Layered Approach to Development of OO War Game Models Using DEVS Framework

Chang Ho Sung*, Su-Youn Hong, and Tag Gon Kim*****
Department of EECS
KAIST
373-1 Kusong-dong, Yusong-gu
Taejeon, Korea 305-701
Tel: +82-42-869-3454
Fax: +82-42-869-8054
chsung@smslab.kaist.ac.kr*; syhong@smslab.kaist.ac.kr; tkim@ee.kaist.ac.kr*****

Keywords: Layered Structure, War Game, Object Oriented Modeling, DEVS

The DEVS framework of discrete event modeling is known to be compatible with the object-oriented (OO) world view. This paper proposes a methodology for the OO development of war game models within the framework. The methodology develops war game models at two layers: a discrete event system (DES) layer and an object model (OM) layer. Within the methodology the DES layer represents abstract behavior of an object using the DEVS formalism; the OM layer represents detailed behavior of the same object using UML. The modeling at the DES layer and the OM layer are mainly responsible for M&S experts and domain experts, respectively. The models development is a co-modeling process in which M&S experts and domain experts work concurrently at different abstraction layers for identical objects in a co-operative manner. The proposed approach would be most effective for modeling systems in which M&S experts do not know domain knowledge in details, such as war game modeling.

1 INTRODUCTION

Discrete event modeling can be considered as a process of abstract knowledge representation about a real-world system. As a model the representation should be executable by a simulation environment to analyze the system with respect to modeling objectives. The process may base on different world views for modelers such as event-oriented, process-oriented, object-oriented, and others. Among them the object-oriented (OO) approach may be most compatible to a real world system from the system-theoretic view point of model representation.

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 DEVS formalism, which represents a discrete event system in the system-theoretic view point, is known to be compatible with the OO world view. Moreover, the formalism supports hierarchical, modular specification of discrete event models which allows us to assemble previously developed component models in a flexible manner.

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

The methodology employs a layered approach in models development in which the upper layer represents abstract behavior of an object and the lower layer represents details of the object. The upper layer is called a discrete event system (DES) layer; the lower layer is called an object model layer. The modeling at the DES layer and the OM layer are mainly responsible for M&S experts and domain experts, respectively. The models development is a co-modeling process in which M&S experts and domain experts work concurrently at different abstraction layers for identical objects in a co-operative manner.

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

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

2 UML AND DEVS FORMALISM: BRIEF REVIEW

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

2.1 UML Modeling

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, and documenting the artifacts of an object-oriented system under development [2][3]. It simplifies the complex process of software design, making a blueprint for construction. This sub-section describes only three diagrams in UML, which are used for modeling of an object at the OM layer. They are use case diagram, class diagram, and sequence diagram.

2.1.1 Use Case Diagram

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

2.1.2 Class Diagram

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

2.1.3 Sequence Diagram

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

2.2 DEVS Formalism

The DEVS formalism specifies discrete event models in a hierarchical and modular form [4]. With this formalism, one can perform modeling more easily by decomposing a large system into smaller component models with coupling specification between them. There are two kinds of models: atomic model and coupled model.

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

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

where

X: a set of input events;
Y: a set of output events;
S: a set of sequential states;
δ_{ext}: $Q \times X \rightarrow S$, an external transition function, where $Q = \{(s,e)|s \in S, 0 \le e \le ta(s)\}$ is the total state set of M;
δ_{int}: $S \rightarrow S$, an internal transition function;
λ: $S \rightarrow Y$, an output function;
ta: $S \rightarrow R^{+}_{0,\infty}$ (non-negative real number), time advance function.

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

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

where

X: a set of input events;
Y: a set of output events;
M: a set of all component models;
$EIC \subseteq DN.X \times \cup M.X$: external input coupling;
$EOC \subseteq \cup M.Y \times DN.Y$: external output coupling;
$IC \subseteq \cup M.Y \times \cup M.X$: internal coupling;
SELECT: $2^M - \emptyset \rightarrow M$: tie-breaking selector.

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

3 PROPOSED LAYERED STRUCTURE FOR OO DES MODELING

This section proposes a layered structure for object-oriented modeling in simulation systems development, especially war game simulation systems. The process of the development may be divided in several layers in which an upper layer may use services provided by a lower layer.

