Collaborative Modeling Process for Development of Domain-Specific Discrete Event Simulation Systems

Changho Sung and Tag Gon Kim, *Senior Member, IEEE*

***Abstract*—The discrete event systems specification (DEVS) formalism supports the object-oriented (OO) specification of discrete event models in a hierarchical, modular manner. If a system that is to be modeled is domain-specific, the development of models with the use of the DEVS formalism would require domain knowledge about the system as well as to understand DEVS semantics. This paper proposes a collaborative modeling process to compensate for the lack of professional engineers. To compensate, this modeling process utilizes three types of engineers: domain engineer, modeling and simulation (M&S) engineer, and platform engineer. The process consists of four steps: conceptual modeling, model partition, model implementation, and model integration/simulation. The system requirements are used to specify domain models in the conceptual modeling step, and the models are partitioned into two types: discrete event-level model (DEM) and behavioral-level model (BM). The DEM is specified as the DEVS formalism, and the BM is defined as algorithms and equations. Each model is implemented separately, and the implemented models are integrated and simulated flexibly by using a dynamic linking library. The modeling process is then applied to develop a war game simulator. The advantage of this modeling process is that the collaborative work is related to the whole series of steps. This collaboration maximally utilizes the capabilities of the professional engineers by seamlessly separating yet correlating their works.**

***Index Terms*—Collaboration, discrete event systems specification (DEVS), domain-specific, modeling process, simulation systems, unified modeling language (UML).**

I. Introduction

DISCRETE event modeling can be considered as a process of abstract knowledge representation of a real-world system. As a model, the representation should be executable within a simulation environment to analyze the modeled system with respect to modeling objectives. The process may be based on the different world views for modelers, such as event-oriented, process-oriented, and object-oriented (OO). Among them, the OO approach may be the most compatible with a real-world system from the system-theoretic viewpoint of model representation [1].

The system-theoretic representation first specifies a system as a set of inputs, a set of outputs, and a set of states. It then defines a set of operations on the representation. The discrete event systems specification (DEVS) formalism [2], which represents a discrete event system from the system-theoretic viewpoint, is known to be compatible with the OO world view. Moreover, formalism supports the hierarchical and modular specifications of discrete event models, which allows us to assemble previously developed component models in a flexible manner.

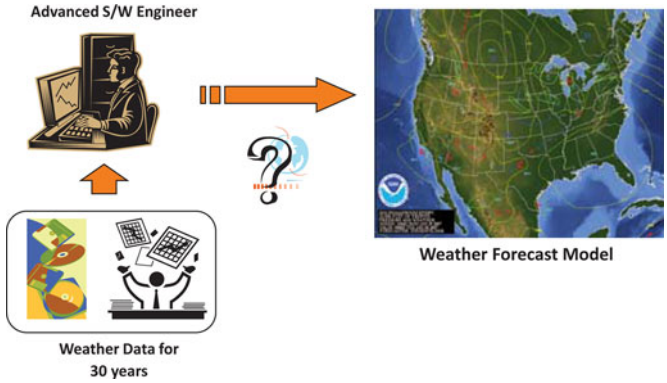

Fig. 1. Difficulty of developing domain-specific simulation systems.

Given our modeling objectives, the modeling and simulation (M&S) of domain-specific systems requires a great deal of domain knowledge. Fig. 1 shows the difficulty of developing models for domain-specific complex systems. Assume that an advanced software engineer has a great deal of knowledge of software technologies, such as programming. When a software engineer develops a weather forecast simulation model, he needs information for a weather system. Although he has weather data details, he cannot make the best use of the data in the development of the simulation model due to his lack of domain-specific knowledge about meteorology. In addition to his lack of domain knowledge, the software engineer may fail to develop the model due to his lack of M&S skills. The simulation model is defined by a set of instructions, rules, equations, and constraints to generate output behaviors from inputs (I/O). Thus, a modeler has to specify the relationship of the I/O and the dynamics. No matter how much the software engineer may have learned from meteorology, he cannot develop the weather model without certain M&S theory. Hence, M&S theory experts are needed as well.

To overcome the difficulty of developing domain-specific simulation systems, we propose a collaborative modeling

Manuscript received October 28, 2010; revised February 13, 2011; accepted March 14, 2011. Date of publication April 29, 2011; date of current version June 13, 2012. This work was supported by the Defense Acquisition Program Administration and Agency for Defense Development under Contract UD080042AD, Korea. This manuscript is an extended version of the invited talk given by T. G. Kim at the 2009 Spring Simulation Multiconference—DEVS Integrated M&S Symposium, San Diego, CA. This paper was recommended by Associate Editor S. Ramaswamy.

The authors are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: chsung@smslab.kaist.ac.kr; tkim@ee.kaist.ac.kr).

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/TSMCC.2011.2135850

process. The process is based on hardware/software (HW/SW) codesign [3], system integration [4], and a collaboration method [5]. The purpose of the process is to put the right professional engineers in the right place. Domain engineers analyze functional requirements about domain knowledge, and M&S engineers design the discrete event model by using the analyzed requirements. Platform engineers [6] implement the models and compose the real systems. Therefore, the development of a domain-specific discrete event simulation system requires cooperative teamwork among experts throughout the whole modeling process, and this paper describes that modeling process.

This paper is organized as follows. Section II presents several of the related works and previous approaches to the M&S of a domain-specific system. Section III introduces some of the basic knowledge about modeling a system. Section IV discusses the roles of professional persons and the necessity of collaboration. Our proposed collaborative modeling process and case studies are described from Section V to Section VII. Finally, Section VIII concludes the discussion.

II. Related Works

In past years, there have been some efforts to develop software systems of particular problem domains. Typically, the domain-specific software engineering [7] is a methodology to analyze regions of the particular domains and build domain-specific software architecture (DSSA) and application software. The DSSA needs to be adapted to the culture of the domain to which it is being applied. The domain model, reference requirements, and reference architecture are represented by using the tools and methodology that the application developers in the domain traditionally used. DSSAs address component reuse [8] and substitutability by identifying the components and topologies reused across architectures in a given application domain, and this reduces development and maintenance costs.

The unified modeling language (UML) is a widely used visual modeling language that is fit for different software development methods and every phase of the software life cycle. The UML 2.0 provides 13 diagram types that enable engineers to model several different views and abstraction levels [9], and then the engineers use the diagrams to describe the domain model of the DSSA [10]. For that reason, the UML has been a representative modeling language for domain analysis, and it serves as a bridge between application domains and discrete event system models.

In the DEVS-based M&S domain, Hong proposed embedment of the UML diagrams into the development process of DEVS models for the first time in [11]. The UML was used as a means of communication between a system domain and DEVS models. M&S engineers develop a simulation model by using documents that are produced by domain engineers. These documents are UML diagrams, and a sequence diagram is transformed into a DEVS diagram. As another study, eUDEVS [12] was proposed, which was an automatic M&S process to develop DEVS models. The tool transforms the UML diagrams into DEVS atomic and coupled models automatically by using XML. However, UML modeling power is not complete enough

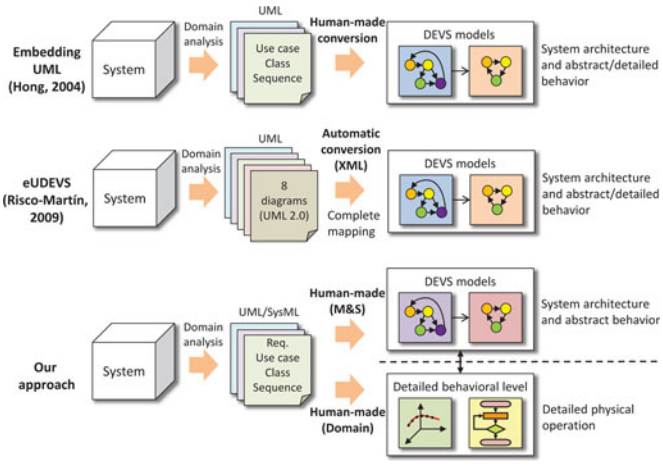

Fig. 2. Comparison with previous research.

to represent discrete event models. In addition, the approaches are the sequential process. Thus, the DEVS models are not built before the developers draw UML diagrams completely.

To overcome the drawbacks of the sequential development process, a comodeling methodology [13] had been proposed in a previous research. This methodology is similar to HW/SW codesign. A model is partitioned into two modeling levels for an object, and each model is developed by using DEVS and UML. The advantage of this methodology is that the domain engineers and the M&S engineers can work simultaneously through a predefined interface. However, the previous research does not mention the development process of simulator architecture from simulator requirements, and it does not formally specify the discrete event system with the predefined interface.

On the other hand, there is ample research that supports the division of roles to develop complex software systems in the software engineering field. Since software development is a difficult and complex task, Zhu *et al.* [14] proposed a role-based software process for the development of software. Similarly, the modeling Turnpike (mTurnpike) [6] clearly separates the task to model and program domain-specific models from the task to transform them into the final compilable code. This design strategy improves the separation of concerns between modelers and platform engineers. Modelers do not have to know how domain-specific concepts are implemented and deployed in detail, and platform engineers do not have to possess detailed domain knowledge and UML modeling expertise. This separation of concerns can reduce the complexity of application development and increase the productivity in modeling and programming domain-specific concepts.

To contribute to these existing works, this paper proposes an efficient and systematic modeling process to develop domain-specific system models. As shown in Fig. 2, the common solution is the employment of UML as a means of communication between complex problem domains and the discrete event simulation model. We agree that the UML helps the DEVS model development. However, the UML is limited in its ability to represent the DEVS model because it lacks attributes, such as arbitrary events and random time advance. In addition, UML

is a visual syntax for software modeling, not a formalism. This makes it hard for its designers to execute and verify the simulation model. In our approach, we use the UML only as a secondary means of DEVS model development. Another problem of previous research studies is their failure to specialize the work of experts in the M&S process. To overcome this problem, we propose a collaborative modeling process that helps us to maximize the abilities of experts by seamlessly separating yet correlating their works.

