Adaptive Discrete Event Simulation Systems to Embrace Changes of Requirements Using Event Control Models

Se Jung Kwon, Bonggu Kang, Changbeom Choi, and Tag Gon Kim, *Senior Member, IEEE*

***Abstract*—During the development or deployment of discrete event simulation systems, many sudden changes in requirements emerge. To embrace the changes rapidly and reduce development costs, attaching reusable, and black-boxed components to existing systems has been regarded as one of the most effective approaches because modifications at the code level are generally costly and risky. However, since this approach requires components that are an exact fit, it may not be easy to avoid the modifications at the source code level actually. Moreover, the required components may not exist. Hence, this paper applies black-box extensibility to simulation systems in order to avoid the modifications. This paper proposes adaptive discrete event simulation systems using event control models for extending and modifying semantics through event-based simulation interface. With the proposed work, the simulation events are modulated, deleted, and generated by the event-oriented control functions in the event control model to embrace the changes of requirements. It can substitute for the modifications at the code level and extend the existing behavioral semantics. As a result, the proposed work provides a new alternative step in the development process with reusable components to avoid modifications at the source code level. The new step will lead to rapid adaptations of existing simulation systems. To support the effectiveness of this approach, this paper will describe applicable examples based on our empirical studies.**

***Index Terms*—Black-box extensibility, discrete event simulation systems (DESSs), discrete event systems specification (DEVS) formalism, event control model, event-based simulation control, modeling, simulation.**

I. Introduction

A DISCRETE event simulation system (DESS) is a widely used abstraction view to analyze the behaviors and performance of real systems in various domains, such as military war games, industrial applications, social decisions, and more [1]–[5]. For the efficient design and maintenance of simulation systems, reusable components or systems have been emphasized. Reusable components or systems at diverse levels have helped a lot to extend or construct the semantics of simulation systems because reuse leads to a reduction in development costs, time, and efforts and also improves other factors. With an emphasis on reusability, the target systems are reused without modifications in internal specifications or codes, or the required semantics can be substituted by reusable components from outside.

To embrace the changes of requirements, developers have tried to choose proper extension approaches based on the reuse of target systems or reusable components. If a certain approach is completely applicable, developers will be able to satisfy the changes. Otherwise, they should approach the code level and regard the existing systems as a white-box. This may cause many shortcomings, i.e., increasing development cost, being confused by fragmented source codes, or increasing possibilities that side effects will occur [6]. Hence, researchers have tried to regard divided and modular components as black-box components when possible.

However, it is often difficult to reuse black-box components or to regard the existing system as a black-box since the available components may be a little different than the required components. This means that developers often have to choose between reusing a component that is an exact fit and rewriting (redoing) the codes. The reuse-redo dilemma [7] is a crucial problem in the development process because the reusable components or target systems should be modified at the code level. Therefore, it requires that the target systems or reusable components be made adaptive.

To avoid the modification of original codes and make DESSs adaptive, this paper adopts black-box extensibility at the antipode of white-box modification. Since the black-boxed component is unchangeable and invisible, it is easier to use and requires less knowledge about internal details [6]. The changeable software with black-box extensibility has extensions and interfaces inside the black-box. Similarly, this paper regards the existing components of the target systems as black-box components, focuses the interactions, i.e., the simulation messages, among the models and the simulation engine as the interfaces, and tries to inject extension semantics, event control models (ECMs), into the target systems. In Fig. 1, the extension corresponds to the control of events by the external ECM, and the interface corresponds to the application programming interface (APIs) of event-based simulation, which are able to access/modify the events. When the components of simulation systems should be extended or modified, developers can

Manuscript received February 27, 2017; revised May 16, 2017; accepted August 18, 2017. This work was supported by the Defense Acquisition Program Administration and Agency for Defense Development, South Korea under Contract UD160075BD. This paper was recommended by Associate Editor W.-K. V. Chan. *(Corresponding author: Se Jung Kwon.)*

S. J. Kwon, B. Kang, and T. G. Kim are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: sejungkwon@kaist.ac.kr).

C. Choi is with the Global Entrepreneurship and ICT Convergence, Handong Global University, Pohang 37554, South Korea.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2747604

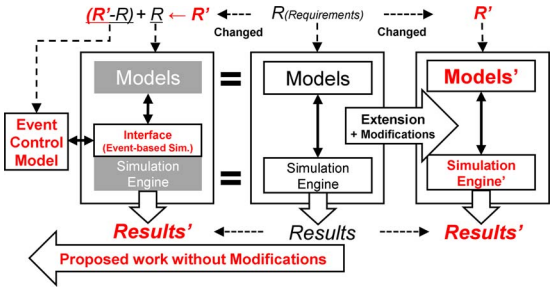


Fig. 1. Adaptive DESS with black-box extensibility.

design ECMs for the target events to access and modify the behaviors of models.

ECMs can be used as a new alternative to support the existing approaches. If one of the extension approaches becomes usable and we can satisfy the requirements by applying our proposed approach, the modification approach will become avoidable and the development costs will be reduced. Consequently, the existing DESSs will become adaptive because the existing systems and components can embrace the change of requirements rapidly with less effort.

This paper focuses on structural and formal DESSs, which consist of structural models for describing the structure of systems and behavioral models for describing the basic behaviors of the systems. We mainly concentrated on discrete event systems specification (DEVS) formalism [8] and its variations, such as in [9]–[13], because it is one of the most frequently used system specifications to model discrete event systems. Moreover, from our perspective, the DEVS formalism is an advanced group of discrete event simulations for reusability and extensibility.

The remainder of this paper is structured as follows. Section II reviews the background and some related works. Section III explains the proposed work, an ECM, and Section IV describes the advantages and constraints in detail. Section V presents the applications according to the various levels of reusability. Section VI illustrates case studies to show the efficiency of this paper, and Section VII concludes this paper.

II. Background and Related Work

In the past, DESSs were designed as programs with procedure-oriented concepts. As many studies have been developed to achieve reusability and various derived concepts, such as the separation of concerns (SoCs) [14], object-oriented (OO) designs, and formalism-based approaches, simulation systems have been divided into modular components at various level as shown in Fig. 2. Fig. 2 indicates each related ability as a subcategory of reusability and which parts are targeted as reusable components.

The simulation systems, which are implemented as simulation software, i.e., simulators, consist of a simulation engine with execution algorithms and simulation models that contain the semantics of system behaviors. The models are divided into structural models and behavioral models. The behavioral models describe the basic behaviors of systems, and the structural models are structured by behavioral models or smaller structural models for easier maintenance and reuses. The models have been mostly implemented as OO models because OO concepts increase the reusability.

This paper based on these modeling concepts focusing on the formal and structural systems. The rest of this section will describe each level with various related works.

1) Level 5 (Interoperability): Regarding the aspect of interoperability, which aims to reuse existing simulators, distributed environments have been created using high level architecture (HLA) standard protocol [15], [16], distributed interactive simulation [17], or common object request broker architecture [18], [19]. By applying the interoperable standard protocol, heterogeneous systems can be interoperated for advanced simulation objectives such as precise simulation [20], physically distributed simulation, hybrid simulation [11], and the integration of existing systems.

According to the purpose of the standard middlewares, modification of an existing simulator will rarely occur. However, the simulator should be modified according to each protocol to make the simulation systems middleware-compliant.

2) Level 4 (Reconfigurability): Developers can enable simulation systems to embed reconfigurable semantics. The basic approach involves instrumenting an experimental frame (EF) into the simulation models [8]. This can adjust the fixed parameters of a target model or control the simulation APIs out of the target model. Regarding the extensions of the EF, there have been many advanced studies. Mittal and Zeigler [21] applied the EF to real-time simulation. Model/simulator/view/control patterns [22] provided the theoretical basis of the simulation control, especially for the dynamic variable structure [10], [23], [24]. Using these concepts, output messages can be dynamically routed to various passes according to the states or variables. Adding the extended aspect of automated simulation, Mittal [25] extended the EF and MVSC patterns for the process of architecture frameworks.

The limitation of level 4 is that the reconfigurable semantics should be designed in the development process. The simulation engine should be able to accommodate certain reconfigurable semantics, and the reconfigurable semantics should be embedded in the simulation model. When the changes of requirement occur, the uncovered requirements will surely lead to a modification process.

3) Level 3 (Black-Box Reusability): There have been many approaches that use repositories of existing submodels and structural semantics to construct the target systems. The modeling and simulation (M&S) methodology for the reuse of the existing models was formalized as model base (MB) management such as system entity structure (SES)/MB [26]–[28], which consists of the SES as an architecture description and the MB as the repository of partial models.

Some studies have adopted service-oriented architecture [29] or restful Web service [30]. The simulation resources are distributed on the Web, and users configure the required simulation from existing models that are distributed. Additionally, some studies have tried to provide an efficient management approach by executing an architecture framework [25], [31]. These studies can also be approximated to MB management in view of the inherent M&S methodology.