3.1 Motivation: OO Modeling

The proposed framework is based on object-oriented model development which establishes an explicit correspondence between real-world objects and simulation models in one-to-one manner. As is well known OO modeling is originated from the discrete event modeling and simulation language, SIMULA’67. However, OO modeling of general software does not need to represent an object at a discrete event system level which requires additional semantics. UML is one such modeling methodology which is widely used in software engineering society [5]. On the other hand, the DEVS formalism supports precise semantics to specify discrete event models in the OO view point, which is widely used in discrete event M&S society. As shown in Figure 1 both DEVS and UML are object-oriented modeling framework which was inherited from the discrete event simulation language, SIMULA’67. However, the main difference between the two methods is that UML may not be basis on formal semantics of specification of an object as a discrete event model. More specifically, UML does not support specification of time advance for discrete event simulation. Thus, as shown in Figure 1 some additional information is required to completely specify an object as a discrete event model by using UML. On the other hand, any operation on the object can be specified by UML. This is a motivation of the proposed layered structure for OO modeling of discrete event models using both UML and DEVS formalism.

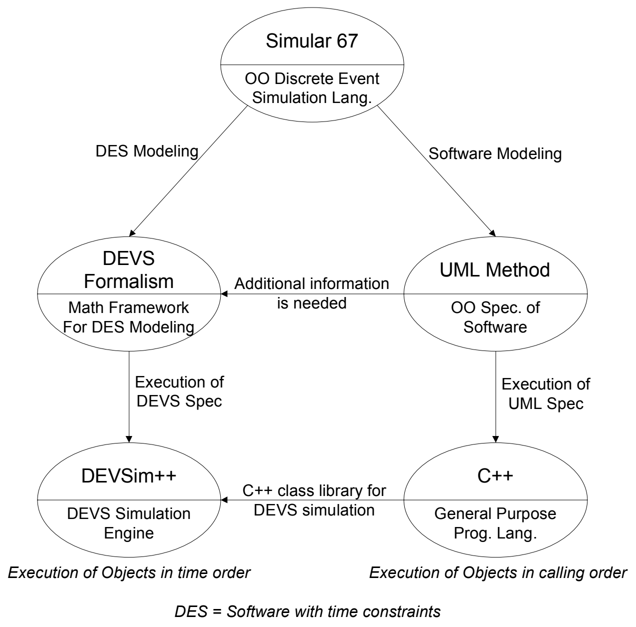
Simular 67
OO Discrete Event Simulation Lang.
DES Modeling
Software Modeling
DEVS Formalism
Math Framework For DES Modeling
Additional information is needed
UML Method
OO Spec. of Software
Execution of DEVS Spec
Execution of UML Spec
DEVSim++
DEVS Simulation Engine
C++ class library for DEVS simulation
C++
General Purpose Prog. Lang.
Execution of Objects in time order
Execution of Objects in calling order
DES = Software with time constraints

Figure 1. OO Modeling: Simulat'67 to UML and DEVS

3.2 Layered Structure

The main purpose of the proposed framework is to partition specifications of a discrete event model of war games in two expert groups: military domain experts and M&S experts. To do so we view the DEVS framework as a superset of the UML methodology in its modeling power. In other word, a DEVS model can represent all information which a UML model can specify, but the other way around is not true. Assuming that domain experts and M&S experts are different we propose a layered approach to OO discrete event modeling. The approach represents a discrete event model in two layers: a discrete event system model (DES) layer and an object model (OM) layer. More specifically, the OM layer specifies details of object operations on variables of an object; the DES layer specifies abstract operations on states of the same object which determine simulation time advance and output events generation. In fact, the proposed layered framework works as a client-server architecture in that the abstract operations of an object exploit the detailed operations of the same object in model representation. Figure 2 shows the layered structure for DEVS framework for development of interoperable war game simulators, which includes the proposed layered approach for DEVS model development.

As shown in Figure 2 domain knowledge for the system to be modeled is represented by mathematical equations, rules or even verbal descriptions which are available in text books, field and/or technical military manuals.

