Design and Implementation of Event-based DEVS Execution Environment for Faster Execution of Iterative Simulation

Se Jung Kwon and Tag Gon Kim
Department of Electrical Engineering
Korea Advanced Institute of Science of Technology
Daejeon, KOREA
(E-mail : sjkwon@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr)

Keywords: Discrete event simulation, DEVS, Discrete event simulation speedup, Event scheduling world view

Abstract

Discrete EVent systems Specification (DEVS) is one of the most frequently utilized system specifications to model discrete event systems in the real world. To analyze system behavior, simulation results are collected by executing the DEVS model. Although a large number of results are essential for the accuracy of analysis, the existing execution environment we developed is not focused on faster iterative simulation for collecting plenty of data.

This paper proposes a new DEVS execution environment applied to event scheduling worldview for faster iterative execution. It proposes an event-oriented DEVS model for simple event scheduling and suggests mediation processes to simulate the DEVS-Compliant model with improved performance, similar to simple event scheduling.

This paper discusses a performance test to verify the execution environment. The experimental results show that the proposed environment has significantly shorter execution time. The expectation is that this improved simulation engine will be applicable to iterative DEVS execution for collecting simulation results. It increases the accuracy of system analysis.

1. INTRODUCTION

Discrete Event System (DES) has been widely used to analyze system behavior in the military domain, industrial domain, social decisions, and more. DEVS is one of the most frequently utilized system specifications to model discrete event systems in the real world.

To analyze the behavior of the discrete event system, many simulation results should be collected by executing the DEVS model using a DEVS executing environment. The executing environment consists of a simulation engine with a scheduling algorithm and implemented DEVS model executed by simulation engine. The simulation engine executes the imminent model of implemented DEVS model with a scheduling algorithm and moves output messages among interconnected components.

In the System Modeling and Simulation laboratory in KAIST, we developed DEVSim++ as a DEVS execution environment [1]. DEVSim++ realizes the DEVS formalism for modeling and associated abstract simulator concepts for simulation, all in C++. The implemented models of DEVSim++ have a hierarchical structure and the same interface as DEVS. In other words, the models are DEVS-Compliant models, which satisfy the interface of DEVS formalism. The DEVSim++ schedules the DEVS-Compliant models and provides logging system for debugging. Simulators can be developed with more convenient modeling, debugging and verification.

The environment has been applied in the Korean military domain, and many simulation models have been developed [3][4][5]. In our empirical studies, however, the simulators produced simulation results with lower performance because the hierarchical scheduling and the modular property of DEVS caused simulation overhead. The execution time of the simulators was relatively short time. However, while executing the simulator with several random variables many times, the short time became a considerable amount of time.

Therefore, improvement in iterative execution performance has become an important issue for our M&S toolkit. The primitive method is improvement in computing power or parallel execution [6][7]. Apart from these methods, this paper focuses on improvement of the simulation algorithm for faster iterative execution.

Our salient attempt to improve the performance is reducing the overhead of hierarchical structure and modular property of DEVS. We developed an event-based engine and event-oriented DEVS model in the view of the event-scheduling world [8][9] (models are not scheduling units, but events of the discrete event model are prescheduled). Whereas the DEVS simulation engine asks the t_N value to model due to modularity, the pre-scheduled information of the event scheduling worldview is globally stored in an event-list.

In the event-based engine, the seeking imminent model is conducted by a sorted event-list, and message passing is conducted by directly event calling. As a result, the simulation speed can be improved more than with hierarchical simulation. For event scheduling, the DEVS should be implemented as an Event Oriented DEVS (EO-DEVS) model, which is a set of event functions, unlike the DEVS-Compliant model. The EO-DEVS models, however,

have a major weakness because the models have little relation to the virtues of DEVS. Hence, this paper proposes mediation processes to simulate the DEVS-Compliant models, similar to the speed of executing EO-DEVS models. The proposed process described in Figure 1 becomes possible from the event-based engine. Supposing modelers develop models using DEVSim++, our proposed environment will be used in the iterative execution step because the environment executes the same implemented model of DEVSim++. Users can choose the DEVS execution environments in accordance with their purposes.