The collaboration approach, of course, has been studied in system M&S. Barjis [15] introduced collaborative, participative, and interactive modeling in the systems engineering. He discussed collaboration and interaction among modeling experts and system users to model and simulate complex systems. Particularly, system users are involved in system development process. However, the role of experts and users are conceptual; therefore, there is a lack of detailed and formal description about their interaction. Our previous research [16] also discussed collaboration among M&S stakeholders, including modeling experts and users.

This paper focuses on collaboration among experts about M&S except system users. Our approach is described in Fig. 2. We partition the system modeling into two levels according to model abstraction level: abstract behavior and detailed physical operation. With the collaborative work, the model partitioning skill in the proposed modeling process allows us to develop the discrete event systems efficiently and flexibly through the modified specification formalism. In addition, the collaborative modeling process improves the reusability of the partitioned models.

III. Backgrounds

This section provides background knowledge about modeling a system. The UML is a well-known standard language for the development of OO systems. The UML allows us to easily understand system requirements and specifications. The DEVS formalism is a general methodology to describe discrete event systems, and we use a DEVSim++ library [17] to model and simulate a DEVS model.

A. Unified Modeling Language and Requirements Diagram of System Modeling Language

The UML is a standardized language to specify, visualize, and document the artifacts of an OO system under development [18], [19]. Note that UML is not a kind of modeling methodology, but instead it provides a visual syntax that we can use to construct models. It simplifies the complex process of software design and presents the various views of a system, thus making a blueprint for construction [20]. Objects contain information and may perform functions, and UML models have a static structure and dynamic behavior. The former relates to the type of objects and the relationship among them, and the latter denotes the functions of the systems [21]. This section describes only three diagrams among several UML diagrams. The diagrams include a use case diagram, a class diagram, and a sequence diagram that is a subdiagram of an interaction diagram. For a detailed analysis of systems requirements, we use a system modeling language (SysML) requirements diagram [22].

1) Requirements Diagram: A SysML requirements diagram is a new diagram that is intended to cover the UML shortage. It helps engineers to better organize requirements, and it also shows explicitly the various kinds of relationships between different requirements. The diagram can fill the gap between natural-language-based specifications, which are too ambiguous and informal, and the UML uses case diagrams. In addition, the SysML requirements diagram can facilitate the transformation of user requirements into system requirements and improve the requirements' traceability throughout the design life cycle.

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

3) Class Diagram: A class diagram is a structure diagram that shows a set of classes, interfaces, and/or collaborations and the relationships among these elements. A class includes the name, attributes, and operations. This diagram is a central modeling technique that is a part of nearly all OO methods and represents the static part of a system.

4) Sequence Diagram: A sequence diagram is an interaction diagram that focuses on the temporal ordering of messages among objects. A sequence diagram depicts a sequence of actions that occur in a system; therefore, it is a very useful tool to easily represent the dynamic behavior of the system. This diagram includes objects and messages in a 2-D form. The horizontal axis shows the life of the objects that it represents, while the vertical axis shows the sequence of the creation or invocation of these objects.

B. Discrete Event Systems Specification Formalism

The DEVS formalism specifies discrete event models in a hierarchical and modular form. With this formalism, one can perform modeling more easily by decomposing a large system into smaller component models with a coupling specification between them. There are two kinds of models: atomic and coupled [2].

An atomic model is the basic model and contains the specifications for the dynamics of the model. A coupled model provides the method of assembly of several atomic and/or coupled models to build complex systems hierarchically. An overall system consists of a set of component models, either atomic or coupled, thus exhibiting a hierarchical structure. Each DEVS model, i.e., either atomic or coupled, corresponds to an object in a real-world system to be modeled [23]. Within the DEVS framework, model design may be performed in a top-down fashion, while model implementation is conducted in a bottom-up manner.

1) Simulation of Discrete Event Systems Specification Model: Simulation of a DEVS model can be done by communicating hierarchical abstract simulators, the architecture of

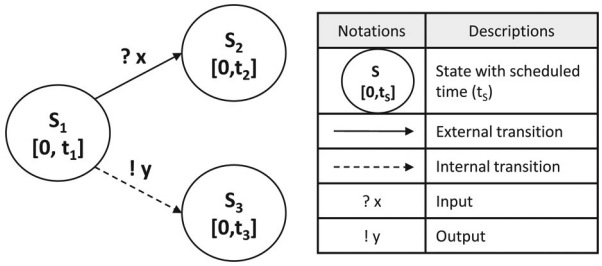

Fig. 3. Graphical notation of DEVS atomic model.

which is the same as the DEVS model architecture. The abstract simulators are a set of distributed simulation algorithms that can be implemented in a sequential as well as in a distributed computing environment [24]. Since DEVS models are developed in an OO manner, OO programming languages, such as C++ and JAVA, are appropriate for implementation of a DEVS M&S environment. In fact, the first such C++ implementation is DEVSim++ [17] in which modeling facilities and abstract simulators are explicitly separated. Within the environment, modeling facilities are opened to modelers; abstract simulators are not accessible externally. The modelers can develop DEVS models by using a modelers' interface, which is a set of application programming interfaces (APIs) to specify DEVS models in DEVS semantics. Thus, APIs for the specification of DEVS models are defined such that there is a one-to-one correspondence between APIs and functions in the formalism. Once DEVS models are developed by using the facilities and APIs, engineers can perform a simulation of such models, thus using abstract simulators that are embedded in DEVSim++.

Various implementations for DEVS M&S are available in public domains, and the DEVS Standardization Group [25] is making an ongoing effort to standardize DEVS modeling/simulation environments.

2) Graphical Representation: An atomic model of the DEVS formalism is illustrated in a labeled graph. There is no standard for the graphical representation of an atomic model. In this paper, we define the DEVS graph as follows:

$$G = \langle V, E \rangle$$

where V is a set of sequential states with scheduled time, where $V = S$ with ta of S, and E is the internal or external transition with labels, where $E \subseteq V \times V$ with a label $(X \cup Y)$.

The graphical notations are defined as shown in Fig. 3. A state with $[0, t_S]$ means that a model waits for any input events in simulation time t_S at the state. A solid line means an external transition with an input event, and a dotted line means that an internal transition happens after an output event occurs in an output function. The graphical representation allows us to understand the DEVS atomic model on sight.

IV. Role of Experts

Commonly, system engineering requires many professional engineers from diverse areas. Among them, we employ three types of engineers in this paper: domain engineers for domain and requirements engineering; M&S engineers to construct the overall modeling process of discrete event systems; and platform engineers to program and test the simulators.

A. Domain Engineers

Domain engineering (DE) is a software engineering discipline that includes the identification, analysis, and design of domain-specific capabilities. In general, DE builds software architectures and reusable assets to address the problems of system development within a domain. The three main activities of DE are domain analysis, domain design, and domain implementation [26]. Domain engineers are the people who perform the DE activities. They perform the domain analysis with domain information, which are gathered through questionnaires, domain expert interviews, a review of relevant information from documentation, reverse reengineering of existing systems, and independent research [27]. Based on the domain requirements, the domain engineers generate the behavioral-level model (BM), which includes mathematical equations, algorithms, and strategies.

B. Modeling and Simulation Engineers

M&S engineering refers to the use of models, including emulators, prototypes, and simulators, either statically or over time, to develop data as a basis to make managerial or technical decisions [28]. The activities of M&S engineering are requirements engineering, modeling theory, simulator implementation/verification, model validation, model behavior analysis, and performance evaluation. In this paper, M&S engineers are in charge of the M&S of discrete event systems according to domain requirements. They are especially interested in the high-level abstract behavior of objects. They generate the discrete event-level model (DEM), which is specified as the DEVS formalism.

C. Platform Engineers

Platform engineers are general program developers who implement simulation models by using model specifications for domain engineers and M&S engineers. They also take charge of construction of the simulation environment, such as geographic information system (GIS), DB, and user interface, and they possess expertise in the platform technologies on which models are deployed. In general, platform engineers do not have to know the details of domain-specific concepts.

D. Collaborative Work

As we have seen in Section I, the cooperation among experts of the various fields is essential to model and simulate domain-specific, complex systems. Fig. 4 shows the role of each expert in developing domain-specific discrete event simulation systems. The M&S of domain-specific systems requires the integration of each expert's professional knowledge. The domain requirements consist of textbooks, field experience, physics, and several data sources, such as conversions about the weather. They are the necessary, basic units to understand the dynamics of the systems, and they are used by domain engineers, who major in electrical

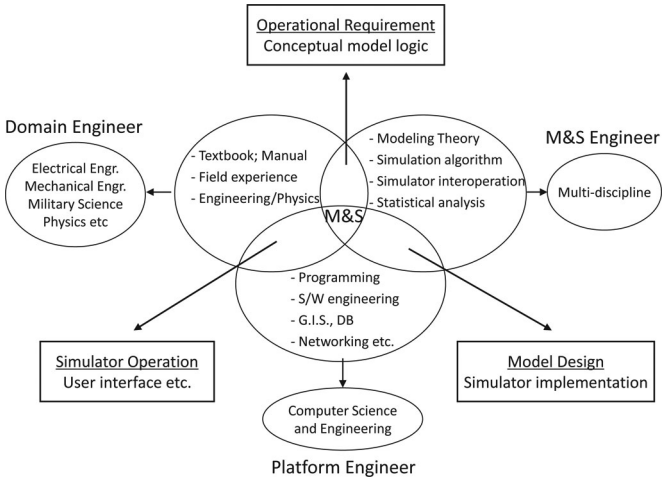

Fig. 4. Expert cooperation for M&S.

engineering, military topics, physics, etc. The material contains the operational requirements of a discrete event simulator, and it is utilized to model conceptual logic and architecture as well as to provide algorithms that detail the operations of a system. M&S engineers observe the domain objects with a view that allows them to analyze the high-level aspects of the objects. On the other hand, domain engineers design detailed algorithms of physical phenomena. Furthermore, designed models that are developed by M&S engineers are used to implement a simulator by platform engineers, who use software technologies, including several programming languages and tools. In general, domain engineers analyze the domain requirements, and they are also the end users, who execute the simulator. Therefore, platform engineers have to implement a simulator in cooperation with the domain engineers to apply a demand from end users. In brief, collaborative work among professional engineers is essential to develop domain-specific systems.

