentioned as the motivation of this paper. Nonetheless, this paper is not limited to the military domain and it can be extended to the similar domains that have much exclusive knowledge.

Against the ambiguous requirements or the changed requirements of the similar domains, the ECM can interfere with the events that are listed in the simulation engine. The ECM can be applied to various cases, e.g. applying environment variables to a mass of events or a substitution of new models. The applications, which can have event-oriented functions without modifying the model inside, can make various iterative development processes efficient.

It is true that many further studies are needed. We have not proposed a methodology or a full development process using ECM yet. An extended paper about those advanced contents will be published in the near future. The extended methodology should include the criteria for deciding whether the ECM is applicable or not for various cases. The usages of ECM are also extended to various cases, e.g., events generations, events deletions, logging/proving, and so on. It can be shown with more various case studies that are not limited to the war-game simulators.

ACKNOWLEDGMENTS

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD140022PD, Korea.

REFERENCES

Cota, B.A., Sargent, R.G., 1992. A Modification of the Process Interaction World View. *ACM Transactions on Modeling and Computer Simulation*, 2(2), 109-129.

Kim, T.G., Ahn, M.S., 1996. Reusable Simulation Models in an Object-Oriented Framework. In: Zobrist, G.W., Leonard, J.V., eds. *Object-Oriented Simulation: Reusability, Adaptability and Maintainability*, USA: IEEE Press.

Kim, T.G., *et al*., 2011. DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation. *The Journal of Defense Modeling and Simulation*, 8(3), 129-142.

Kwon, S.J., *et al*., 2011. Effectiveness Analysis of Anti-torpedo Warfare Simulation for Evaluating Mix Strategies of Decoys and Jammers. *Proceedings of AsiaSim '2011*. Seoul (Korea).

Kwon, S.J., Kim, T.G., 2012. Design and Implementation of Event-based DEVS Execution Environment for Faster Execution of Iterative Simulation. *Proceedings of Spring Simulation Multiconference, Symposium on Theory of Modeling and Simulation (TMS'12)*. March 26-29, Orlando (Florida, USA).

Martin, J., 1990. *RAD, Rapid Application Development*. USA: MacMillan Publishing Company.

Rowen, R.B., 1990. Software Project Management under Incomplete and Ambiguous Specifications. *IEEE Transactions on Engineering Management*, 37(1), 10-21.

Seo, K.M., *et al.*, 2011. Measurement of Effectiveness for an Anti-torpedo Combat System Using a Discrete Event Systems Specification-based Underwater Warfare Simulator. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 8 (3), 157-171.

Zeigler, B.P., Praehofer, H., Kim, T.G., 2000. *Theory of Modeling and Simulation*. 2nd ed. USA: Academic Press.

AUTHORS BIOGRAPHY

Se Jung Kwon received his B.S. in Dept. of Computer Science of KAIST in 2009 and M.S. in Department of Electrical Engineering of KAIST in 2011. He is currently a Ph.D. student in the Department of Electrical Engineering at the KAIST. His research interests include simulation algorithms for DES, DEVS execution environments, and hybrid systems M&S.