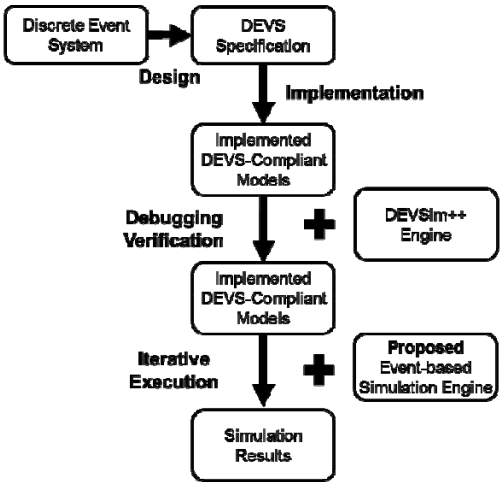

Figure 1. The proposed process of M&S

This paper is organized as follows. Section 2 presents DEVS scheduling and several of the related works. Section 2 also presents event-scheduling world view. Section 3 explains how the DEVS is executed on the event scheduling world view, and Section 4 proposes the pre-process and the mediation process for DEVS-Compliant models. From the processes, the event-based engine can execute same DEVS-Compliant model of existing simulation environment. Section 5 illustrates two case studies for correctness test and performance test. Finally, Section 6 concludes this paper.

2. RELATED WORK

We firstly introduce background knowledge of DEVS execution and surveyed papers about improvement of DEVS execution. Secondly, we briefly introduce event scheduling.

2.1. DEVS Execution

The classic DEVS execution is performed on a hierarchical structure like the structure of DEVS [8]. Each Atomic model is mapped to Simulator, and each Coupled model is mapped to Coordinator. The top coordinator of hierarchy is called Root Coordinator, which decides what model is executed next.

Due to the modular property of DEVS, there is no global information in the simulation process. Hence, DEVS simulation is performed by moving four messages: (*, t), (done, t_N), (x, t), and (y, t). (*, t) is delivered to the next scheduled component at time t, and (done, t_N) is sent to their parent from components to notice the next time that the model should be executed. (x, t) and (y, t) are related to message passing between atomic models. As each message is moved among coordinators and simulators on the hierarchical structure, the DEVS model is executed.

In this simulation process, there are two kinds of overhead [10]. One is an overhead of scheduling because the time information about the imminent model is collected through the hierarchical structure. The other is overhead of message passing between atomic models because the messages are conveyed by hierarchical structure. Although DEVS models with no hierarchy are not always faster than hierarchical structure [12], it is a general tendency that the classic DEVS execution is slower as the hierarchical structure gets deeper. Consequently, there have been efforts to improve the DEVS execution algorithm for various DEVS formalism.

For the classic DEVS, there have been two efforts to improve the simulation speed. One is the work of Kim et al. They reduced execution time by restructuring the hierarchy through a static optimization approach based on profile analysis [12]. The other is the work of Lee et al. They proposed a compiled technique, which combines component models into a single model. By composition at compile time, they reduced message passing overhead [10].

For parallel and dynamic structure DEVS, A. Muzy and J. J. Nutaro proposed an improved algorithm focused on active components using an event-list [13]. Because the priority of the model is unnecessary, unlike classic DEVS, all the atomic models can be scheduled in the event-list, regardless of hierarchical structure. However, message passing is conducted by hierarchical structure.

There also have been various attempts for Cell-DEVS. Cell-DEVS is extended DEVS formalism for cellular space [8]. The forest fire spreading model [14] is a representative application of Cell-DEVS. For the Cell-DEVS, Wainer and Giambiasi proposed improved simulation by flattening of the hierarchical model [15]. Hu and Zeigler proposed a simulation algorithm by constructing a complete tree to utilize spatially distributed causal events [16].