V. Collaborative Modeling Process

In HW/SW codesign, the system specification is partitioned into hardware and software parts for the concurrent design process. After that, each part is implemented and then integrated for cosimulations [3]. The proposed collaborative modeling is based on this HW/SW codesign. The process is a procedure to develop simulation systems by separate groups of professional engineers, who cooperate with each other. The professional engineers are domain engineers, M&S engineers, and platform engineers, as we have mentioned in Section IV.

The development of domain-specific discrete event systems is a difficult task, which requires a high degree of expert knowledge from the engineers who are involved. The proposed process is similar to the general system development process [29], which include the progression from requirements analysis to system testing. As shown in Fig. 5, the overall modeling process is partitioned into four parts: conceptual modeling, model partition, model implementation, and model integration/simulation. The process begins from analyzing system requirements specification that is generated by system users. The users are expected to tell the professional engineers what needs to be done as well as the motivation for the development from the system user's perspective. Conceptual modeling step involves requirements analysis and an understanding of domain knowledge. After the conceptual modeling, the designed model is partitioned into two levels: DEM and BM. These partitioned models are independently implemented by platform engineers. Finally, the separately implemented models are integrated via communication interfaces, which are predefined within the modeling formalism and supporting libraries.

In a practical manner, each process step supports technical collaboration. Collaborative work is applied from the beginning to the end. The first step, making domain models and the simulator's architecture, involves drawing UML and SysML diagrams in practice. The diagrams have to be drawn with the collaboration of all engineers. In the second step, each model is specified by using a suitable formal description, such as the DEVS formalism and mathematical equations. The communication interface between formal descriptions has to be defined, the engineers collaborate over the formalism specifications. In the third step, the experts implement their models separately in the programming viewpoint. Finally, the implemented form of the BM is linked dynamically to the DEM; thus, collaboration in the run-time execution is supported technically.

A. Conceptual Modeling

The first step of the modeling process is the conceptual modeling of a system. A conceptual model is a statement of the content and the internal representations that are the user's and the developer's combined concept of the model. The model includes logic and algorithms, and it explicitly represents assumptions and limitations. The purposes of the conceptual modeling in the domain-specific simulation systems are to clearly define the meaning of terms and concepts that are used by domain engineers to discuss the problem and to find the correct relationships among various concepts. The conceptual model is the basis for subsequent application developments in the domain, and it is built by data modeling. A requirements engineering approach can be used to model the data [30]. The conceptual model is also made independent from implementation details by representing the model with symbols, such as UML and entity relationships). We adapt UML diagrams and SysML requirement diagrams for the conceptual modeling process. Normally, the graphical approaches are more efficient than text-based analysis [31]. In [32], UML is used for the conceptual modeling of the structure and behavior of a system. SysML is applied to the development of simulation systems, especially conceptual analysis, such as requirements modeling [33], [34].

A general complex system consists of structural and behavioral parts. In order to express the parts of the system, we use UML *use case*, *class*, *sequence diagrams*, and a SysML *requirements diagram*. Note that other UML diagrams can be optionally required in addition to the basic diagrams for detailed analysis. We call the diagrams for conceptual modeling as the *domain model*. The domain model is made by collaborative work among experts. All diagrams are drawn according

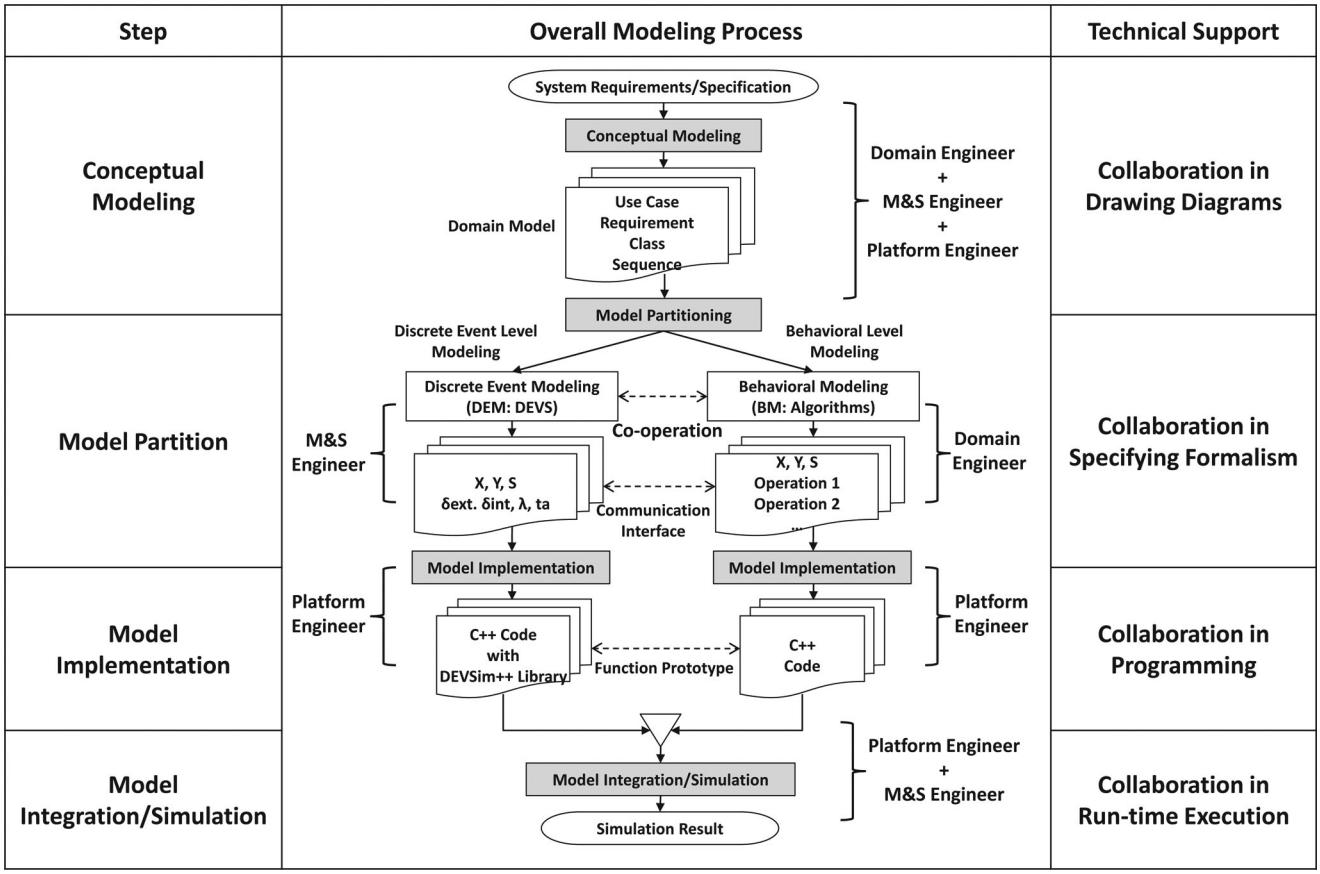

Fig. 5. Overall collaborative modeling process.

to an analysis of requirements, and domain engineers and M&S engineers take part in analyzing and modeling simulation system requirements from an OO point of view. Platform engineers take charge of the deployment and implementation of a system. After that, they design the simulator architecture together by using the domain models.

B. Model Partition

The domain model that was developed in the conceptual modeling step is decomposed into submodels from a top-down perspective. The architectural design is completed from an OO modeling viewpoint. The OO modeling approach for system modeling views a system as an object in which its representation and associated operations are explicitly defined. A designed model presents an object, and the model is partitioned into two modeling levels of the object in terms of layered structure architecture [13]: DEM and detailed BM. The domain model describes the dynamic behavior of an object, and the behavior includes the I/O events of a model, state transitions, and operations that are processed by specific events and state transitions. Among them, state transitions by I/O events are described by M&S engineers by using the DEVS formalism, and detailed operations at specific states are described by domain engineers by using algorithms and mathematical equations. Fig. 6 shows a simple example of model partitioning.

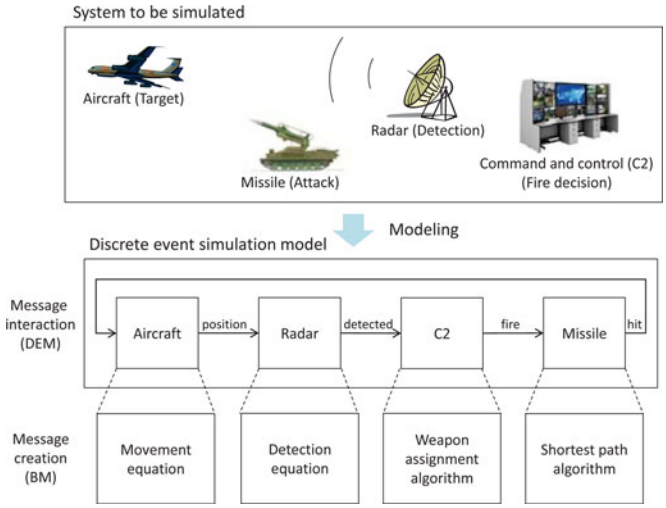

Fig. 6. Simple war game model partition.