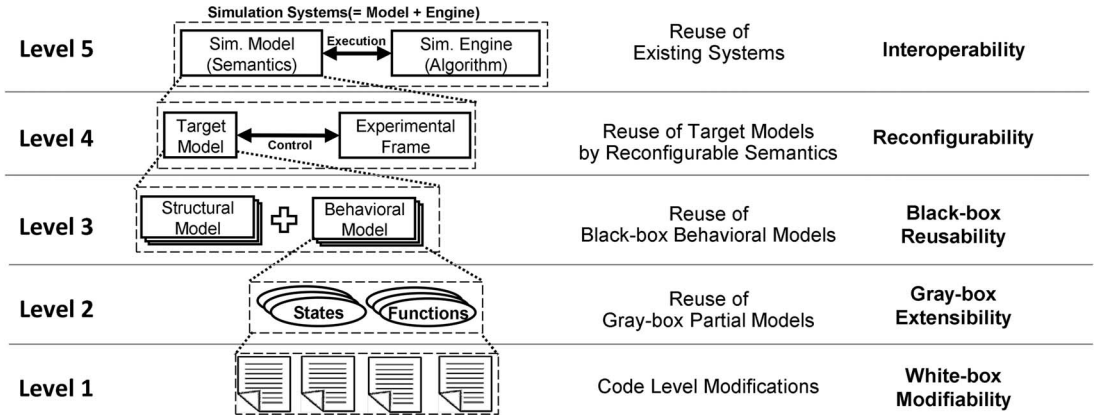


Fig. 2. Level of reusability according to the composition hierarchy.

These studies definitely reduce development costs by reusing existing resources. Nonetheless, the problem posed in this paper still exists because these studies have not been able to guarantee the existence of proper M&S resources that are satisfying the target objectives. Besides, even if there exists a proper resource, it may occasionally have a slightly different interface, resolution, or abstract levels.

4) Level 2 (Gray-Box Extensibility): OO concepts for discrete event models [32], [33] have been proposed for the reuse of behavioral models. There has been a study showing that the simulation engine enables users to exchange behavior functions as dynamic library types at run time [34]. They correspond to gray-box extensions, which means that the target source codes are visible but not directly extensible [6]. In other words, a portion of the codes are protected; nevertheless, the rest of the codes should be modified.

In this paper, the smallest unit of the "box" is a behavioral model because this paper was written in the context of M&S engineering. As this level needs to modify the source codes under the behavioral models, it corresponds to our attempt to avoid target levels. Under this level, there exist various techniques for reuse at the source code level. This, however, does not relate to our concerns, and we try to avoid all the approaches that involve breaking black-boxes.

III. Event Control Model

In order to make the black-box DESSs more adaptive, this paper propose the ECM as the extensible semantics that covers the changes of requirements. First, this section classifies the simulation events of DEVS formalism into two types of events. Based on the classified events, this section mainly provides the specification of the ECM, simple algorithms, and various extensible semantics.

A. Events and DEVS Systems [8], [35]

DEVS formalism consists of coupled models (CMs) as structural models and atomic models (AMs) as behavioral models [8]. CMs consist of submodels and coupling relations, and AMs consist of three sets (X, Y, and S) and four functions: 1) external transition function (δ_{ext}); 2) internal transition function (δ_{int}); 3) output function (λ); and 4) time advance function (ta).

The original execution algorithm of DEVS models [8] progresses hierarchically through four simulation messages, (x, t), (y, t), $(*, t)$, and (done, t_N). There are two simulation processes according to the state transition: 1) the external transition, by the input from the other components and 2) the internal transition, when the system changes the state by itself at the scheduled time. When the internal state transition occurs, λ, δ_{int}, and ta functions should be called. Then, the output of λ functions is translated to output message (y, t), and the message will be transmitted to the destination model as the input message (x, t). From the output of ta function, the model generates a (done, t_N) message and notices the next scheduled time for the simulation engine. When the δ_{ext} and ta functions should be called according to the (x, t) message, the (done, t_N) is also generated.

Although the abstracted algorithm of DEVS formalism has been implemented in various simulation engines [36]–[39], the kernels of the simulation processes are summarized as two parts: 1) deciding on a minimum time by the (done, t_N) and 2) transmitting messages among the AMs from (y, t) to (x, t). Hence, the simulation algorithm can be abstracted to management of the abstraction unit in DESSs, i.e., the *event*, which is mapped to the transitions of discretized states. The events imply the information that is transmitted out of each behavioral model, i.e., variables related to output messages or scheduled times. The events are managed in a chronologically sorted event list, and the top event on the list is executed by a simple simulation engine with an event-based simulation algorithm.

Definition 1: In a set of events, E, an event is defined as $E_k = <tg, src, t_N, v>$, where

tg target of the event;
src source of the event;
t_N execution time of the event;
v set of variables.

There are two types of events in DEVS formalism: 1) data events (E^d) and 2) time events (E^t). The data events are mapped on each type of output message (Y) generated from the λ function. In other words, the data events are mapped to (x, t) and (y, t) messages. When the messages are transmitted from

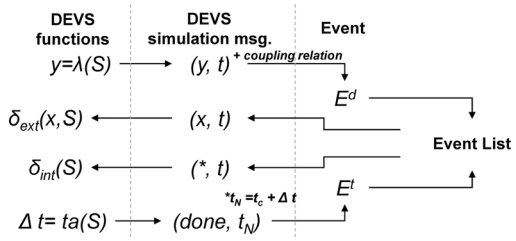


Fig. 3. DEVS simulation messages and events.

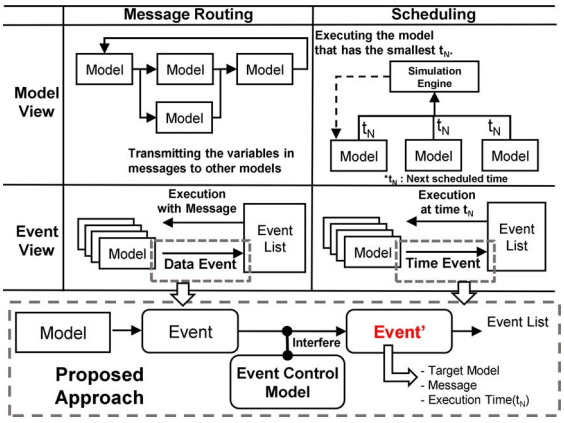


Fig. 4. Discrete event simulation and proposed ECM.

source models to target models through a simulation engine with variables, the *tg* is mapped on a input port (X in AMs) in its target model, the *src* is mapped on a output port (Y in AMs) in its source model, and the variables are stored in the set, v. The t_N value of the data events is same as the current simulation time because the messages are passed to the destination model immediately in the DEVS simulation.

The time event is generated by *ta* functions in each AM and transmitted to the simulation engine in order to execute itself after an advanced time. In other words, the time events are mapped to $(*, t)$ and (done, t_N) messages from the *ta* function. Hence, both the *tg* and *src* must point themselves out, and the set v must be empty. Indeed, the time t_N indicates the next scheduled time of the model. Fig. 3 describes the relation between DEVS simulation and the events.

B. Event Control Model for Extending Semantics

Since the DEVS simulation can be regarded as the sequences of events in the event-based simulation, the proposed work tries to access the event list in order to modify the information extracted from the AMs. As a result, the simulation behaviors and results can be changed by the ECM: changing time values in events can reorder the event sequence, and changing internal data in the events can modify the influence of the events on other models. As depicted in Fig. 4, the ECM interferes with the variables of the events, which are generated from the models and passed into the engine.

The ECM was published in [40] to enhance the fast prototyping methods in the development process. For this paper, we polished the definition and theory and extended the usages and application scopes.

Definition 2: The ECM is defined as ECM $=$ $<\{E_k\}, S_{\text{ecm}}, \{f_i\}, \text{CR}, \text{SELECT}>$, where

- $\{E_k\}$ set of target events in a simulation system;
- S_{ecm} set of states for the ECM;
- $\{f_i\}$ set of event-oriented control functions. : $(E_k \cup \varnothing) \times S_{\text{ecm}} \times t \rightarrow (\{E_k\} \cup \varnothing) \times S_{\text{ecm}} - t$: A current time when the f_i is executed;
- CR set of relations between $\{E_k\}$ and $\{f_i\}$. : CR $\subset$ $\{E_k\} \times \{f_i\}$;
- SELECT tie-breaking selection function. : $2^{\{f_i\}} - \varnothing \rightarrow f_i$.

From the target events in the DESS, developers can describe an ECM for the extra semantics due to changes of requirements. The key point of the ECM is the event-oriented control function, f_i, which generates a modified event or other related events from the generated event of a model, occasionally eliminates the event when a certain condition is met, or generates source-irrelevant events at a certain time with no input event. The input parameter t corresponds to the current time when the f_i is executed. Specially, the t will be a more meaningful parameter for the f_i, which generates source-irrelevant events.

In cases when the control functions need to store information, the state set S_{ecm} is included in the specifications. The SELECT function exists to resolve the priority problem because two or more functions, mapped on the same event, are rarely in conflict. The relation CR exists for mapping between the control functions and the events, i.e., mapped output messages or mapped models.

C. Event-Based Simulation Algorithm for ECM

The event-based simulation algorithm is composed of two parts: 1) a scheduling function for inserting the events and 2) a main routine for executing the events in the event list in order [26], [41]. The original event-oriented models directly call the scheduling function to insert an event to the event list, which contains the events in chronological order. The events are executed by the main simulation routine one by one.