2.2. Event Scheduling World View

Event scheduling world view is one of the most efficient discrete event simulation strategies because of its simplicity. Event scheduling works with prescheduling of all events in an event-list [8]. In other words, because the event-list is always arranged in order of time, the scheduling is performed at insertion time of events in advance. In this view, DES is specified to the EO model with an event as a unit in contrast with the object-oriented view. The EO model consists of functions mapped to events and global variables modified by functions. The simulation engine

executes a function mapped to an event by extracting from the event-list, and the executed functions insert a newly generated event to the event-list for scheduling:

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

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

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

Algorithm 1. Simple algorithm of event scheduling

3. EVENT-ORIENTED MODEL FOR DEVS

Above all, we propose the event-oriented model for event-scheduling DEVS execution. While DEVS is specified with components as a unit, the EO-Model must be specified with events as a simulation unit. In addition, unlike DEVS, the EO model has global variables with no hierarchy and no modularity. The functions of the EO model schedule the next events. That means that the model is not separated from simulation engine.

To execute DEVS using the algorithm described in Section 2, DEVS should be translated to the set of events. In DES, the state is changed when the system has no input until the specified time or an inputted message occurs. Two events can be declared according to the state transitions*.

Firstly, an internal transition event occurs when there is no input until the specified time. In the case of DEVS, the specified time is declared as t_N, which all the model have. The coordinators select imminent t_N and execute functions of corresponding model: δ_{int}, λ and ta functions. From the return values of λ and ta functions, message passing and scheduling are conducted. In the view of event scheduling, all the processes are united into one internal function, mapped from an internal transition event as in Figure 2.

Figure 2. An internal transition function of EO-DEVS Model

* DEVS can also be declared as only one event because the external transition occur by internal transition at the same time. Nonetheless, this paper declares two events according to the state transition.

When the internal transition event occurs, the mapped internal function is executed and internal state transition and output generation are conducted. At this time, two new events are created. One is an external transition event destined for other model. An output message is a trigger that executes other models. The internal transition function creates an external transition event and is inserted into the event-list. The output message is attached to the external transition event instead of message passing process; i.e., the external events signify the message itself. The other is an internal transition event for the next execution of itself.

Secondly, an external transition event occurs when an inputted message occurs. In the case of DEVS, the δ_{ext} function of DEVS is called by the output message of the other model. Because the calling of δ_{ext} means that the state of system is modified and has to newly reschedule, the ta function of DEVS is recalled, and the t_N of the system is modified. Likewise, in the view of event scheduling, external state transition and regenerating the next execution time become one external function mapped from the external transition event, as in Figure 3.

Figure 3. An external transition function of EO-DEVS Model

When the external transition event occurs, external state transition is conducted by the mapped external function, and the next internal event newly occurs from a generated time value that specifies the next execution.

The EO-DEVS model can show dramatically improved speed with event-based simulation due to simplicity. Nevertheless the EO-DEVS model has a weak relation to the property of DEVS. When we implement the EO-DEVS model, we implement two functions from 4 DEVS functions and add an event scheduling routine. The implementation process often has bugs and errors. In addition, the EO-DEVS model has no hierarchy and no modularity. That is the opposite against the positive property of DEVS. If someone wants an extremely fast simulation, the EO-DEVS model is useful. However, the advantages of DEVS are generally good reasons that the DEVS has widely used. Hence, a new requirement is applying event scheduling to the DEVS-Compliant model. The next section explains the event-based engine reflecting the requirement.

4. EVENT-BASED ENGINE FOR DEVS-COMPLIANT MODEL

To apply event scheduling to the existing DEVS-Compliant model, we propose processes between the DEVS-Compliant model and the event-based engine. In the

view of the simulation engine, the differences between the DEVS-Compliant model and EO-DEVS model are as follows:

i) DEVS-Compliant models have a hierarchical structure and I/O interface. Many existing implemented models consist of coupled models and atomic models. In contrast, EO-DEVS models consist of only event functions, and there is no interface for the message I/O.

ii) DEVS-Compliant models consist of 4 functions like DEVS. In contrast, EO-DEVS models consist of only 2 functions mapped to events.

iii) DEVS-Compliant models are passive models and EO-DEVS models are active models. In the DEVS-Compliant model, the functions of the DEVS model only receive the calling of the simulation engine and the called function returns results—i.e., output message or next time value. In contrast, the functions of EO-DEVS models schedule an event into the event-list of the simulation engine directly.