Assume that a system consists of four military objects. An enemy aircraft moves to attack our airbase, and one of our radars detects the aircraft. In order to defend our airbase, a

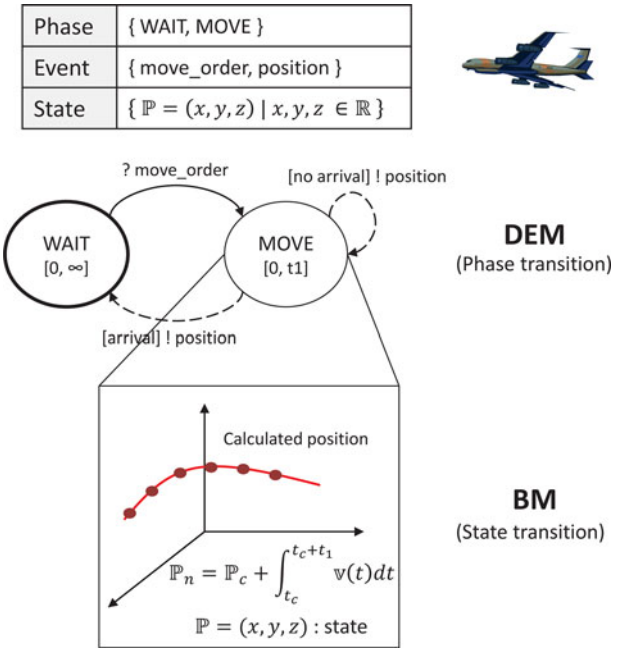

Fig. 7. Example. Phase and state transition.

command and control system decides what and how we attack, and then, a missile is launched at the enemy aircraft. This system is modeled as a simple war game, and a general war game model is conducted by I/O events. Each object is represented as a DEVS atomic model, and messages in the events are created by calculating mathematical equations and algorithms, such as movement equations and shortest path algorithms. Specifically, an aircraft moves to update its current position, and the position information is determined by 3-D movement equations in the case of takeoff, flying, and landing. Most of the equations are complex, and they are too detailed to be suitable for DEM. In the simple case, the structured and high-level state transition model is the DEM, and the detailed model for the generation of information is the BM.

In order to explain DEM and BM in detail, we use the concept of a *multimodel* [35], [36]. A multimodel is a model that is composed of other models. The purpose of a multimodel is to have a unifying system representation that contains many different levels of abstraction and perspective views within the system [37]. In our methodology, each level of a multimodel can be represented by either DEM or BM models. The concept of the multimodel allows us to create a more flexible modeling environment wherein each level is represented by its most appropriate form.

In a multimodel, a *phase* means a set of contiguous states that are equivalent to each other in the sense of λ and ta, and qualitative changes in behavior are *phase transitions*. A *state* means a low-level state, and the *state transition* occurs within a phase; a phase is a high-level state. The relation between the phase and the state is shown in Fig. 7. An aircraft waits for the command *move_order*. When the event occurs, the phase changes from *WAIT* to *MOVE*, and the aircraft starts to move to a designated point. The phase transition is defined as the DEVS formalism of DEM. During the movement of the aircraft, the *move* operation is executed periodically in the *MOVE* phase. The operation is defined as a *communication interface* between DEM and BM. The operation of the BM generates an event to change the phase of the DEM. The event includes the current position value $\mathbb{P}$ that is coordinated with the x-, y-, and z-axes; and $\mathbb{P}$ is also a state value of the BM. The operation is defined as a mathematical motion equation, such as uniform velocity and uniform acceleration motion. When the coordinates are located in the designated point, i.e., when the aircraft arrives at the point, the phase changes from *MOVE* to *WAIT*.

We extend the classic DEVS formalism in order to explain the activities within the phases. The extended form is similar to real time DEVS (RTDEVS) [38]. An atomic model of the extended DEVS formalism has I and ψ_I in addition to the sets and functions of classical formalism as follows:

$$\mathrm{AM}_{\mathrm{DEM}} = \langle X, Y, S, I, \delta_{\mathrm{ext}}, \delta_{\mathrm{int}}, \psi_I, \lambda, \mathrm{ta} \rangle$$

where

- X set of input events;
- Y set of output events;
- S set of phases;
- I set of communication interfaces;
- δ_{ext} $Q \times X \to S$, an external transition function, where $Q = \{(s, e) | s \in S \text{ and } 0 \le e \le \mathrm{ta}(s)\}$, total phase set of M;
- δ_{int} $S \to S$, an internal transition function;
- ψ_I $S \to I$, an interface mapping function;
- λ $S \to Y$, an output function;
- ta $S \to R^{+}_{0,\infty}$ (nonnegative real number), time advance function.

I is a set of communication interfaces, and the set is specified in the BM. The interface specification is in the form of algorithms and equations. ψ_I is an interface-mapping function, and it connects the phases to the proper communication interfaces. When the external and internal transition functions are executed, the mapping function is activated and refers to the interface that is fully described in the BM. Therefore, specifying the DEVS model of the DEM means defining four sets ($X, Y, S,$ and I) and five functions ($\delta_{\mathrm{ext}}, \delta_{\mathrm{int}}, \psi_I, \lambda,$ and ta). Through the extended DEVS formalism, we can formally specify the discrete event simulation systems that communicate with other computing operations.

C. Model Implementation

In the collaborative modeling process, M&S engineers specify DEVS models that represent system behaviors at the phase-transition level, as shown in the bottom diagram of Fig. 7. Domain engineers are responsible for the development of specific algorithms as individual functions. These algorithms typically require domain-specific knowledge. After that, each engineer implements a simulation model of his own specification with platform engineers. Similarly, the model implementation is separated into the DEM and the BM implementations. Each model is coupled with a communication interface, which is the so-called *function prototype* in C++ language. Platform engineers

implement the high-level DEVS models of the DEM by using C++ language with DEVSim++ library, and they also use C++ language to implement the detailed algorithms and questions of the BM.

Separation of implementation eases the testing and maintenance of simulation models. In general, a modification of simulation requirements brings more frequent changes in detailed algorithms than changes in abstract behavior. When a variation of rules or scenarios alters a specific algorithm, M&S engineers do not have to perform the modification, if the change only affects the behavioral operations of the modeled objects. Technically, this switch of model algorithms at the run-time or the load-time of a simulation run becomes possible because of the dynamically linked library (DLL) technology [39].

Using the DLL in the implementation step improves the reusability of the algorithms of the BM. It is one of the most efficient methods to develop combined simulation model because of its flexibility and simplicity. When the algorithms are implemented as the DLL, we can only know the inputs, the output, and the name of the function prototype because of the modularity of the library. This means that we can employ the BM in simulation with another DEM, if the DEM is implemented by using the same function prototype with the same I/O. Therefore, separating implementation in the collaborative modeling process allows us to develop the models efficiently because of the reusability of the models from the programming viewpoint.

D. Model Integration/Simulation

The majority of collaborative work between M&S engineers and domain engineers in this modeling process happens in the specification of communication interfaces. The interfaces should be designed independent of other functions and should be as self-contained as possible. The function prototype is a C++ format of the communication interface, as shown in Section V-C. The DEVS models call associated functions within the phase-transition functions in the extended formalism. The contents of the function do not affect the model behavior as long as the I/O types of the function are consistent. Therefore, the simulation model is a combined form of the DEVS models and the implemented interface functions.

Platform engineers are able to supply a set of behavior functions, which is specified by domain engineers, in the DLL format. The library may be made implicit at the compile-time, or explicit by the application run-time. Traditional static linking binds a program and libraries together at the compile-time. Therefore, the static linking produces one executable file, including all data and codes. In this technology, the program should be recompiled in order to replace a function. However, a dynamically-linked library that contains functions to be replaced can be loaded after the application starts. If we implement each function in a separate DLL, the simulator will be able to execute user-supplied functions in the library without recompiling the simulator program.

In combination with the implemented DEVS models, a complete simulator is synthesized by selecting supplied DLLs. This selection can be done at run-time. Therefore, domain engineers can easily compare different combinations of algorithms without changing the DEM. These procedures are described in Fig. 8. DEM is a DEVS-coupled model as a composition of DEVS atomic models, and the models are developed by the M&S engineers and implemented by platform engineers, thus using the DEVSim++ library. The BM is a set of object classes that define numerous operations and detailed algorithms of the modeled objects. Platform engineers implement the BM by using C++ in the form of a DLL. The DEM and BM are synthesized at the run-time, and after that, a DEVS atomic model is linked to a corresponding object class. Although the modification of simulator requirements affects the change in the contents of object operations, the modified object class is easily linked to a DEVS atomic model. In addition, different simulation results can be achieved under varying operations of objects without recompiling a simulator through flexible simulator synthesis. After the data is collected through the simulation, M&S engineers analyze the data and evaluate the simulation model.

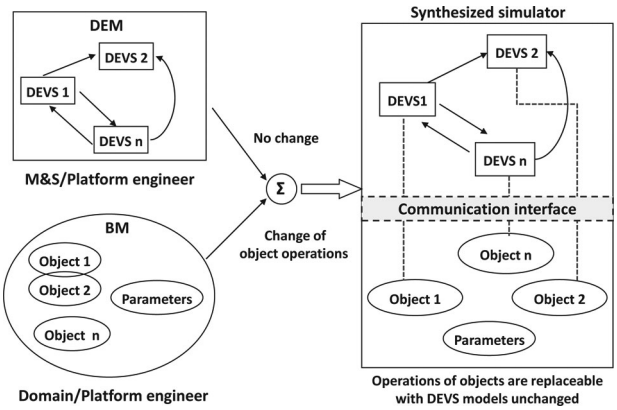

Fig. 8. Flexible simulator synthesis.