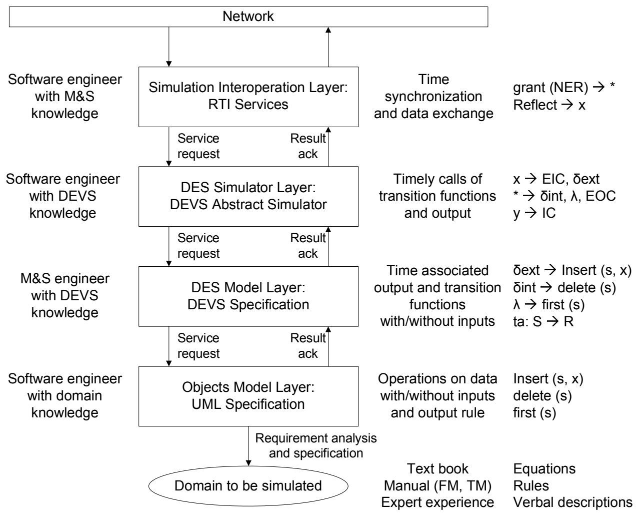
Network
Software engineer with M&S knowledge
Simulation Interoperation Layer: RTI Services
Time synchronization and data exchange
grant (NER) → *
Reflect → x
Service request
Result ack
Software engineer with DEVS knowledge
DES Simulator Layer: DEVS Abstract Simulator
Timely calls of transition functions and output
x → EIC, δext
* → δint, λ, EOC
y → IC
Service request
Result ack
M&S engineer with DEVS knowledge
DES Model Layer: DEVS Specification
Time associated output and transition functions with/without inputs
δext → Insert (s, x)
δint → delete (s)
λ → first (s)
ta: S → R
Service request
Result ack
Software engineer with domain knowledge
Objects Model Layer: UML Specification
Operations on data with/without inputs and output rule
Insert (s, x)
delete (s)
first (s)
Requirement analysis and specification
Domain to be simulated
Text book
Manual (FM, TM)
Expert experience
Equations
Rules
Verbal descriptions

Figure 2. Proposed layered structure of OO DES modeling

Such knowledge can even be obtained by expert experience in military service. Extraction of appropriate equations and rules from the domain knowledge which meet modeling objective can be done by military experts. To be successful they should clearly identify modeling objectives. This is not the case for modeling of hardware and/or processes for which M&S experts without military knowledge can understand. Examples of such modeling include performance modeling of computer/communication systems and of manufacturing systems. Technically, extracted information should be specified as objects and their relationship at the OM layer. UML is employed at this phase of specification. More specifically, simulator architecture should be identified in form of a collection of objects and relationship between objects. Each object should be represented by inputs, outputs and variables and their operations. For example, an object QUEUE may have such operations as insert(s, x), delete(s) and first(s) where s is state and x is an input event. Note, however, that such operations are not complete enough to represent QUEUE as a discrete event model.

A discrete event model for QUEUE can be represented by the DEVS formalism. The formalism has four characteristic functions for discrete event modeling: external transition, internal transition, output, and time advance function. Consider the external transition function of the QUEUE model, which specifies the update rule of state s when an input is received. Intuitively, the function is the same as insert(s, x) defined in the OM layer. Likewise, the internal transition function is same as delete(s); the output function is the same as first(s). Thus, the QUEUE model at the DES layer using the DEVS formalism can directly use services of the QUEUE model provided by the OM layer. To be complete, however, the QUEUE model at the DES layer should has time advance information for discrete event simulation, which can be done by the time advance function of the DEVS formalism.

The proposed layered approach makes it possible for domain experts and M&S experts to work concurrently in cooperative manner. This is a major advantage of the proposed approach in OO war game model development. More specifically domain experts focus just on UML modeling and M&S experts on DEVS modeling. Of course, the process from UML to DEVS modeling may not always be sequential unless all operations needed in DEVS modeling are not ready in UML models. In such a case, M&S experts may request domain experts to define operations which M&S experts need. However, we believe that good cooperation between domain experts and M&S experts from the beginning of system design may minimize such a backward process or even can eliminate it.