Therefore, the proposed event-based engine supports two processes. One is the pre-process for i) as described in Section 4.1. In the pre-process, the DEVS-Compliant models are translated to executable model by event scheduler. The other is the mediation for ii) and iii) as described in Section 4.2. The mediation translates the callings of event functions to callings of DEVS-Compliant functions and schedules events form the return values of DEVS. The event-based engine with the processes is called "E-DEVSim++".Section 4.3 proposes an event-based DEVS execution algorithm.

4.1. Pre-process

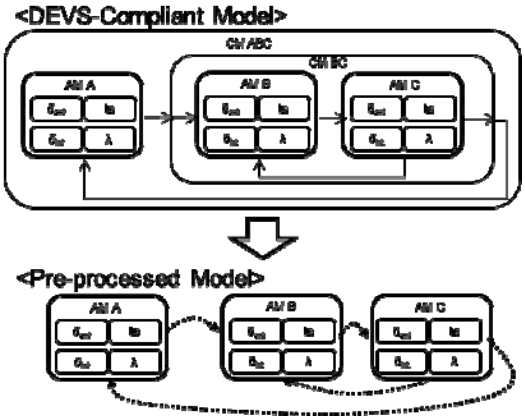

Figure 4. An example of the pre-process

At first, the proposed engine decomposes the coupled model from the DEVS-Compliant model because the EO-DEVS models have no hierarchical structure and I/O interface. By analyzing the hierarchical structure, the engine eliminates coupled models except for atomic models. In addition to this process, by analyzing the message routing path from a departure atomic model to a destination atomic model, the engine leaves one-depth message paths a port. When the output message is returned from δ_{ext} of DEVS, the engine finds the destination of created events from the information from the reduced path.

From the example in Figure 4, the DEVS-Compliant models, which have a hierarchical structure and I/O interface, are translated to a pre-processed model with no hierarchy and direct message paths.

4.2. Mediation Process

To resolve the differences, the mediation process translates the function calls and the scheduling parts adequately. When the simulation engine calls the functions mapped to event, proper functions of the DEVS-Compliant model is called through the mediation. From the return values of DEVS functions, external transition events and internal transition events are scheduled through the mediation process. In detail, the output messages from the δ_{ext} function are translated to external transition events, and the t_N value from ta function is translated to an internal transition event.

We can infer that new overhead is occurred from the mediation process, though the overhead is the price that pays for using existing DEVS model. Modelers can implement DEVS-Compliant model with comparative ease and execute the model with the event-based engine through the mediation process.

4.3. Event-based algorithm for DEVS-Compliant models

This section explains the detailed algorithm for implementing the 'E-DEVSim++'. Algorithm 2 describes the simulation initialization including the pre-process. At first, the simulation engine extracts atomic models from the DEVS-Compliant model. Next, the engine finds the destination ports and atomic models for each port of every model. Each atomic model has a port-mapping table, which has a coupling relation from the output port to the destination's input port.

```
OutmostCoupled // the top coupled model of DEVSim++
M              // set of pre-processed atomic models
dest       // indicates a desitnation model and port.
Simulation_INIT()
  M = Flattening( OutmostCoupled )
  For each m of M
     For each output port p of m
        dest = FindDestinationPort( p )
        m.port_table.add( p, dest )
     end For
  end For
```

Algorithm 2. Initialization including the pre-process

Because we discuss the execution of classic DEVS, the proposed engine should deal with the select function. All the coupled models of DEVS have the select function to select

an imminent model when the t_N of models are same. Therefore, the role of select function in the proposed engine is performed at event insertion time. Hence, the events should be identified and the priority of events should be set up during the pre-process.