VI. Case Study: Amphibious Operation of the Marine Corps

The proposed process is applied to develop a war game model of an amphibious operation of the Marine Corps. A general war game model is a representation of the domain-specific, discrete event simulation system, and it has many objects and complex strategies. The number of objects for the training war game model is about 12 000 in practice [40], and the model has many decisions and operations. Domain knowledge about the war game model consists of tactics, strategy, and information of equipment, such as weapons, warships, and aircraft. The knowledge is very professional and specialized; therefore, domain engineers are definitely necessary. The objective of this case study application is to simulate and analyze a simplified training exercise scenario. The simulator is useful to evaluate the efficiency of a plan of military operations.

An amphibious operation of the Marine Corps consists of four parts: movement to the amphibious objective area (AOA), reconnaissance, fire support, and ship-to-shore movement. Generally, a troop moves from a port of our forces to an AOA of the enemy force in order to conduct an amphibious operation.

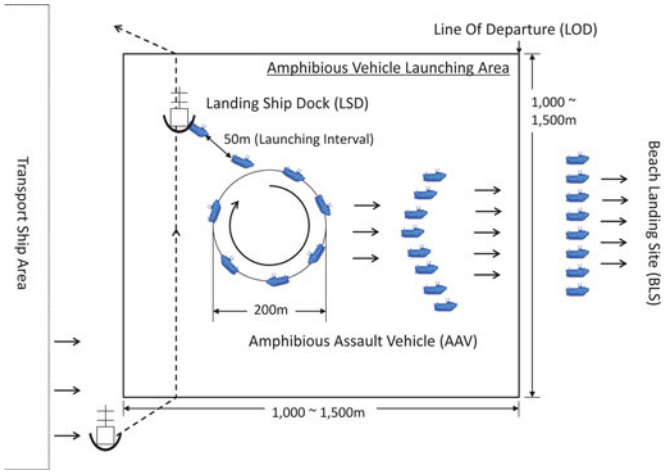

Fig. 9. Exercise scenario: ship-to-shore movement of amphibious operation.

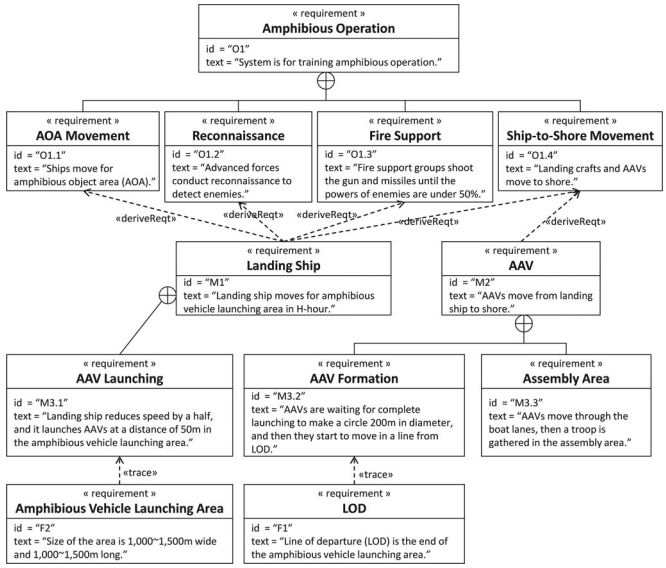

Fig. 10. Requirements diagram for amphibious operation system.

An advance force searches for enemies, and the fire support group attacks them for easier amphibious operation. After that, the main scheduled wave starts to move to shore. The ship-to-shore movement is the most important task of the amphibious operation. The entities of the scenario include 10 ships, three advance forces, eight fire support groups, 20 landing groups, and 10 enemy forces. All operations are activated by events, and the events are generated by external inputs or mathematical equations and algorithms in each entity. In order to simplify the scenario, we only explain the ship-to-shore movement.

Fig. 9 depicts a simple exercise scenario for the ship-to-shore movement of an amphibious operation. A landing ship dock (LSD), which is a form of amphibious warship designed to support amphibious operations, moves to an amphibious vehicle-launching area, a zone with the form of a square. The square is 1 000–1 500 m wide and 1 000–1 500 m long. An LSD has many amphibious assault vehicles (AAVs), and it transports AAVs on the square. After that, it starts to launch AAVs in the launching area, and the launching interval is set at a distance of 50 m. The AAVs orbits make a circle of 200 m in diameter until the LSD finishes launching them. When the LSD completes the launch, the AAVs move to the end of the launching area, which is called the line of departure, and they start to move in a line that head toward the beach landing site. After landing, a company of AAVs assembles in an appointed area and conducts its mission. We generated this scenario from a field manual.

A. Conceptual Modeling

During the conceptual modeling step for the development of the amphibious operation simulator, all engineers make domain models in cooperation with each other. A requirements diagram is shown in Fig. 10. An amphibious operation consists of four operations, and the requirements of a landing ship and an AAV are necessary to satisfy the requirement of a ship-to-shore movement. Likewise, other requirements are related to each other.

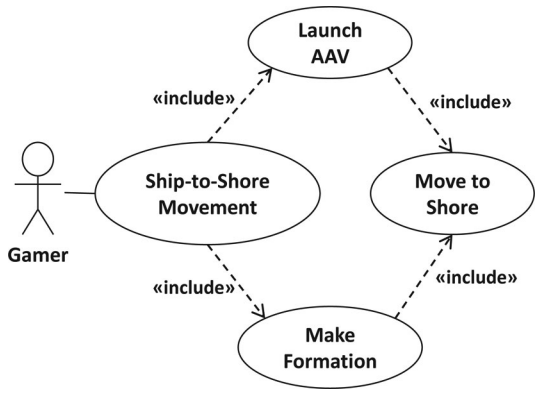

Fig. 11. Use case diagram for ship-to-shore movement.

The use case diagram for the *ship-to-shore movement* function is described in Fig. 11. The actor is a gamer, and the gamer controls the *ship-to-shore movement* use case. The use case is also related to other use cases. We can understand the functionality of the simulator from the following use case diagram.

The dynamic behavior of the *ship-to-shore movement* use case is represented by a sequence diagram, as depicted in Fig. 12. Three objects are necessary to satisfy the requirement. The operation is started with the message *start movement*. After that, an object *LandingShip* conducts the *Move* operation and creates the object *AAV* in the amphibious vehicle-launching area. Launched AAVs circle around in front of the ship. When the AAVs arrive at the shore, the object *LandingForce* is created, and the object *AAV* is destroyed. Through the sequence diagram, we can discern the dynamic relationship between objects in the *ship-to-shore movement* use case. The sequence diagram will be the basis of modeling discrete event simulation systems in the next section.

In addition to these diagrams, an extra diagram is optional for structuring the overall simulator architecture. The diagram

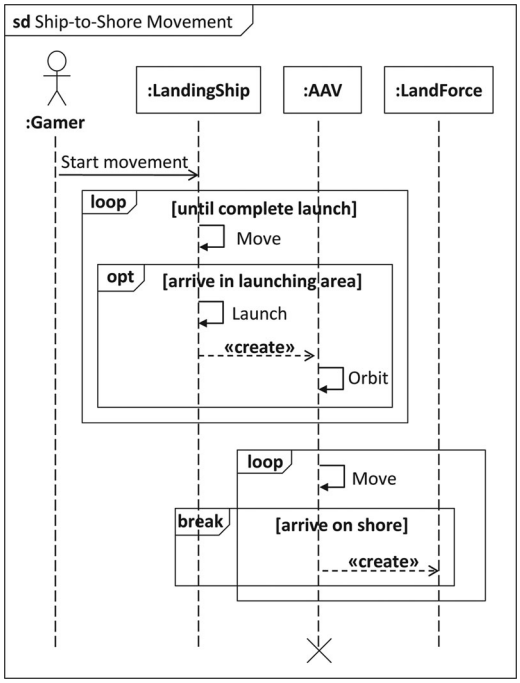

Fig. 12. Sequence diagram for ship-to-shore movement.

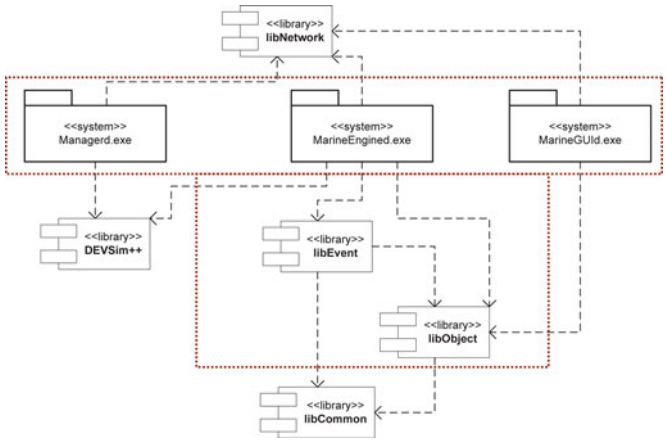

Fig. 13. Component diagram for amphibious operation system.

is a component diagram, as illustrated in Fig. 13. A component is a replaceable, executable piece of a larger system whose implementation details are hidden [41]. The component diagram shows the relationships among components, and platform engineers have to support drawing the diagram. Fig. 13 depicts the simulation model (*MarineEngined.exe*), libraries, and simulation environments (*Managerd.exe* and *MarineGUId.exe*).