In order to execute the DEVS models by this algorithm, the flattening process should be performed before executing the models. Since the event-based algorithm has no concern about the hierarchical structure, the structural models should be disintegrated, and divided into the AMs and mapped events. The data events are generated by analyzing the coupling relations and message ports. Each pair from a source port to a destination port is transformed to a data event. The time events are mapped to each AM. In the simulation time, an additional mediation algorithm resolves the differences in interfaces between the event-based simulation and DEVS simulation. The simulation messages of DEVS simulation are transformed to corresponded events. The (done, t_N) messages are mapped to time events and the (y, x) messages are mapped to data events. When the events are executed, the λ, δ_{int} and *ta* functions are executed in order according to the time events and the δ_{ext} and *ta* functions are executed according to the data events. By inserting the events, which are transformed from results of the functions, to the event list, the simulation

Algorithm 1 Event-Based Simulation With ECM

```
1: ECM : Event Control Model
2: T_global : Current simulation time
3: EventList : List of sorted event (time, mapped_func)
4: newEv : Set of new events generated by generation type functions
5: procedure SIMULATION_RUN                    ▷ Simulation Main Routine
6:     sort the ECM.CR according to the ECM.SELECT
7:     while EventList is not empty do
8:         first ← the top of EventList
9:         if ECM.nextTN < first.time then ▷ nextTN is a variable in S_ecm
10:            (newEv, ECM.S_ecm) ← ECM.f_k(null, ECM.S_ecm, ECM.nextTN)   ▷ f_k is the corresponding generation function
11:            Recalculate the nextTN
12:            if newEv and is not null then
13:                insert the events in newEv to EventList
14:                continue
15:            end if
16:        end if
17:        delete the top of EventList
18:        T_global ← first.time
19:        execute first.mapped_func
20:    end while
21: end procedure
22: AEv : Set of Additional Events generated by f_k
23: procedure SCHEDULE_EVENT((time, mapped_func))    ▷ API for event scheduling
24:    create an event Ev with the pair(time, mapped_func)
25:    if there are mapped control functions in ECM then
26:        for each mapped control function f_k in ECM.CR do
27:            (Ev, AEv, ECM.S_ecm) ← ECM.f_k(Ev, ECM.S_ecm, current time)
28:            insert the events in AEv to EventList
29:        end for
30:    end if
31:    if Ev is not null then
32:        insert Ev to EventList
33:    end if
34: end procedure
```

goes on. We omitted the detailed algorithm, but you can refer to our previous paper [35].

Based on this event-based algorithm, we added two parts of codes, lines 9–17 in the main simulation routine and lines 26–32 in the scheduling function, for the executing the ECMs.

The algorithm added to *Schedule_Event* is quite simple. Before the generated events are inserted into the event list, the simulation engine calls the proper functions of the ECM. In the case that multiple control functions are mapped to one event, the ECM functions should be basically sorted according to the SELECT function. As the control functions may generate more events except for the input events, the algorithm should also insert the additional events, *AEv*, to the event list.

The *Simulation_Run* function is extended for the generation of new source-irrelevant events. The *nextTN* state in S_{ecm} indicates the next execution time of the generation functions in *ECM* and the corresponding control function. Each generation function has an execution interval to be executed by the simulation engine without source events. The next execution time is decided among each generation function. If some generation function has same execution time, the priority will be decided by the SELECT function in *ECM*. If the *nextTN* is faster than the execution time of the *first* event when comparing the two values on line 9, the simulation engine should execute the control function of generation type. When it generates new events, the events are inserted into the event list, the rest steps of the while loop are skipped due to the *continue* statement on line 14. As a result, the original top event, *first*, is pushed back and it will be changed to one of these inserted events.

Even if a simulation engine does not use the event-based simulation explicitly, the simulation engines surely embed functionalities that correspond to an event list for scheduling information and messages. If any other simulation engine provides the APIs for accessing the information or the event sequences, the basic assumption for our proposed work can be satisfied. On the other hand, this paper also suggests the instrumentation of APIs into the simulation engine to access information at the I/O levels.

D. Extensible Semantics Through ECM

This section provides the detailed explanations about how the ECM can be used. Fig. 5 lists the five cases of extensible semantics through ECM. For better understanding, the figure provides the functionalities and the structural view of each case. We do not address this section in the view of state transitions. It is true that the ECM can alter the results of simulation models as if the state spaces or state transition functions were modified. However, because the state transition view is hiding under the assumptions of this paper, black-boxed behavioral models, describing the behavior of ECM in the state transition view is not our concern. We just concentrate on the opened information out of the behavioral models.

To apply the ECM to message flows of formal structural models, users have to consider I/O specifications and design the control function for the messages. In brief, the messages can be regarded as the events. If the messages should be modified, then the modulation function will be suitable. If the messages should be delayed or vanished, then users can design the function, which alters the time value of the data events.

The function f_i in case (a) changes the variables v in data event with or without the S_{ecm} before the event is passed through the simulation engine. In the DEVS simulation, the output messages y from the λ function are translated to the data events. When the requirement that an output y of an AM should be changed to y' occurs, the function in case (a) can substitute with the modifications of the λ function. This case can be used as the extension of the generator in EF. For example, this modulation is used for the parameter tuning as the most basic application. By extending the generator in EF, the variables between models can be parameterized through these event-oriented control functions.

The control target of this case (b) is the time value, t_N, generated from the scheduling functions, i.e., *ta* functions in DEVS formalism. Though the produced t_N is same as the current simulation time, the ECM can handle the value to create a time delay in the execution of events for advanced control. Adding the delayed time to t_N, users can cause the output

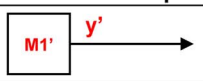
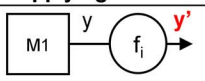
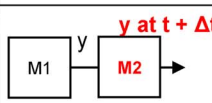
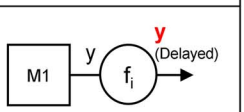
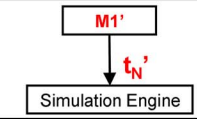
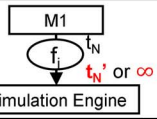
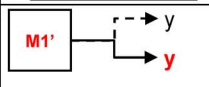
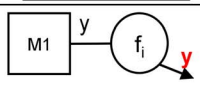
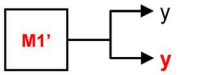
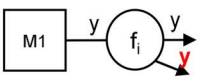
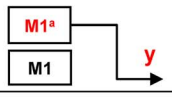
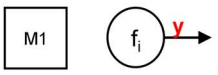
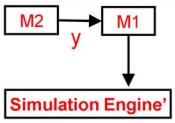
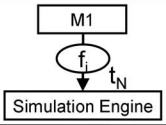
Control Target	Role of control function f_i	Applicable Functionality	Structural View: Modification in past	Structural View: Applying with ECM
(a) Output Variables <*tg*, src, t_N, v>	Changing the y to y' Changing the y to y' with additional states of ECM	- Parameter Adjustment as Extended Experimental Frame - Fault injection to messages		
(b) Execution Time of Events <tg, src, t_N, v>	- E^d Substituting new model out of the M_1. The data will be transmitted after the delay time	- Substituting abstracted model for the physical phenomena - Deletion of message by certain conditions		
	- E^t Modulation of execution times Make the value of t_N infinity	- Modulate the time intervals of model execution - Make the decision time stochastics - Stop the execution of models		
(c) Change of Destination <*tg*, src, t_N, v>	Changing the destination of y	- Substituting dynamic structure - Change of message passes by certain condition		
(d) Generation of Other Data Event <*tg*, src, t_N, v>	Generation of additional output y', (After transmitting y, the y' will be transmitted to other destinations)	- Broadcasting a message - Logging		
(e) Generation of Unexpected Events E_k	- E^d Generation of unexpected outputs with no relation with M1	- Source-irrelevant messages - Generation of rare events - Event injection by human-in-the-loop simulation		
	- E^t Execute the stopped model by external effects	- Resume the execution of the model		

Fig. 5. Extensible semantics through ECM.

to arrive in the destination model late. If the t_N value of an event becomes infinity, it eliminates an event and prohibits the event from arriving at its destination model. By the control function of this case, a data message can vanish according to a certain probability function, and the destination model will never know. The control functions of this case abstract the delay effects of messages from the models, e.g., the communicational delay. Since the messages in the DEVS simulation are delivered immediately, developers should have designed additional models for message delay. With the ECMs, they can substitute the new development of the models using the ECMs as depicted in the structural view of Fig. 5.

Case (c) modulates the *tg* of the date event. Since the *tg* is generated by analyzing the coupling relations in structural models (the CMs in DEVS), it corresponds to changing the destination of the target message in the same way to the dynamic variable structures. It can substitute the changes of the structural models or the extensions of the simulation engine for reconfigurable semantics.

Case (d) produces an effect that creates the additional outputs triggered by the *y*. The extra events registered in the event list can be other messages or the same output to other destination models. It can substitute with the modifications of λ functions in DEVS formalism. This case is also related to the data events toward the simulation engine for calling application programming interface (API), telling the simulation to terminate/stop/resume, logging, making assertions, and so on. The event-oriented control functions can monitor the variables generated from models for various purposes and can generate messages to the simulation engine. In this case, the *tg* of the E^d is the simulation engine and the *v* stores the arguments about the APIs.