Algorithm 3 describes the main function of simulation. The simulation engine has two kinds of data structures for storing events. One is a heap-based event-list for internal transition events. The other is an event queue for external transition events. Although external transitions are even events in the view of state transition, the external transitions occur by output function at the time of the executed internal event. Hence, the engine does not have to schedule the external events. After one internal event is executed, the engine handles only external events sequentially.

```
T_global // current simulation time
INT_EventList(time, target_model)
     // List of sorted event, Heap-based
EXT_EventQueue(dest, output)
     // List of unsorted event, Queue-based

Simulation_Run()
  while ( isEmpty( INT_EventList ) )
    first = top.INT_EventList(time, target-model)
    delete the top.INT_EventList from Event-List
    T_global = first.time
    target = first.target_model

    message_set = target.output()
    for each msg of message_set
       Schedule_EXT_Event(msg.port_name, msg.output)
    target.int_trans()
    new_tn = taget.ta()
    Schedule_INT_Event( new_tn, target )

    while( isEmpty( EXT_EventQueue ) )
      first = front.EXT_EventQueue
      delete the front.EXT_EventQueue
      target = first.target_model
      target.ext_trans( first.output )
      new_tn = target.ta()
      Schedule_INT_Event( new_tn, target )
    End while
  End while
```

Algorithm 3. Main simulation part

Algorithm 4 describes scheduling functions called by the main simulation function. The functions are called by event-oriented models in original event scheduling. In contrast, DEVS-Compliant models are the passive models. So the simulation run function calls these functions.

Each function creates an event from arguments and inserts or pushes back to proper data structures. When the Schedule_INT_Event function is called, the previous event should be deleted because when the model's state is changed by external transition and has a new t_N value, the prescheduled event indicating the target_model is still stored in the event-list.

When the Schedule_EXT_Event is called, the argument is an output port and output messages. Hence, the function finds destination information from port_table and creates a new event with the information.

```
Schedule_INT_Event( time, target_model )
  delete previous event of target_model
  create an event with the pair( time, target_model )
  insert the event to INT_EventList

Schedule_EXT_Event( port_name, output)
  dest = port_table.find( port_name )
  create new event with the pair( dest, output )
  pushback the event to EXT_EventQueue
```

Algorithm 4. Scheduling function

5. CASE STUDY

In this section, we perform two experiments. One is a correctness test, and the other is a performance test. In the correctness test, we confirm that the sequence of E-DEVSim++ is the same as DEVSim++, using the Single Server Queuing Model. In the performance test, we show the performance improvement compared with DEVSim++.

5.1. Correctness Test

For the correctness test, we implement the single server queuing (SSQ) model for DEVSim++. The model design of SSQ is depicted in Figure 5. The SSQ model consists of a generator, a buffer, four processors, and a transducer. A generated signal from the generator queues up in the buffer until the processor can handle this signal. Through the processor, processed signals arrive in the transducer. When the goal number of customers is reached, the transducer sends a stop message to the generator.

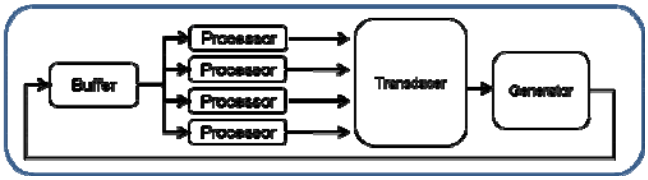

Figure 5. Single Server Queuing Model

We execute two simulation environments using the same DEVSim++ model. Under the same parameters and the same random seed, we log generation time, service-start time, and service-end time of all the signals and compare time-stamps between two simulation engines.

We experiment and compare two logs until the number of customers reaches 1,000,000. For the sake of easier viewing, Figure 6 depicts only the fore twenty signals for graphical representation of comparison between two simulation engines. '×' means a time stamp of E-DEVSim++ and '+' means a time stamp of DEVSim++.