B. Model Partition

The domain models of the amphibious operation simulator are partitioned into the DEM and the BM. Since a model is built for an object from an OO viewpoint, an AAV object is developed

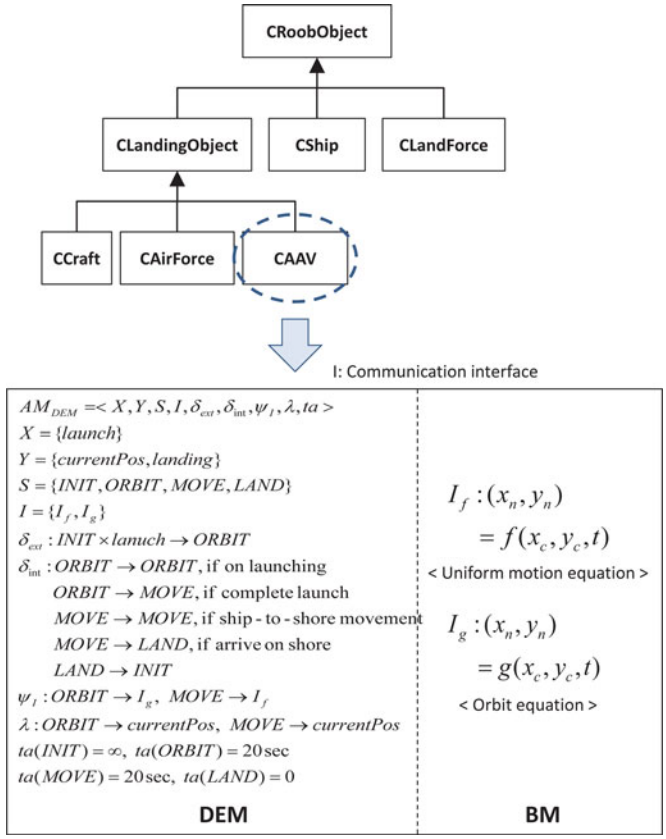

Fig. 14. Partition of AAV model.

as a model, and the model is partitioned into the DEM and the BM, as shown in Fig. 14. There are five types of objects, as shown in the class diagram in Fig. 14. Among them, the CAAV object is specified as a DEM and a BM. The DEM of the object is stipulated as the extended DEVS atomic model, as described in Section V-B. There are two communication interfaces: I_f and I_g, and each interface is linked to *MOVE* and *ORBIT*, respectively. In BM, the interfaces are described in detail as equations, such as a uniform motion equation and orbit equation. The detailed descriptions of the DEM and the BM are shown in following sections.

1) Discrete Event-Level Model: At the DEM level, a landing ship and an AAV are specified in the form of a phase transition by using arbitrary events with a high-level viewpoint, and the models are built by M&S engineers. Fig. 15(a) and (b) shows the phase-transition diagrams of the DEVS formalism. The landing ship has three phases: *INIT*, *MOVE*, and *LAUNCH*. The AAV has four phases: *INIT*, *ORBIT*, *MOVE*, and *LAND*. The *start* event is initiated by the gamer, and the event changes the phase of the landing ship to *MOVE*. The phase changes of the AAV happens when the *launch* event occurs at the landing ship model.

2) Behavioral-Level Model: The detailed behavior of the object is modeled as the BM. An AAV moves from the landing ship to the shore to achieve the mission. The next position of the AAV is calculated periodically, and the equations of the

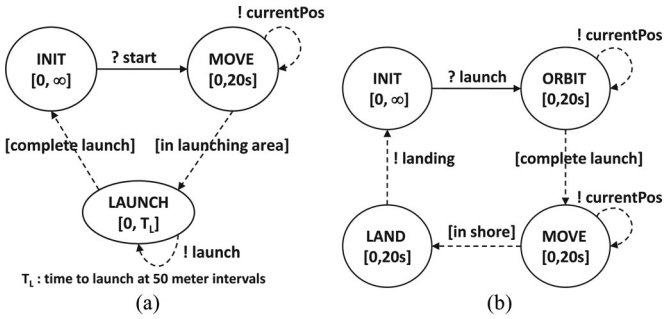

Fig. 15. DEVS diagram for landing ship and AAV. (a) Landing ship. (b) AAV.

operation are as follows.

$$I_f : CALCULATE\_NEXT\_POSITION$$

$$I_f \begin{cases} \dfrac{dx}{dt} = \upsilon_x, & x(0) = x_c \\ \dfrac{dy}{dt} = \upsilon_y, & y(0) = y_c \end{cases} \tag{1}$$

$$\upsilon_x = \text{spd} \times \sin(\text{hdg})$$

$$\upsilon_y = \text{spd} \times \cos(\text{hdg})$$

$$x_n = x_c + \int_{t_c}^{t_c+\Delta t} \upsilon_x dt$$

$$y_n = y_c + \int_{t_c}^{t_c+\Delta t} \upsilon_y dt$$

where

spd speed of AAV (in knots);
hdg heading of AAV (in degrees, range: 0°–360°);
t_c current time (in seconds).

The name of the operation is *CALCULATE_NEXT_POSITION*, and the arguments comprise the current position (x_c, y_c), next position after calculation (x_n, y_n), heading of the AAV (hdg), speed of the AAV (spd), and time step for the updation of the position (Δt). The equation includes the uniform motion equation, and the next position is calculated in (1). The equation changes the low-level states that indicate the AAV position information. The BMs of the AAV are built by domain engineers who have expertise in marine or mechanical engineering because the equations of the BMs are representations of detailed physical operations of the real system.

3) Communication Between Discrete Event-Level Model and Behavioral-level Model: The operation of the BM is connected to the proper phase of the DEM. The operation is called the *communication interface* between the DEM and the BM. As depicted in Fig. 16, the *CALCULATE_NEXT_POSITION* operation is linked with the *MOVE* phase. Thus, M&S engineers use the communication interface to design DEVS models, and the detailed description of the operation is a function of domain engineers.

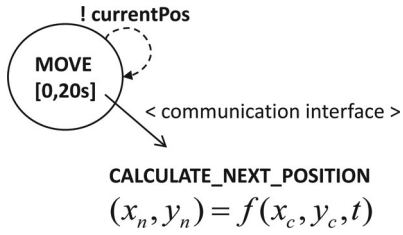

Fig. 16. Communication between DEM and BM.

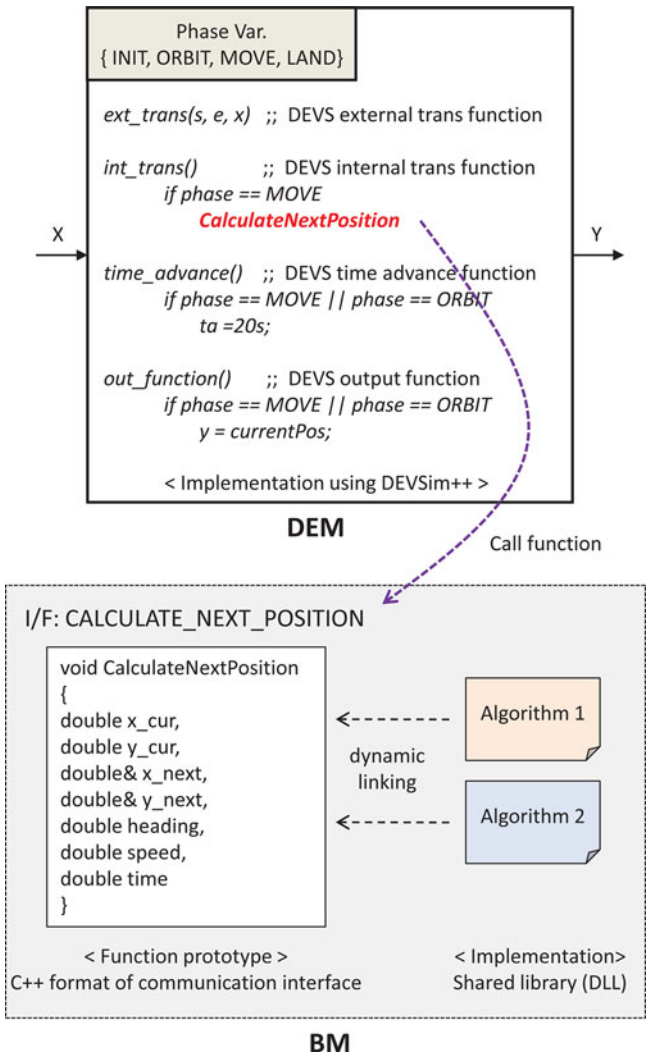

Fig. 17. Implementation of AAV in DEM and BM.

C. Model Implementation

Platform engineers implement the DEM and BM by using DEVSim++ [17] and C++, respectively. Fig. 17 presents the implementation of the DEM and the BM for an AAV. The DEM describes the abstract maneuver behavior of the AAV, as shown in Fig. 15(b). When the AAV phase is *MOVE*, the *CalculateNextPosition* function is called periodically, per 20 s in this example. The communication interface should be defined by the collaborative work of M&S and domain engineers