Case (e) is the generation type of the control functions. The control functions of this case generate the external events or source-irrelevant events with a distinct scheduling algorithm or an external input. It seems that there is an additional model, $M1^a$, which has a certain state generating the output *y*.

In the case of time events, the control function in the case (b) changes the value of t_N to other values. It means the modifications of *ta* functions in the DEVS formalism. Before a mapped time event from time advance function in DEVS formalism is scheduled into the simulation engine, the control function changes the time value in the time events. As the modulated execution time is scheduled into the simulation engine, resultingly, the time advance values of each behavioral model can be modified. Furthermore, the control function can shut down or execute a certain model from the deletion or generation of the time events without modifying the submodel inside. The cases will be utilized to extend the time management policy in simulation engines.

However, applying the ECM to each time event is very limited. The value of the scheduled time must be related to each internal status of the models that is contrary to the messages. To control the time values, the event-oriented functions should know in which state the time scheduling occurs. Hence, the ECM cannot know the meaning of the time advance values without the internal status, and this leads to the limitation of the time event controls.

These cases can be combined for more complex behaviors with the interference of only an ECM. For example, f_i can correspond to both cases of (a) and (b) by changing the t_N and v simultaneously, or f_i of (b) and (d) can generate additional events to other destinations and send the event to the original destination with a time delay.

IV. Advantages and Constraints

This section describes the advantages and constraints of the ECM. The ECM is not always applicable to the development process or analysis process of DESSs. We expect this section to help users make decisions.

A. Advantages

The ECM enhances reusability and extensibility. The ECM makes effect to modify the existing models without modifying the models actually. It also performs the role of a glue among models or simulators without changing their existing components, as Section V will demonstrate. It prevents increasing development costs due to the possibilities of side effects, fragmented codes, modification of white-box models, or development of new models.

This advantage is due to the transformation of concerns from formal and structural systems to event-based systems. The proposed work abandons the inner parts of AMs from the perspective of the SoCs. The proposed approach breaks the formal structures, moves the concern from the model view to the event view, and describes the only necessary semantics.

With the ECM, users can skip describing the scheduling functionality in DEVS formalism because the role of *ta* functions and scheduling algorithm in DEVS formalism can be embedded in the event scheduling. The event is inserted directly into the event list, and the execution order will be determined by sorting the event list according to the execution time. When the functions schedule the next events, the time advance values are described in the scheduling API directly instead of the extra descriptions. Even if a DEVS model performs a simple role, e.g., a transmitter, we used to describe all the four functions in DEVS formalism. On the contrary, an event-oriented function in ECM can cover the behaviors of a DEVS AM. As the substituting ECM is not scheduled on the event list and is called less than the DEVS functions, the execution time with ECM is also faster than that of a simulator with only DEVS models.

Moreover, the number of actual lines of source code could also be reduced quantitatively. When users modify or add a new submodel to a deployed simulation software, there are increased costs for not only developing the submodel but also modifying other related parts of the simulation systems, i.e., a model manager, coupling information in the structural models, a scenario generator in EF or message formats, and so on. With the additional semantics about the events, the related parts will be not modified, and the modification costs can be reduced. Instead, just a few lines about the CR are added.

In some cases, only a small difference may exist because the absolute quantity of needed semantics must be unchanged. Nonetheless, as the target events are related to more components in the simulation systems, the number of reduced lines must increase. Since the concern view is changed from couplings among models to events, multiple similar events can be targeted by a few event-oriented control functions. Even if the source and destination models/ports are different, several events are composed of the same or similar variables. Considering the existing systems, there can be many same classes or relationships of inheritance. As the structure of target systems becomes more complex, the benefits can be more effective. We will show how the ECM reduces the source codes of simulation systems effectively in the case study.

B. Constraints

This paper is based on the assumption that the extension target system is composed of agglomerated black-boxes and that the details under the box are not our concern. In the structure composed of the simulation engine and models, the controllable information is restricted to the opened variables from the models to the simulation engine. The states and operations are hidden from the view of the simulation engine. For this reason, the extensible semantics of the ECMs must not be equal to the ordinary discrete event semantics. The ECM has distinct constraints to extend the semantics of the behavioral models. In the view of the event-based simulation, there are three main constraints.

First, substituting for the modifications of the event generators, e.g., the λ function in DEVS, is restrictively permitted. The ECM can change from y to y' with or without additional states, S_{ecm}. If the y' need to be calculated with other states or variables in the black-box, the ECM cannot be applied due to violating the assumptions. And the modulated output y' should be interpretable to the destination model because we cannot modify the event interpreter, the δ_{ext} in DEVS formalism. In other words, it is impossible for them to add new events, i.e., X or Y in DEVS, which are not interpretable.

Second, the state space of behavioral models should not be actually expanded from a similar perspective. The ECM only makes effects to expand the state space out the behavioral models, but it does not modify the model inside actually. Since the states of the existing models are hidden to ECMs, the modified semantics cannot be related to the internal states. Instead, there are additional states for ECM, S_{ecm}.

Third, the behavioral models should have no relation with the exported events after the events are generated. In other words, the models should not memorize the exported output. The ECM modifies and deletes the events without considering their source. The concerns of the ECM are generated events, not the generating events in the source models. The relation between the states in the source model and the generated events may cause unexpected results. For example, assume that a certain model stores the exported output and gathers the feedback of the output to compare the feedbacks and the exported output. Since an ECM modulates the exported output, they cannot guarantee the correctness of the models. Likewise, they should also be careful of the modulation of time events

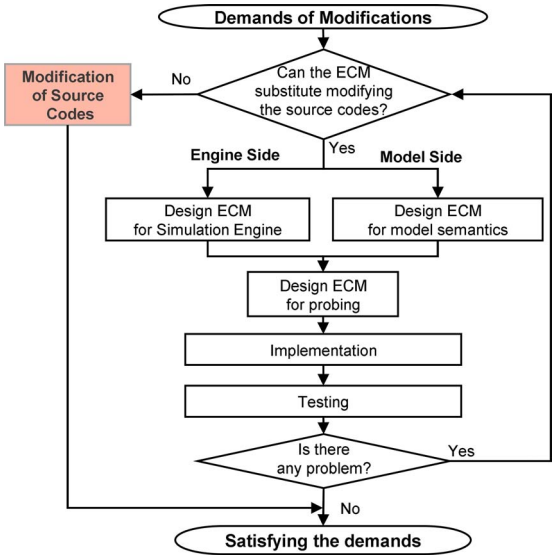


Fig. 6. Design process of ECMs before modifications of the source codes.

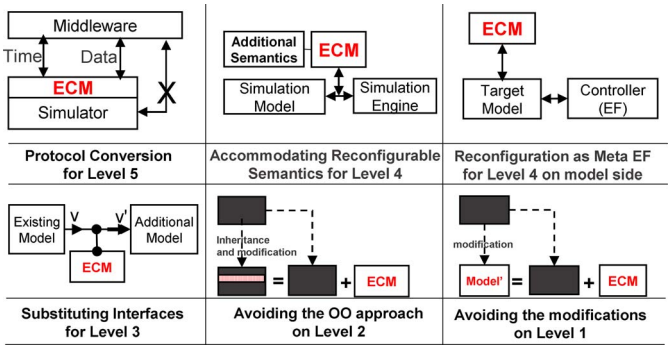


Fig. 7. Roles of ECM for each level.

because delayed or deleted time events may bring about a deadlock of the destination model.

Conversely, the key to avoiding this third constraint is enhancing the modularity. The DESSs on which this paper focused have been developed in that direction. Since high modularity means less relations among subcomponents and variables, developers do not need to be concerned with this constraint. It is easier for developers to estimate the resulting behaviors of the models by only using the I/O interface.

V. Applying ECM to the Various Levels of Reusability

The ECM can be applied to the existing process to embrace changes of the requirements. As argued in Section II, when developers apply the approaches at each level to their target systems, they often need modifications at the code level, such as making the systems middleware-compliant, making the systems capable of accommodating the reconfigurable semantics, providing the interfaces in such a way that resolves the differences, and the so on. If there is no reusable component, they finally may be compelled to break the black-boxes.

Before modifying the systems or components at the code level, the proposed work provides alternative steps using the ECM, as depicted in Fig. 6. If users confirm that the ECM is effective, they will design the ECM in accordance with the attempted approaches.

To design the ECMs to structural models, user should extract the events. Extracting can be performed automatically or manually according to the type of simulation engines. Analyzing the coupling relations and message ports, the data events are identified. And each behavioral models are mapped to the time events.

The utilizations of ECM at each level are different and can be classified into two types: 1) the simulation engine and 2) the simulation model since the simulation events are simulation units and are also mapped on the model functions. On the side of the engine, they need to design the extension modules to satisfy the extension approaches. The extension process may still remain after the reuse of the simulation engine. On the contrary, the ECM on the side of the simulation models will itself be sufficient to satisfy the requirements.

The probing functions are also needed supplementally. Malfunctions may occur whenever developers use the control functions. When the users design the ECM and execute the existing systems, they must know whether the systems operate normally. Hence, users should set the assertion functions and should probe the variables for the normal behavior of the simulation model.