3.3 Semantics for DES Model in two layers

We formalize the proposed layered framework using set theoretic representation. To show relation between two layers in formal manner we employ the Finite State Machine formalism as semantics for UML modeling. Then, a simulation model is represented by two objects and relationship between the two: (1) one represented by finite state machine (FSM) at the OM layer; (2) the other by DEVS formalism at the DES layer; (3) relation between elements in DEVS formalism and those in FSM specification. The formalism may give sound semantics for the proposed layered approach. The relations are as follows.

A DEVS atomic model includes three sets and four functions, and FSM model includes three sets and two functions. Figure.3 shows the mapped relation between DEVS and FSM. Input events, output events, and the state sets of DEVS are mapped to those of FSM with same relation. However, there are two transition functions in DEVS formalism, while only one transition function in FSM.

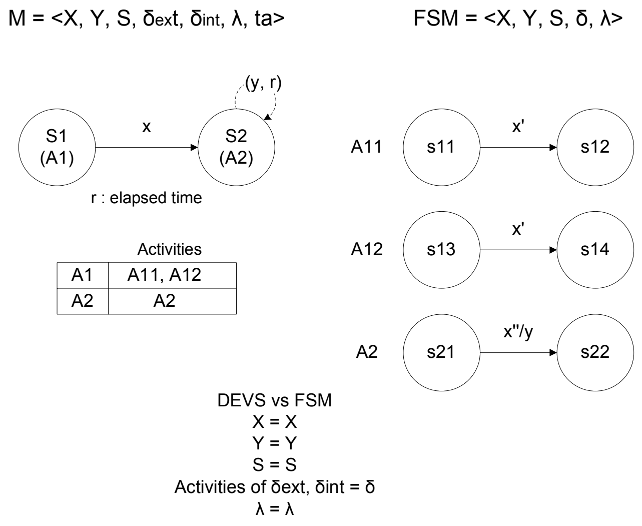

Figure 3. Relation between DEVS and UML

In this methodology, two transition functions are mapped to activities, and each activity means a transition of FSM. When the current state is S1 and input event is x, activity A1 should be operated. In the current state S1, activity is defined by transition from s11 to s12 of FSM. If there are several activities, the same number of transitions of FSM exists. Therefore, DEVS formalism express the activities occurred by two transition functions as FSM state transition.

4 CO-MODELING METHODOLOGY USING LAYERED STRUCTURE

To exploit the layered approach efficiently we propose a co-modeling methodology in discrete event models development. The methodology is analogous to HW/SW co-design in VLSI systems design.

4.1 Modeling Process using UML and DEVS

HW/SW co-design means the meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design. To do so, system specification is partitioned in hardware and software parts for concurrent job. After that, each part is implemented and then integrated for co-simulation [6] [7].

Co-modeling methodology is analogous to HW/SW co-design and the process is shown by Figure 4. At first, we design the simulator architecture from requirements and specification for a system to be simulated. If M&S experts do not need detailed knowledge on an object in atomic DEVS modeling, they define three sets and four functions according to DEVS formalism. On the hand, if atomic DEVS modeling needs detail knowledge on the object, military domain experts defines such knowledge in forms of operations on the object using UML. Then, M&S experts employ the operations as services. Specifications of DEVS and UML models are implemented by DEVSim++ and C++, respectively. Figure 4 shows the co-modeling methodology.