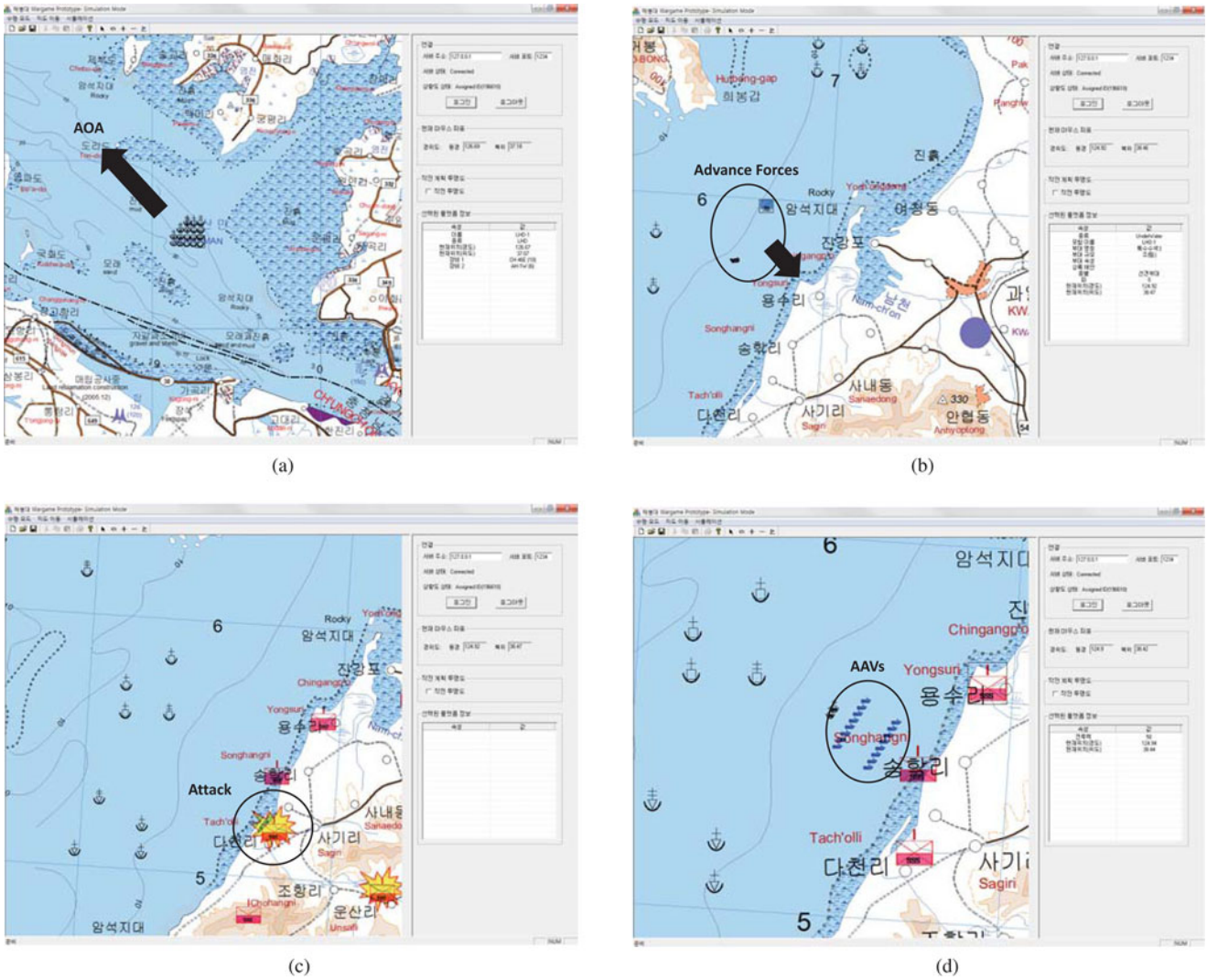

Fig. 18. Simulation of amphibious operation. The operations occur in sequence from (a) to (d). (a) AOA movement. (b) Reconnaissance. (c) Fire support. (d) Ship-to-shore movement.

when the model is partitioned or under development. The *CalculateNextPosition* function may use various algorithms and strategies, such as uniformly accelerated motion and uniform motion along a line. Each function is developed as a separate DLL .

D. Model Integration/Simulation

The simulation model is an integrated form of DEVS models and object algorithms, as shown in Fig. 17. The algorithms are implemented by using a shared library, such as a DLL, and the library is linked in the DEVS models automatically. The simulation result is represented in a GUI that is developed by platform engineers. According to the predefined exercise scenario, the amphibious operation is simulated, and the result is shown in Fig. 18. Fig. 18(a) depicts the movement from the home port to the AOA. Fig. 18(b) shows the reconnaissance mission, whereby the advance forces cross the sea and find the position of their enemies. The advance forces search for ten enemy forces, then the fire support groups attack the enemy forces until their combat power is reduced to half. The fire support is illustrated in Fig. 18(c). Fig. 18(d) represents the ship-to-shore movement of the AAVs, which is the most important mission of the amphibious operation of the Marine Corps. Through this simulation, an officer of the Marine Corps can evaluate the efficiency of a plan for exercise operations.

VII. Discussions

Within the collaborative modeling process, the M&S engineers and domain engineers design simulation models in close cooperation. They analyze the system requirements together in the conceptual modeling step, and they define the communication interface between the DEM and the BM in the model partition step. Platform engineers also cooperate with the other two types of engineers in designing a simulation environment in the conceptual modeling step and in implementing the specified models. The main advantage of this modeling process is that collaborative work is related to the whole series of steps. This collaboration maximizes the capabilities of the professional

TABLE I
APPLICATION: WAR GAME MODEL DEVELOPMENT

	Description	Development period	SLOC	Real effort (Person-Months)
Project 1	Navy war game model	04.06-06.06	66,968	240
Project 2	Air force game model	05.03-07.10	66,040	256
Project 3	Marine war game model	05.07-08.07	85,527	324
Project 4	Warship gun analysis model	04.01-05.12	53,653	124

TABLE II
DEVELOPMENT EFFORT OF APPLICATION USING COCOMO

	E_{min} (EAF: 0.6, P: 1.04)	E_{max} (EAF: 1.4, P: 1.24)
Project 1	116	630
Project 2	114	619
Project 3	150	853
Project 4	92	478

engineers by separating their work, especially in the large and complex simulation system. This specialization makes a concurrent development process possible.

In addition to collaborative work, the engineers apply extended DEVS formalism to explicitly specify the discrete event systems within the communication interfaces. Formalization eases the model verification, validation, and testing phases. Finally, the separation of design and implementation by using a DLL improves the reusability of the models.

We applied the proposed process to several war game model developments, and the result is shown in Table I. The applications are long-term projects that develop simulation models to train and analyze various military scenarios. All of them were developed by using the proposed process, and all the three engineer groups participated in the development of all applications. The real efforts are represented as the multiplication of the number of engineers and development months. In order to compare the real effort of the applications with the general software development process, we used the constructive cost model (COCOMO) [42] to estimate software development effort. Equation 2 is a software development cost-estimation model of the COCOMO.

$$E = 2.45 \times \text{EAF} \times (\text{KLOC})^P \tag{2}$$

where

E effort (Person-Months);
KLOC thousand lines of code (SLOC/1000);
EAF effort adjustment factor (range: 0.6–1.4);
P project complexity (range: 1.04–1.24).

We calculated the effort by using (2) with the values of SLOC, which is depicted in Table I, and the estimated values are shown in Table II. The arguments EAF and P of (2) have a wide range of values. $E_{\min}$ is the minimum effort when applying the lowest values of EAF and P. The lowest EAF means that most of the cost drivers are very high or extra high, i.e., the team's skills are exceptional. The lowest P means that the project complexity is very low. It denotes that the estimated minimum effort is the value of the ideal case. In contrast, $E_{\max}$ is the maximum effort, and the value is in the worst case.

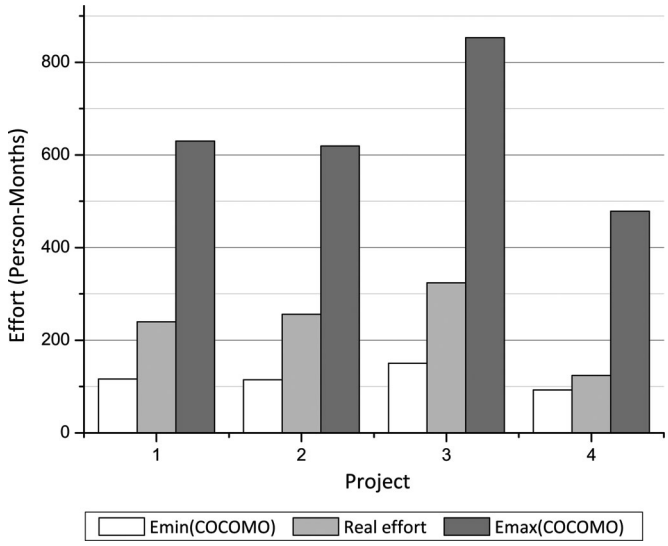

Fig. 19. Comparison between real development effort and estimated effort by COCOMO.

The comparison between real development effort and the estimated effort by the COCOMO is described in Fig. 19. All of the applications have real effort values between minimum and maximum cost. The COCOMO is used to estimate the costs of general software development, and the model has no factor for system M&S, including our collaborative modeling methodology. For a more accurate comparison, further analysis with the proposed collaborative modeling process is worthwhile.

VIII. CONCLUSION

This paper proposed a collaborative modeling process to develop a domain-specific discrete event simulation system. The modeling process covers everything from the conceptual modeling step to the simulation execution step by formally defining the roles and responsibilities of domain engineers, M&S engineers, and platform engineers. This collaborative work between engineers has been required to complete these types of complex domain-specific systems, yet past works have not been supported by any formal collaborative work process or modeling framework in particular. This paper introduces the combination of work process and M&S logic formalism to enable fast, verifiable, and cost-efficient M&S system developments. We achieved quickness and cost-efficiency through parallel system building, and we prevented its possible miscommunications by employing a formal specification method (DEVS in our case) and a common representation method, UML. This idea appears

to be sound when we perform a case study to develop war game models, with which we compare our parallel work process to a possible sequential work practice. There are, however, a number of problems that remain to be explored, and we enumerated the problems with the following questions.

1) Would the collaborative work cover the whole simulation software development and application process?
2) Would the collaborative work be sound in other types of domain-specific systems, i.e., expert systems?

The proposed process only focuses on the development of M&S software. Generally, the software development process consists of various activities, from requirements analysis to simulation analysis, including simulation verification and validation. In addition, there are many kinds of domain-specific systems in various fields, and there are also difficulties in developing the systems because of the expert domain knowledge that they require of the engineer. We hope that the proposed collaborative work may be expanded to cover the whole development process and be applied to the development of general domain-specific systems in addition to simulation systems. We expect that this work process will provide better development practices and collaboration protocols in future M&S software development and applications.