The rest of this section will describe how the ECM is applied to each level in the context of Fig. 2. Additionally, Fig. 7 is provided as a supplement for understanding.

1) Protocol Convertors for Level 5: To be reused at the level of simulation systems, the existing simulator must be interoperable. This means that it should have interface modules for the middleware. If the modifications were not planned at the development of the simulation systems, the reusable components should be modified.

For the interoperable simulation engine, the ECM conducts the protocol converters to make the systems middleware-compliant. The middleware-compliant simulator should be able to accommodate the protocols of the middleware. Although there should be many functionalities to make the systems compliant, the required modifications can be summarized as data transmitting and time management. Other functionalities are implemented using an external module such as KHLAAdaptor [42], which is connected to the ECM. Since the protocols can be regarded as the subset of discrete event simulation [16], the instrumented ECM can conduct the role of the protocol convertors.

a) Time management: The discrete event simulations are conducted in this order: 1) each model of a simulation system notifies the next execution times; 2) the simulation engine executes the top model, which has the minimum time; 3) to interoperate with other simulators, the middleware functions as the time manager of all the simulators; and 4) each simulator requests the next execution time (time advance request) and awaits the grant (time advance grant) from the middleware.

The ECM for the interoperation should assist the protocols. Even if only a few models are related to the interoperation, the control function for the time events must be attached to all the models in order to control the execution of all the models.

The control function gathers all the time events and generates only the granted events. Other ungranted events, whose t_N is over the granted time, are inserted with the infinity time stamp. When the time of an ungranted event is granted, the control function, as the generation function, inserts the time event again into the original t_N.

b) Data management: When messages or variables need to be transmitted to other simulators, a control function captures the outputs of the models and calls the proper APIs for the middleware. Conversely, when the transmitted input occurs from the other simulators, a control function for the input generates the data event at the time it occurs. The control functions for data transmission do not need to involve the time policy because the data transmission will occur in the granted time.

2) Reconfigurable Semantics for Level 4: To reconfigure the simulation model, there should also exist modifications for accommodating the reconfigurable semantics. The ECM can also conduct this role as the extension of the simulation engines. The ECMs for this case are concentrated in the data events because most of the reconfigurable semantics are about messages or routes of messages.

On the other side, the control functions of the ECM can be regarded as the extension of the reconfigurable semantics at level 4, such as the controller in EF or the dynamic structures. The related works about reconfigurability, as mentioned above, have assumed that the target of simulation control is the operation of the simulation systems, the input parameters, or the model structure. Though some studies have set up the variables at the level of model I/O, they did not consider modifying the dynamic behaviors of the simulation model over the dynamic structures. In addition, the reconfigurable behaviors of the listed studies should be considered and settled in the design and implementation of models. By contrast, the proposed work handles the behaviors of simulation models by dynamically interfering with the simulation events. The scopes of our control target are dramatically extended compared to those studies. In this case, the ECM alone conducts the reconfigurations of the model semantics, whereas the extension modules of the simulation engine need more descriptions for the extended semantics.

3) Interfaces Between Components for Level 3: As with the general software, the reusable components in the simulation models could not be instrumented completely because there may be differences in resolution, data types, and so on. Most of the differences are related to the data at the interface level. In addition, there may be errors in source codes typed by humans despite the design of the origin model not being wrong. When developers attach the existing components to extend the semantics, they may find errors. In this case, the models should be modified to achieve the requirements.

When developers face differences in the interfaces between the behavioral models at levels 3 and 5, the ECM can resolve the problems. Differences may exist in terms of measures, resolution, or fidelity between the data of each model. Furthermore, the reusable model has a few incorrect semantics, which can be resolved at the I/O level. Using the ECMs, developers can resolve the differences or inaccuracies easily while also avoiding modifications of the code level. Since the ECM focuses only on the target events, it will be more efficient than designing a new model or modifying the model inside to resolve the differences. In addition, there may be simulation environments with inaccessible behavioral models in the model repository, which is based on the MB management. In these cases, since the reusable components must be unmodifiable, the ECM will be more valuable.

4) Avoiding Modifications for Levels 2 and 1: If there is no reusable component to satisfy the requirements, developers should modify the model inside or develop new models, even if the degree of the changed behavior is small. Even if there are reusable codes at lower levels of reusability, i.e., OO inheritance, the fact remains that the rest of them should be modified. The ECM, even if it is limited, can be used in order to provide the extension/correction method of the model semantics. The ECMs make the target systems reusable without modifications by extending the controllable semantics, modulating the existing semantics, or substituting the development of the models.

VI. Case Study

This section presents actual examples in which the proposed ECM is applied. We assume the development process and changes of requirements using existing military simulation models, developed for the Korean Army in the past. The simulation models were developed for the DEVSim++ [42] and have been utilized for evaluating measures of performance.

This case study uses the E-DEVSim++ [35], which executes the DEVSim++ models in the C++ language with an event-based simulation algorithm. The E-DEVSim++ translates the formal DEVS models to the events and simulates them with an event list, and it provides external APIs to describe the relation, CR, as

*void AddRelation(Message *m,void (*f)(Event *e)*
*void AddRelation(Model *m,String *p,void (*f)(Event *e))*
*void AddRelation(Model *m,void (*f)(Event *e)).*

There are generally three methods. Users can describe the control functions (*void *f*) for each message class (subclasses of the *message* class), models (subclasses of the model class), or the port name (*p*) of models. By using the first type of *AddRelation* function, the control function will be operated on all the same output messages. The second type is the general and basic form, and the third type is suggested for the control function of the time events (or scheduling).

The control function *f* is described by users. The simulation engine calls this function with the pointer parameter, *event*, which is the class for event management of E-DEVSim++. When the return value of *ta* function or λ function occurs, it will be translated to the *event* class and simulated. For example, assume that a message of the Msg class from "out" port in a source model to "in" port in a destination model exists and it should become transmitted after a delay time according to an embedded variable, "*flag*." The control function for this message in C++ is described in Fig. 8. The message, Msg, is simulated in the simulation engine as an event, E^d_{Msg} with the variable *flag*. The event-oriented function *MsgDelay* in *ECM_example* delays the event according

TABLE I
COMPARISONS BETWEEN PAST APPROACH AND THE PROPOSED ECM

	Changes of Requirements	Modifications in the Past	Control Functions in Proposed ECM
First Change	Precise simulation with comm. effects → Interoperation with OPNET → Change of behavioral models	E) Make the simulator HLA-compliant M) Modify all the unit CMs M) Implement an AM for transmissions **M: Model side / E: Engine Side*	f_{tm} for substituting time management - Modulation of E^t / Fig. 5(b) - Generation of E^t / Fig. 5(e) f_{se}/f_{re} for substituting data management - Change of Destination in E^d / Fig. 5(c) - Generation of E^d / Fig. 5(e)
	Simulation with stochastic decision times → Change of behavioral models	M) Modify the time functions in the C2 models	f_{prob} for semantics extensions - Modulation of E^t / Fig. 5(b)
Second Change	Faster iterative simulations → Meta model for transmissions	M) Design an AM for the meta model M) Modify all the unit CMs	f_{comm} for substituting new model - Delayed E^d / Fig. 5(b)
	Switching comm. simulator and meta model	E) Make the simulator reconfigurable	Switch ECM_1 and ECM_2 out of models


```
class ECM_example{
  double delayedTime;
public:
  ECM_example(double delayedTime){
    EngineInterface->AddRelation(mSource, "out", MsgDelay);
    // or EngingInterface->AddRelation(Msg, MsgDelay);
    this->delayedTime = delayedTime
  }
  void MsgDelay(Event *e){
    if( e->GetVariable(GetVariable("flag")) e->SetTime( delayedTime );
  }
};
```

Fig. 8. Example of the event-oriented control functions.

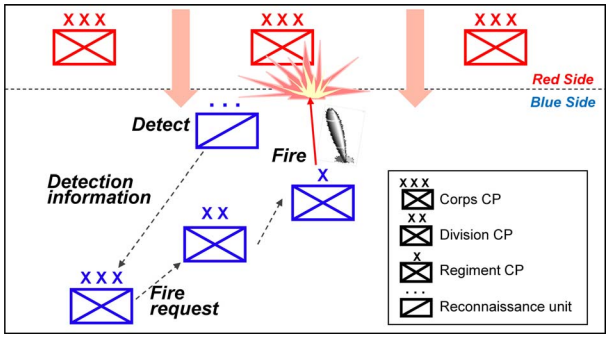


Fig. 9. Scenarios of the wargame model.

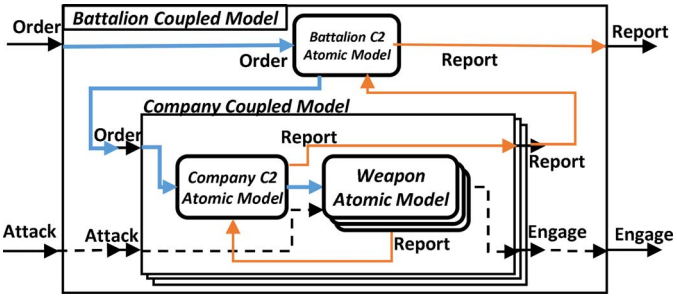


Fig. 10. Structure of the initially developed model.