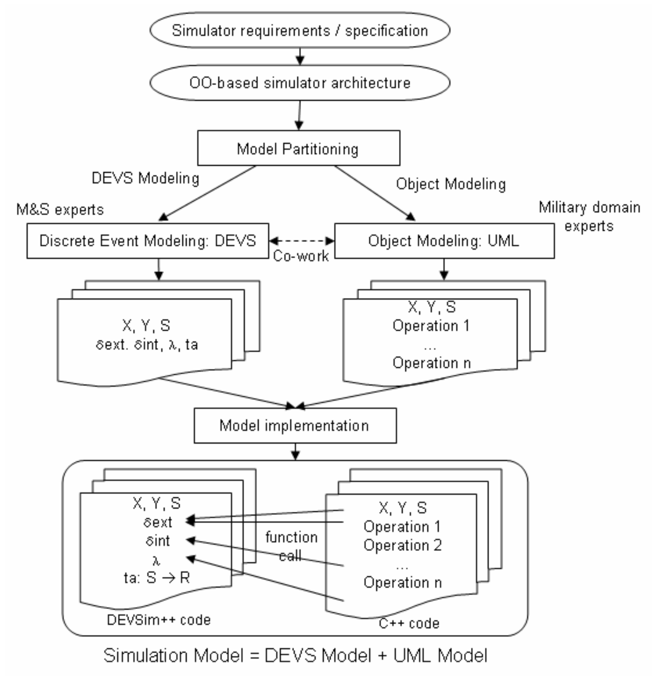

Figure 4. Co-modeling methodology using UML and DEVS

4.2 Advantage

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

5 Example: A Simple War Game Model

The proposed layered approach is applied to a simple war game model. Assuming that there are many ships in a navy war game model and that the ships are moving on the sea, the positions are ordered by a gamer. The scenario of the war game simulation is following:

1. A gamer gives a command to move a ship to some point as a destination.

2. The ship calculates the next position per interval time until it arrives at the destination point.

If M&S experts do not know operational rules for a ship to move they could not specify a discrete event simulation model for the ship. In such a case, the rules should be specified by navy domain experts at the OM layer modeling, which the M&S experts can use at the DES layer modeling. Let us explain the model specifications of the ship object at the OM layer and the DES layer.

5.1 OM Layer

In the OM layer a ship object can be modeled by attributes and operations defined on the object. The modeling at the layer is the same as one used in a general class definition in OO software development.

For maneuver of a ship, the ship has velocity, heading, and position as attributes. Similarly, operations on the ship include "move" the definition of which needs additional operations. Those operations are to calculate heading, velocity, and next position. Thus, at this OM layer military domain experts should offer algorithms for the operations. Figure 5 represents a class diagram of the ship object at the OM layer in relation to maneuver.

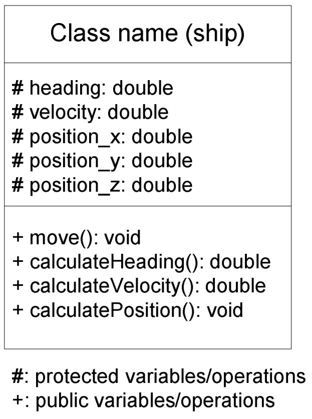
Class name (ship)
# heading: double
# velocity: double
# position_x: double
# position_y: double
# position_z: double
+ move(): void
+ calculateHeading(): double
+ calculateVelocity(): double
+ calculatePosition(): void
#: protected variables/operations
+: public variables/operations

Figure 5. Class diagram of the system

5.2 DES Model Layer

In the DES layer a ship object should be specified in form of timed state transition. In other word, the object has specification of a state transition with an associated occurrence time.

M&S experts specify such information on the ship object using the DEVS formalism as follows:

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

$X = \{\text{"move\_order"}\}$

$Y = \{\}$

$S = \{\text{WAIT, MOVE}\}$

$\delta_{ext}:\ \text{WAIT} \times \text{"move\_order"} \rightarrow \text{MOVE}$

$\delta_{int}: \text{MOVE} \rightarrow \text{MOVE}$

$ta(\text{WAIT}) = \infty$

$ta(\text{MOVE}) = t1$ (interval time for move operation)

As shown in the specification a ship is waiting for an input command of "move_order" at the initial state of "WAIT." Time advance at the state is defined as an infinite in the DEVS formalism, meaning that nothing can happen at the state unless an input event occurs.

When the state of the ship is 'WAIT' and an input event of "move_order" occurs, the state transitions to the 'MOVE' state and the ship begins moving for a destination point. In this state, the time advance is some finite time because the position of the ship is changed at each unit time. In the 'MOVE' state, a sequence of operations is calculated in order to compute the next position of the ship after a unit time. Those operations might be defined in the OM layer. Figure 6 shows the ship object model at both the DES layer and the OM layer. Note in the Figure that a state transition at the MOVE state of the DES layer three operations – calculations of heading, velocity and next position - defined at the OM layer.