REFERENCES

[1] C. A. Roberts and Y. M. Dessouky, "An overview of object-oriented simulation," *Simulation*, vol. 70, no. 6, pp. 359–368, 1998.

[2] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*. Orlando, FL: Academic, 2000.

[3] J. K. Adams and D. E. Thomas, "The design of mixed hardware/software systems," in *Proc. 33rd Annu. Des. Autom. Conf.*, Las Vegas, NV, Jun. 1996, pp. 515–520.

[4] C. Chittister and Y. Haimes, "Systems integration via software risk management," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 26, no. 5, pp. 521–532, Sep. 1996.

[5] W. Shen, Q. Hao, H. Mak, J. Neelamkavil, H. Xie, and J. Dickinson, "Systems integration and collaboration in construction: A review," in *Proc. 12th Int. Conf. Comput. Support. Cooperat. Work Des.*, Xi'an, China, Apr. 2008, pp. 11–22.

[6] H. Wada, J. Suzuki, and K. Oba, "Modeling turnpike: A model-driven framework for domain-specific software development," in *Proc. Companion 20th Annu. ACM SIGPLAN Conf. Object-Oriented Programm., Syst., Lang., Appl.*, San Diego, CA, Oct. 2005, pp. 128–129.

[7] W. Tracz, L. Coglianese, and P. Young, "A domain-specific software architecture engineering process outline," *SIGSOFT Softw. Eng. Notes*, vol. 18, no. 2, pp. 40–49, 1993.

[8] P. Vitharana, H. Jain, and F. Zahedi, "Strategy-based design of reusable business components," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 4, pp. 460–474, Nov. 2004.

[9] C. Lange, M. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *IEEE Softw.*, vol. 23, no. 2, pp. 40–46, Mar. 2006.

[10] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. New York: Wiley, 2009.

[11] S.-Y. Hong and T. G. Kim, "Embedding UML subset into object-oriented DEVS modeling process," in *Proc. SCSC*, San Jose, CA, Jul. 2004, pp. 161–166.

[12] J. L. Risco-Martn, J. M. de la Cruz, S. Mittal, and B. P. Zeigler, "eUDEVS: Executable UML with DEVS theory of modeling and simulation," *Simulation*, vol. 85, no. 11–12, pp. 750–777, 2009.

[13] C. H. Sung, S.-Y. Hong, and T. G. Kim, "Layered structure to development of OO war game models using DEVS framework," in *Proc. SCSC*, Philadelphia, PA, Jul. 2005, pp. 65–70.

[14] H. Zhu, M. Zhou, and P. Seguin, "Supporting software development with roles," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 6, pp. 1110–1123, Nov. 2006.

[15] J. Barjis, "Collaborative, participative, and interactive modeling and simulation in systems engineering," in *Proc. Spring Simul. Multiconf.*, Orlando, FL, Apr. 2010, pp. 68:1–68:6.

[16] C. H. Sung, I.-C. Moon, and T. G. Kim, "Collaborative work in domain-specific discrete event simulation software development: Fleet anti-air defense simulation software," in *Proc. IEEE Int. Workshop Collaborat. Model. Simul.*, Larissa, Greece, Jun. 2010, pp. 160–165.

[17] T. G. Kim and S. B. Park, "The DEVS formalism: Hierarchical modular systems specification in C++," in *Proc. Eur. Simul. Multiconf.*, New York, U.K., Jun. 1992, pp. 152–156.

[18] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide*, 2nd ed. (Addison-Wesley Object Technology Series). Reading, MA: Addison-Wesley, 2005.

[19] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*, 2nd ed. Boston, MA: Pearson, 2004.

[20] P. Kruchten, "Architectural blueprints: The "4+1" view model of software architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, Nov. 1995.

[21] W. Han and M. Jafari, "Component and agent-based fms modeling and controller synthesis," *IEEE Trans. Syst., Man, Cybern. C*, vol. 33, no. 2, Appl. Rev., pp. 193–206, May 2003.

[22] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. San Francisco, CA: Morgan Kaufmann, 2008.

[23] X. Hu and B. Zeigler, "Model continuity in the design of dynamic distributed real-time systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 35, no. 6, pp. 867–878, Nov. 2005.

[24] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*. San Diego, CA: Academic, 1984.

[25] *DEVS Standardization Group*. [Online]. Available: http://cell-devs.sce.carleton.ca/devsgroup/

[26] M. E. Stropky and D. Laforme, "An automated mechanism for effectively applying domain engineering in reuse activities," in *Proc. Conf. TRI-Ada*, Anaheim, CA, Nov. 1995, pp. 332–340.

[27] W. Tracz, L. Coglianese, and P. Young, "Domain-specific software architecture engineering process guidelines," in *Domain-Specific Software Architecture Engineering Process Guidelines*. Owego, NY: IBM, ADAGE-IBM-92-02, 1993.

[28] E. H. Page and R. Smith, "Introduction to military training simulation: A guide for discrete event simulationists," in *Proc. 30th Conf. Winter Simul.*, Washington, DC, Dec. 1998, pp. 53–60.

[29] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, 4th ed. (Prentice-Hall International Series in Industrial and Systems Engineering). Upper Saddle River, NJ: Pearson Prentice-Hall, 2006.

[30] J. L. de la Vara, M. H. Fortuna, J. Sánchez, C. M. L. Werner, and M. R. S. Borges, "A requirements engineering approach for data modelling of process-aware information systems," in *Proc. 12th Int. Conf. Bus. Inf. Syst.*, Poznan, Poland, Apr. 2009, pp. 133–144.

[31] B. A. Berenbach, "Comparison of uml and text based requirements engineering," in *Proc. Companion 19th Annu. ACM SIGPLAN Conf. Object-Oriented Programm. Syst., Lang., Appl.*, Vancouver, BC, Canada, Oct. 2004, pp. 247–252.

[32] I. Reinhartz-berger, "Conceptual modeling of structure and behavior with uml the top level object-oriented framework (tloof) approach," in *Proc. ER*, Klagenfurt, Austria, Oct. 2005, pp. 1–15.

[33] M. dos Santos Soares and J. Vrancken, "Requirements specification and modeling through SysML," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Montreal, QC, Canada, Oct. 2007, pp. 1735–1740.

[34] M. dos Santos Soares and J. L. M. Vrancken, "Model-driven user requirements specification using SysML," *J. Softw.*, vol. 3, no. 6, pp. 57–68, 2008.

[35] P. A. Fishwick and B. P. Zeigler, "A multimodel methodology for qualitative model engineering," *ACM Trans. Model. Comput. Simul.*, vol. 2, no. 1, pp. 52–81, 1992.

[36] J.-S. Lee, M. C. Zhou, and P.-L. Hsu, "Multiparadigm modeling for hybrid dynamic systems using a petri net framework," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 38, no. 2, pp. 493–498, Mar. 2008.

[37] P. A. Fishwick, "An integrated approach to system modeling using a synthesis of artificial intelligence, software engineering and simulation methodologies," *ACM Trans. Model. Comput. Simul.*, vol. 2, no. 4, pp. 307–330, 1992.

[38] J. S. Hong, H.-S. Song, T. G. Kim, and K. H. Park, "A real-time discrete event system specification formalismfor seamless real-time software development," *Discrete Event Dyn. Syst.*, vol. 7, no. 4, pp. 355–375, 1997.

[39] D. Beazley, B. Ward, and I. Cooke, "The inside story on shared libraries and dynamic loading," *Comput. Sci. Eng.*, vol. 3, no. 5, pp. 90–97, Sep./Oct. 2001.

[40] S.-Y. Hong, J.-H. Kim, and T. G. Kim, "Measurement of RTI performance for tuning parameters to improve federation performance in real-time war game simulation," in *Proc. SCSC*, Philadelphia, PA, Jul. 2005, pp. 71–76.
[41] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell*. Sebastopol, CA: O'Reilly, 2005.
[42] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice-Hall PTR, 2000.

Changho Sung received the B.S. degree in electrical engineering from Pusan National University, Busan, Korea, in 2003. He is currently working toward the Ph.D. degree with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea.

He has experience in defense modeling and simulation (M&S). From 2005 to 2009, he participated in the U.S.–Republic of Korea combined exercises, such as Ulchi Focus Lens, as an M&S Engineer. His research interests include discrete event systems modeling, collaborative M&S, distributed simulation, and hybrid simulation.

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, in 1988.

From 1989 to 1991, he was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Kansas, Lawrence. In 1991, he joined the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea as an Assistant Professor, where he has been a Full Professor since 1998. He is a coauthor of the book *Theory of Modeling and Simulation* (Burlington, MA: Academic, 2000). He has published more than 200 papers about modeling and simulation (M&S) theory and practice in international journals and conference proceedings. He is very active in research and education of defense modeling and simulation in Korea. He has been a Technical Advisor in the defense M&S area at various Korean Government organizations, including the Ministry of Defense, the Defense Agency for Technology and Quality, the Korea Institute for Defense Analysis, and the Agency for Defense Development. He developed DEVSimHLA, which is a tools set for the development of high-level architecture compliant war game simulators, which has been used in the development of three war game simulators for the Navy, Air Force, and Marines in Korea. Since 2004, the three simulators have interoperated with the U.S. war game simulators at the U.S.–Republic of Korea combined exercises, such as Ulchi Focus Lens.

Dr. Kim was the President of The Korea Society for Simulation and the Editor-In-Chief for *Simulation: Transactions for Society for Computer Modeling and Simulation International* (SCS). He is a Fellow of SCS and a Senior Member of Eta Kappa Nu.