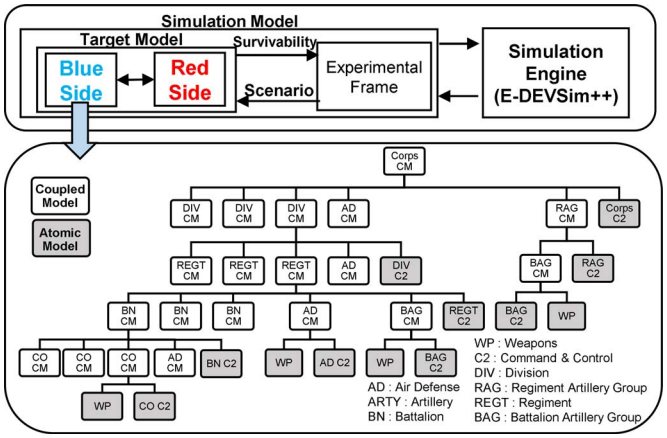


Fig. 11. Structure of battalion CM as an example.

to the variable *flag*. In this example, there are two descriptions of CR. First, the control function can be connected to the output ports of DEVS models, the 'out' port in mSource. Second, it can be connected to all the same message from the Msg class.

A. Corps-Scaled Defense Operations Model

The target model in this case study is the corps-scaled defense operations simulation model [43]. In the scenario in Fig. 9, the blue forces construct a defensive position and defend against the red forces' attack. The concerned output of the simulation was the survivability of the red forces.

The model was designed as a hierarchical structure, as in Fig. 10. Each CM is mapped on the military units and has the CMs of the subunits, their weapons models, and a command and control (C2) model. There are two messages about the C2 hierarchy, as depicted in the example of Fig. 11: order messages for the C2 of the higher units to the C2 of the lower units and report messages for vice versa.

When the simulation model was developed and deployed for the Korean military, the requirements were changed due to stakeholders' responses by observing the results of the simulation models. We assumed that the actual responses and redevelopment could mainly be divided into two changes, and Table I explains the changes of requirements and compares the past approach and the proposed approach with the ECM, which was newly developed for this case study. The actual C++ codes of this case studies are omitted. Instead, the changed model structure and specifications of the ECM are described.

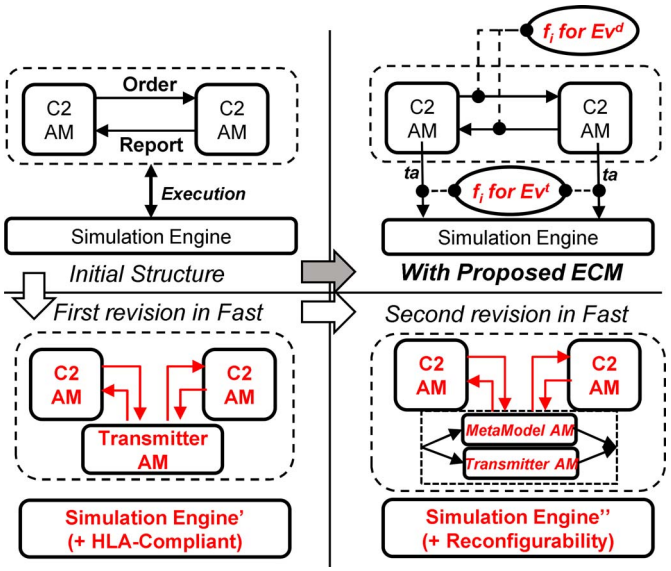


Fig. 12. Brief comparisons in the perspective of C2 models.

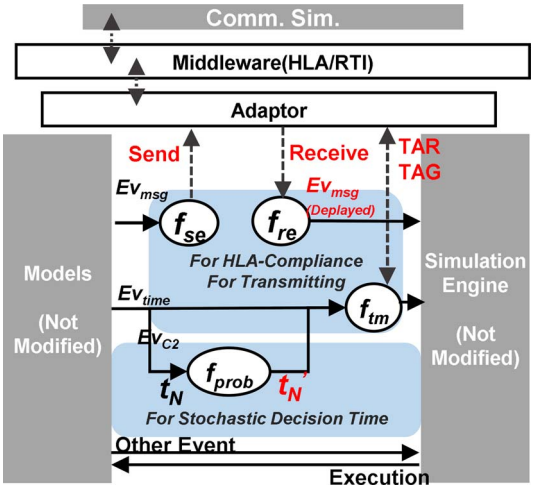


Fig. 13. First changed structure with the ECM_1.

B. First Change

The initial simulation system did not consider the communication effects through the signal equipments and emphasized the validation of the C2 structure and the evaluations of the defense operations. However, stakeholders required an extended simulator with a communication simulator for more precise simulation. Considering the hierarchical C2s, the communication traffic or the performance of the equipment causes delivery delays in the transmission of messages. This leads to an increase in the survivability of enemies. In addition, stakeholders wanted to change the fixed decision times to stochastic times, while also including probability functions.

Under these assumptions, the simulation systems should be extended to be able to interoperate with a specialized communication simulator, and the C2 models should be modified. The engine should be extended to be suitable for the standard protocols of HLA/RTI because the E-DEVSim++ is not HLA-complaint. On the model side, the messages of C2 models should be passed through the HLA/RTI according to the data management protocols. The time advance values of the C2 models should be changed to stochastic values.

For reusing the external simulator, the engine and model should be extended by the modifications at the code level. Fig. 12 compares the partial structures, which include only two C2 models and their interactions, between the past approach and the proposed approach. In the first revised structure, a new transmitter model for each CM is added with modified coupling schemes in order to intercept the messages and send them to the external simulator, and each C2 model is modified for the stochastic decision times. The simulation engine is also modified for interoperation with the external simulator. On the contrary, the existing model and simulation engine are not modified with the proposed ECM. The control functions for each event are attached to the initial systems and modulate the outputs of the AMs according to the changed requirements.

Fig. 13 shows the instrumented ECM_1 for the first change in detail. The figure draws the simulation events, which are generated by the models and delivered to the simulation engine, and the control functions to the events. The f_{se} and f_{re} are connected to the external adaptor and substitute for the transmitter model. The f_{se} deletes the data events E^d_{msg}, and the f_{re} generates the data events E^d_{msg}, which have the delayed times that have been changed from the current time (t_c), following certain times produced by the external adaptor. The f_{tm} is attached to all the models to conduct the time management policy, as mentioned in the previous section. Each event set is the subset of whole events in E^d or E^t. The sets are identified from the existing simulation models.

The f_{prob} modulates the time advance values of each C2 model. Attaching the f_{prob} to the C2 models, we can modify the time scheduling. Since the target events of f_{tm} and f_{prob} are overlapped, the SELECT function makes the f_{prob} be executed by priority.

$ECM_1 = \{\{E_k\}, S_{ecm}, \{f_{se}, f_{re}, f_{tm}, f_{prob}\}, \text{CR}, \text{SELECT}\}$.

1) $\{E_k\} = E^d_{msg} \cup E^t_{c2} \cup E^t_{model}$.
   - E^d_{msg} is a union set of data events mapped to the Order and Report messages.
   : $E^d_{msg,i} = <Dest\,C2, Src\,C2, t_c, \text{msg}>$.
   - E^t_{model} is a union set of all the time events.
   : $E_{model,j} = <\text{Model, Model}, t_N, \varnothing>$.
   - E^t_{c2} is a union set of all the time events of C2 models.
   : $E^t_{c2,k} = <C2\,\text{Model}, C2\,\text{Model}, t_N, \varnothing>$.
2) S_{ecm} = {List of sent message, List of ungranted models, Granted time, Prob. parameters}.
3) f_{se}: Sends the data in events to the comm. simulator and deletes the events.
4) f_{re}: Receives the data in events from the comm. simulator and generates the events.
5) f_{tm}: Manages the times of all the models and generates the time events that RTI permits.
6) f_{prob}: Changes the t_N to randomized values according to a probability function.
7) CR = $\{(E^d_{msg,1}, f_{se}), \ldots, (E^d_{msg,l}, f_{se}), (E^d_{msg,1}, f_{re}), \ldots, (E^d_{msg,l}, f_{re}), (E^t_{model,1}, f_{tm}), \ldots, (E^t_{model,m}, f_{tm}),$

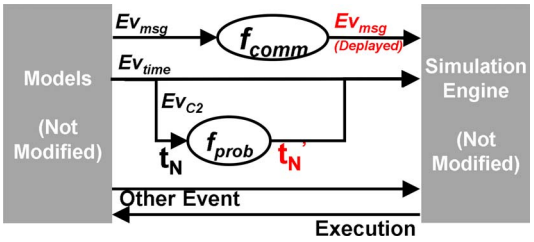


Fig. 14. Second changed structure with the ECM_2.

$(E^t_{C2,1}, f_{\text{prob}}), \ldots, (E^t_{C2,n}, f_{\text{prob}})\}$
$- n(E^d_{\text{msg}}) = l, n(E^t_{\text{model}}) = m, n(E^t_{C2}) = n.$

8) SELECT : $(f_{\text{tm}}, f_{\text{prob}}) \rightarrow f_{\text{prob}}$.