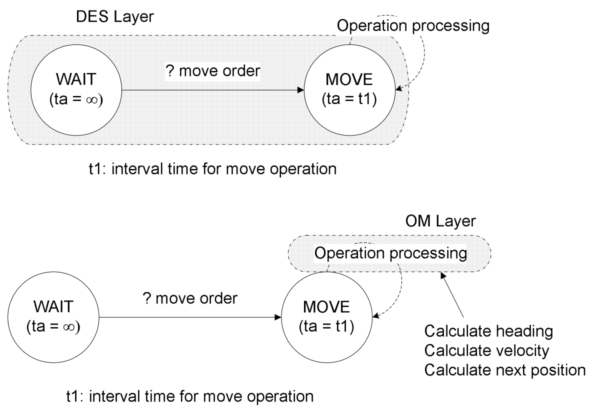

Figure 6. DEVS graph and classification of DES layer and OM layer

The relation between the OM layer and the DES layer is shown by Figure 7. This figure also represents the modeling procedure from domain requirements to the DES layer.

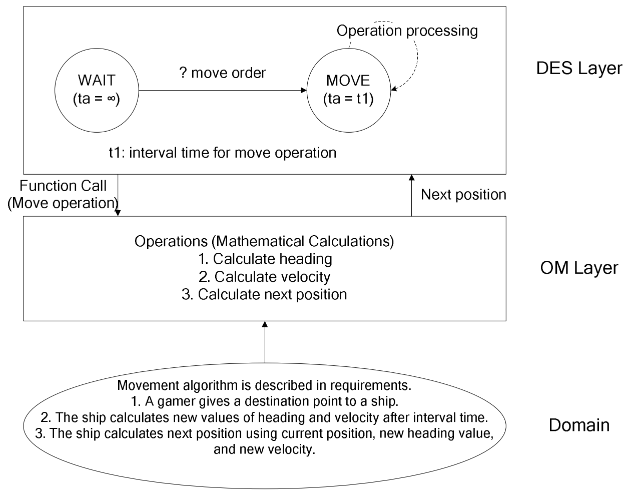
Operation processing
WAIT (ta = ∞)
? move order
MOVE (ta = t1)
t1: interval time for move operation
DES Layer
Function Call (Move operation)
Next position
Operations (Mathematical Calculations)
1. Calculate heading
2. Calculate velocity
3. Calculate next position
OM Layer
Movement algorithm is described in requirements.
1. A gamer gives a destination point to a ship.
2. The ship calculates new values of heading and velocity after interval time.
3. The ship calculates next position using current position, new heading value, and new velocity.
Domain

Figure 7. Example of layered structure of war game model

6 CONCLUSION

This paper proposes a methodology for the OO development of war game models within the framework. The methodology develops war game models at two layers: a discrete event system (DES) layer and an object model (OM) layer. The main objective of the approach was to partition model specifications in two layers by which co-work between military domain experts and M&S experts would be possible. In the OM layer, domain experts define attributes and operations of an objects using UML. In the DES layer, M&S experts define time associated output and transition function using DEVS formalism by use of operations defined in the OM. A co-modeling methodology using this approach was also proposed in this paper. A case study demonstrated effectiveness of the proposed approach. This approach may be applied to more complex war game modeling projects.

REFERENCES

[1] Su-Youn Hong and Tag Gon Kim, "Embedding UML Subset into Object-oriented DEVS Modeling Process," in *Proceedings of the Summer Computer Simulation Conference*, San Jose, California, USA, July 2004, pp. 161-166.
[2] Booch, Rumbaugh, and Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesely, Reading, Massachusetts, 1998.
[3] Rumbaugh, *Unified Modeling Language Reference Manual*, Addison-Wesely, Reading, Massachusetts, 1999.
[4] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon kim. *Theory of Modeling and Simulation,* ACADEMIC PRESS, 2000.
[5] Mark Priestley, *Practical Object-Oriented Design with UML*, The McGraw-Hill Companies, 1996.
[6] Jay K. Adams and Donald E. Thomas, "The Design of Mixed hardware/software systems," in *Proceedings of the 33rd annual conference on Design automation*, Las Vegas, Nevada, USA, June 1996, pp. 515-520.
[7] David W. Franke and Martin K. Purvis, "Hardware/Software CoDesign: A Perspective," in *Proceedings of the 13t international conference on Software engineering*, Austin, Texas, USA, 1991, pp. 344-352.