C. Second Change

The second change was caused by performance degradations because interoperation overheads caused inefficiency despite the results' accuracy. Therefore, a new meta model as a compromise approach between accuracy and performance was developed and added to satisfy the requirement [43]. The meta model has a substituting function, which is generated by a second-order response surface method applied to the actual results of the communication simulator. The meta model with a message queue stores the C2 messages and transmits each message after some time according to the approximated function. In addition, the engine was also modified for reconfigurable semantics between the interoperation and the meta model, as depicted in Fig. 12.

On the contrary, applying the ECM to the first revised systems, f_{comm} is added as the role of the meta model, and switching codes are added out of the simulation model, as shown in Fig. 14. The initial codes are still not modified during two of the changes. When the Order and Report messages occur, the f_{comm} calculates the communication effects from the approximated function and communication parameters, and it makes the data events be executed after the calculated time.

$ECM_2 = \{\{E_k\}, S_{\text{ecm}}, \{f_{\text{comm}}, f_{\text{prob}}\}, \text{CR}, \text{SELECT}\}$

1) $\{E_k\} = E^d_{\text{msg}} \cup E^t_{C2}$
   - E^d_{msg} is a union set of data events mapped to the Order and Report messages.
   : $E^d_{\text{msg},i} = <DestC2, SrcC2, t_c, \text{Msg}>$
   - E^t_{c2} is a union set of all the time events of C2 models.
   : $E^t_{c2,j} = <C2\,\text{Model}, C2\,\text{Model}, t_N, \varnothing>$.
2) $S_{\text{ecm}} = \{\text{Comm. Parameters}\}$.
3) f_{comm} changes the t_N of each event from zero to delay values according to the approximated function.
4) $\text{CR} = \{(E^d_{\text{msg},1}, f_{\text{comm}}), \ldots, (E^d_{\text{msg},l}, f_{\text{comm}}),$
$(E^t_{C2,1}, f_{\text{prob}}), \ldots, (E^t_{C2,n}, f_{\text{prob}})\}$.

D. Experimental Results

The original model with the communication effects was executed with several parameters. This paper only illustrates an experimental graph with two parameters: 1) the data-forwarding rate and 2) the processing delay time. These results were generated on the first revised simulator.

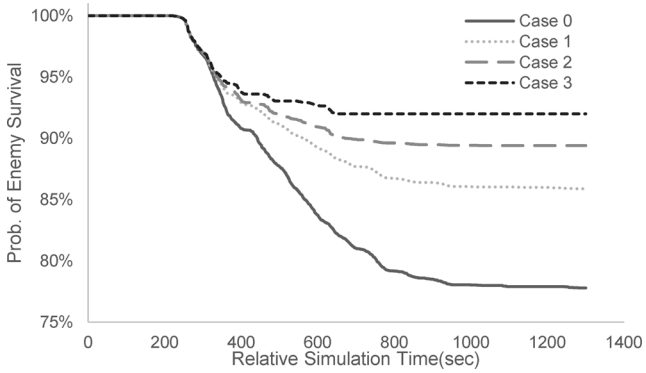


Fig. 15. Survivability of the enemy according to the four cases.

TABLE II
COMMUNICATION PARAMETERS FOR EXPERIMENTS

#	Data-forwarding rate	Processing delay time
0	Original model w/o comm. effects	
1	5k bits/sec	0 sec
2	Not considered	3 sec
3	5k bits/sec	3 sec

TABLE III
ACTUAL # OF ADDED/MODIFIED/DELETED LINES DURING REVISIONS

	Past approach	Proposed approach
First change		
Time Management	102 (2 classes)	51 in f_{tm}
Data Management	95 (2 classes)	46 in f_{se}/f_{re}
Transmitter AM	173 (1 class)	
Coupling Schemes	152 (8 classes)	4 in CR
Each C2 AM	45 (9 classes)	4 in f_{prob}
Second change		
Reconfigurability	10 (2 classes)	4 for switching
Coupling Schemes	283 (8 classes)	2 in CR
Meta Model AM	109 (2 classes)	12 in ECM_2
Summation	**995 (34 classes)**	**121 (4 classes)**

Fig. 15 shows the results of the experiments in four cases. The graph of case 0 was produced by the original simulation model without the communication simulator, and the rest of the results were from experiments using the ECM_1 according to the parameters in Table II.

These results show that the communication effects are important factors of war-game simulation. As the communication effects increase, the delay times of C2 messages increase, and more enemies can survive. Specially, the new insights, e.g., that the processing delay time is a more relevant factor than the data-forwarding rate, could be provided to the domain experts.

To verify that the ECMs operated correctly, we generated the simulation and interoperation logs from the past DEVSim++ and the E-DEVSim++ with the ECMs. By comparing time stamps and messages, we confirmed that the ECMs exhibited proper behaviors.

We can show that the substituting ECMs reduced the actual efforts during the processes. Table III compares the number of code lines in C++ language between the past and the proposed approach. The left side is the results applying modifications of white-box models to the two revision processes. On contrary, the right side is the results of the developed ECM. We counted

the actual lines of C++ codes including header files except for blank lines and lines with just only braces. Using the ECMs, the effort of the development process was reduced by about 88% in total, and we dealt with only two classes per change, a main class and each ECM class. The target system becomes a better application due to the hierarchical structures and various models that have similar events.

Moreover, there is also no uncountable effort due to modifying the existing source codes, such as increased side effects or fragments of the source codes. The proposed ECMs are related to only two classes, whereas 34 classes were modified or added in the past. This will definitely prevent unexpected increases in development costs.

VII. Conclusion

This paper proposes the ECM for the adaptive DESSs to embrace the changed requirements. The control functions in the ECM interfere with the events in the simulation systems. By succeeding the advantages of the black-box extensibility, the proposed work resolves the redo-reuse dilemma of reusability, provides disjointed semantics, reduces the fragments of the source codes, and decreases the probabilities of side effects. This paper also suggests the alternative step to avoid the modification of source codes. As a result, the ECM makes the existing DESSs more adaptive.

There may be troubled communications because it may be difficult for simulation system engineers to understand the domain-specific knowledge [44]. The misunderstanding of both the stakeholders and the developers may emerge late because they cannot forecast the validity of the requirements and designs before observing the executed results. For this reason, the adaptive system in the proposed work is even more valuable to the M&S engineering. It would be efficient to use the DEVS formalism or formal DES methodology in making the overall structure and then attach the event-based parts, i.e., ECM, to each black-box component of the initial structure when it is necessary to modify something already made.

It must not be an almighty approach for covering all the possible semantics of DESSs. Nonetheless, there are many cases in which the ECM can be utilized to resolve the problems during the development process, especially the revision process between the developers and the stakeholders. The case study, which was based on past developments for the Korean military, showed the improvement according to the proposed work in DEVS formalism. This paper obviously enhanced the reusability because the ECM prohibited users from modifying the initial simulation system or external components at the source code level during the two changing processes. However, the effectiveness of ECM may be lower when the developed simulation systems are expanded with many ECMs because a lot of partial behaviors to existing models definitely confuse developers. Hence, we recommend to avoid applying ECMs to incremental development processes.

The ECM can be applied to other simulators that provide the APIs to access simulation events or mapped information. According to the respective implementations, some types of ECMs, such as time modulation of the data events, may be impossible. If the ECM cannot be applied to a certain simulation engine, partially or totally, we would like to suggest adding the event-based APIs for the instrumentation of the ECMs. Although the ECM makes the effects on the behaviors of simulation models, the internal specifications of simulation models are not the concerns of the ECM. The ECM can only control the extracted events between simulation models and the engine. Hence, other DESSs can easily embrace the concepts of the proposed ECM.

In future work, the ECM will be applied to various steps in the life cycles of DESSs in detail. For examples, we are planning detailed applications, such as fault injections analysis in the experiment process, an enhanced rapid prototyping methodology by using ECMs, or advanced experiment designs using the extended EF. This paper does not intentionally describe the methodological contents, e.g., which development process, short-term and long-term, is more suitable to apply the ECM, because this paper focuses on the introduction of the ECM and it is hard to determine the methodological criteria in general. Instead, the future papers will focus the methodological life cycles of the ECM limited to a specified case studies.

References

[1] R. Sato and H. Praehofer, "A discrete event system model of business system-a systems theoretic foundation for information systems analysis. I," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 27, no. 1, pp. 1–10, Jan. 1997.

[2] A. I. El-Osery *et al.*, "V-Lab-a virtual laboratory for autonomous agents-SLA-based learning controllers," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 6, pp. 791–803, Dec. 2002.

[3] T. G. Kim and I.-C. Moon, "Special issue: Highlighting the military modeling and simulation work at the systems modeling and simulation laboratory at KAIST," *J. Defense Model. Simulat. Appl. Methodol. Technol.*, vol. 8, no. 3, pp. 127–128, 2011.

[4] J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. M. Al-Ahmari, "R-TNCES: A novel formalism for reconfigurable discrete event control systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 4, pp. 757–772, Jul. 2013.

[5] V. Augusto and X. Xie, "A modeling and simulation framework for health care systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 1, pp. 30–46, Jan. 2014.

[6] M. Zenger, "Programming language abstractions for extensible software components," Ph.D. dissertation, Dept. Comput. Sci., École Polytech. Fédérale de Lausanne, Lausanne, Switzerland, 2004.

[7] B. Meyer, *Object-Oriented Software Construction*. Upper Saddle River, NJ, USA: Prentice-Hall, 1997.

[8] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Amsterdam, The Netherlands: Academic Press, 2000.

[9] A. C.-H. Chow, "Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator," *Trans. Soc. Comput. Simulat.*, vol. 13, no. 2, pp. 55–67, 1996.

[10] F. J. Barros, "Modeling formalisms for dynamic structure systems," *ACM Trans. Model. Comput. Simulat.*, vol. 7, no. 4, pp. 501–515, 1997.

[11] S. J. Kwon, C. Sung, H. S. Song, and T. G. Kim, "Integrated hybrid systems modeling and simulation methodology based on HDEVS formalism," in *Proc. Summer Comput. Simulat. Conf.*, Toronto, ON, Canada, 2013, Art. no. 52.

[12] G. Zacharewicz, M. E.-A. Hamri, C. Frydman, and N. Giambiasi, "A generalized discrete event system (G-DEVS) flattened simulation structure: Application to high-level architecture (HLA) compliant simulation of workflow," *Simulation*, vol. 86, no. 3, pp. 181–197, 2010.

[13] J. W. Bae and I.-C. Moon, "LDEF formalism for agent-based model development," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 793–808, Jun. 2016.

[14] W. L. Hürsch and C. V. Lopes, "Separation of concerns," College Comput. Sci., Northeastern Univ., Boston, MA, USA, Tech. Rep. NU-CCS-95-03, Feb. 1995.

[15] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[16] Y. J. Kim, J. H. Kim, and T. G. Kim, "Heterogeneous simulation framework using DEVS bus," *Simulation*, vol. 79, no. 1, pp. 3–18, 2003.

[17] P. K. Davis, "Distributed interactive simulation in the evolution of DOD warfare modeling and simulation," *Proc. IEEE*, vol. 83, no. 8, pp. 1138–1155, Aug. 1995.

[18] R. Ben-Natan, *CORBA: A Guide to the Common Object Request Broker Architecture*. New York, NY, USA: McGraw-Hill, 1995.

[19] D. R. Hild, H. S. Sarjoughian, and B. P. Zeigler, "DEVS-DOC: A modeling and simulation environment enabling distributed codesign," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 32, no. 1, pp. 78–92, Jan. 2002.

[20] C. Sung and T. G. Kim, "Framework for simulation of hybrid systems: Interoperation of discrete event and continuous simulators using HLA/RTI," in *Proc. IEEE Workshop Principles Adv. Distrib. Simulat.*, Nice, France, 2011, pp. 1–8.

[21] S. Mittal and B. P. Zeigler, "Dynamic simulation control with queue visualization," in *Proc. Summer Comput. Simulat. Conf. (SCSC)*, vol. 5. Philadelphia, PA, USA, 2005.

[22] J. Nutaro and P. Hammonds, "Combining the model/view/control design pattern with the DEVS formalism to achieve rigor and reusability in distributed simulation," *J. Defense Model. Simulat. Appl. Methodol. Technol.*, vol. 1, no. 1, pp. 19–28, 2004.

[23] A. M. Uhrmacher, "A system theoretic approach to constructing test beds for multi-agent systems," in *Discrete Event Modeling and Simulation Technologies*. New York, NY, USA: Springer, 2001, pp. 315–339.

[24] J. W. Bae, S. Lee, J. H. Hong, and I.-C. Moon, "Simulation-based analyses of an evacuation from a metropolis during a bombardment," *Simulation*, vol. 90, no. 11, pp. 1244–1267, 2014.

[25] S. Mittal, "Extending DoDAF to allow integrated DEVS-based modeling and simulation," *J. Defense Model. Simulat. Appl. Methodol. Technol.*, vol. 3, no. 2, pp. 95–123, 2006.

[26] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*. London, U.K.: Academic Press, 1984.

[27] T. G. Kim, C. Lee, E. R. Christensen, and B. P. Zeigler, "System entity structuring and model base management," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 5, pp. 1013–1024, Sep./Oct. 1990.

[28] B. P. Zeigler, T. H. Cho, and J. W. Rozenblit, "A knowledge-based simulation environment for hierarchical flexible manufacturing," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 26, no. 1, pp. 81–90, Jan. 1996.

[29] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS unified process," *Simulation*, vol. 85, no. 7, pp. 419–450, 2009.

[30] K. Al-Zoubi and G. Wainer, "Rise: A general simulation interoperability middleware container," *J. Parallel Distrib. Comput.*, vol. 73, no. 5, pp. 580–594, 2013.

[31] B. P. Zeigler and S. Mittal, "Enhancing DoDAF with a DEVS-based system lifecycle development process," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 4. 2005, pp. 3244–3251.

[32] T. G. Kim and M. S. Ahn, "Reusable simulation models in an object-oriented framework," in *Object-Oriented Simulation*. Piscataway, NJ, USA: IEEE Press, 1996, ch. 4.

[33] L. Yilmaz, "On the need for contextualized introspective models to improve reuse and composability of defense simulations," *J. Defense Model. Simulat. Appl. Methodol. Technol.*, vol. 1, no. 3, pp. 141–151, 2004.

[34] J. W. Bae and T. G. Kim, "DEVS based plug-in framework for interoperability of simulators," in *Proc. Spring Simulat. Multiconf.*, Orlando, FL, USA, 2010, Art. no. 127.

[35] S. J. Kwon and T. G. Kim, "Design and implementation of event-based DEVS execution environment for faster execution of iterative simulation," in *Proc. Symp. Theory Model. Simulat. DEVS Integr. M S Symp.*, 2012, Art. no. 14.

[36] G. A. Wainer and N. Giambiasi, "Application of the cell-DEVS paradigm for cell spaces modelling and simulation," *Simulation*, vol. 76, no. 1, pp. 22–39, 2001.

[37] A. Muzy and J. J. Nutaro, "Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators," in *Proc. 1st Open Int. Conf. Model. Simulat. (OICMS)*, 2005, pp. 273–279.

[38] J. Himmelspach and A. M. Uhrmacher, "Sequential processing of PDEVS models," in *Proc. 3rd EMSS*, Barcelona, Spain, 2006, pp. 239–244.

[39] S. J. Kwon, K.-M. Seo, B.-S. Kim, and T. G. Kim, "Effectiveness analysis of anti-torpedo warfare simulation for evaluating mix strategies of decoys and jammers," in *Advanced Methods, Techniques, and Applications in Modeling and Simulation*. Tokyo, Japan: Springer, 2012, pp. 385–393.

[40] S. J. Kwon, C.-B. Choi, and T. G. Kim, "Event-oriented control functions for enhancing development process of war-game simulators," in *Proc. Eur. Model. Simulat. Symp. Conf.*, 2014, pp. 184–190.

[41] B. A. Cota and R. G. Sargent, "A modification of the process interaction world view," *ACM Trans. Model. Comput. Simulat.*, vol. 2, no. 2, pp. 109–129, 1992.

[42] T. G. Kim *et al.*, "DEVSim++ toolset for defense modeling and simulation and interoperation," *J. Defense Model. Simulat. Appl. Methodol. Technol.*, vol. 8, no. 3, pp. 129–142, 2010.

[43] B. G. Kang and T. G. Kim, "Reconfigurable C3 simulation framework: Interoperation between C2 and communication simulators," in *Proc. Winter Simulat. Conf. (WSC)*, Washington, DC, USA, 2013, pp. 2819–2830.

[44] C. Sung and T. G. Kim, "Collaborative modeling process for development of domain-specific discrete event simulation systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 532–546, Jul. 2012.

Se Jung Kwon received the B.S. degree with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2009 and the M.S. degree with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, in 2011, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering.

His current research interests include simulation algorithms for discrete event systems modeling and simulation, DEVS execution environments, and hybrid systems modeling and simulation.

Bonggu Kang, photograph and biography not available at the time of publication.

Changbeom Choi, photograph and biography not available at the time of publication.

Tag Gon Kim (SM'95) received the Ph.D. degree in computer engineering with specialization in systems modeling and simulation from the University of Arizona, Tucson, AZ, USA, in 1988.

He was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Kansas, Lawrence, KS, USA, from 1989 to 1991. He joined the Electrical Engineering Department, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 1991 and has been a Full Professor with the EECS Department since 1998. He has co-authored the text book entitled *Theory of Modeling and Simulation* (Academic Press, 2000). He published about 200 papers in modeling and simulation theory and practice in international journals and conference proceedings.

Dr. Kim was the President of The Korea Society for Simulation. He was the Editor-in-Chief for *Simulation: Transactions for Society for Computer Modeling and Simulation International*. He is very active in defense modeling and simulation in South Korea. He was/is a Consultant for defense modeling and simulation technology at various Korea government organizations, including Ministry of Defense, Defense Agency for Technology and Quality, Korea Institute for Defense Analysis, and Agency for Defense Development. He is a fellow of SCS.