The horizontal axis is customer number generated in order and the vertical axis is time. We collected logs from each simulation engine and drew '×' or '+'. In this graph, we can

confirm that there are only '✳' stamps because '×' and '+' stamps are perfectly overlapped[†]. We also compared logs of two engines, and they were perfectly equal. From the comparison, we can confirm that two engines show the same results using the SSQ model.

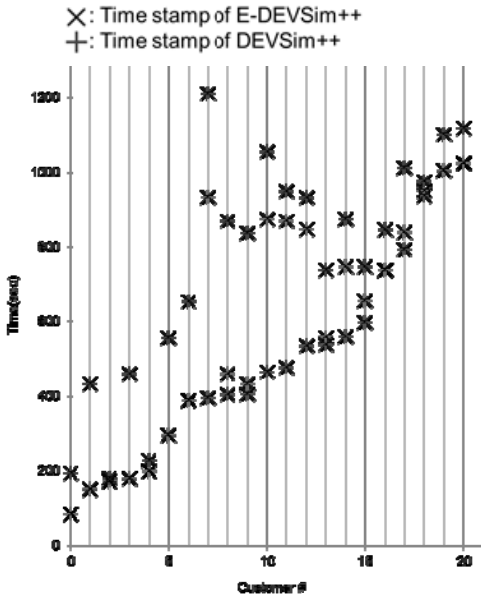

Figure 6. Graphical representation of comparison when customer number is below 20

5.2. Performance Test

We completed a performance test using a simple SSQ model. We then applied our proposed engine to the developed model for real-world domain, which is Anti-torpedo simulation model. All the models are existing DEVSim++ models with no revision. The experimental environment is listed in Table 1.

Table 1. Experimental environment

CPU	Intel i7 860 (2.6GHz, 4 core, No hyper-threading)
RAM	6.00GB
OS	Windows 7
Language	Microsoft Visual C++ 10.0

5.2.1. Single Server Queuing Model

We experimented with a performance test using the SSQ model as with the correctness test. Because performance of DEVSim++ is generally worse as the depth of the model gets deeper, we used flattened SSQ models (one outmost coupled model and atomic models) for pair comparison. The 3 test subjects are DEVSim++, E-DEVSim++, and EO-DEVS models with simple event-based scheduling.

We executed the model on each execution environment and timed the execution time. To experiment in scalability, we executed many SSQ models simultaneously while increasing the number of models from 1 to 20,000. The results are depicted in Figure 7.

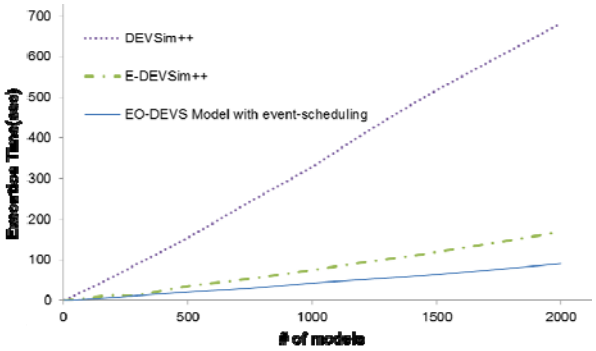

Figure 7. The experimental results of SSQ model (10,000 customers for each SSQ model)

The graph shows how the event-based engine enables performance improvement. E-DEVSim++ is about 4 times faster than DEVSim++, and the EO-DEVS model is about 8 times faster than DEVSim++. Naturally, the fastest EO-DEVS model with event-scheduling is a better choice for the iterative execution. Considering the weakness of the EO-DEVS model, however, E-DEVSim++ is the more preferable choice. The time gap between E-DEVSim++ and the EO-DEVS model is due to the overhead of the mediation process. Nonetheless the time gap is less as compared to the execution time of DEVSim++.

5.2.2. Anti-torpedo Warfare Simulator

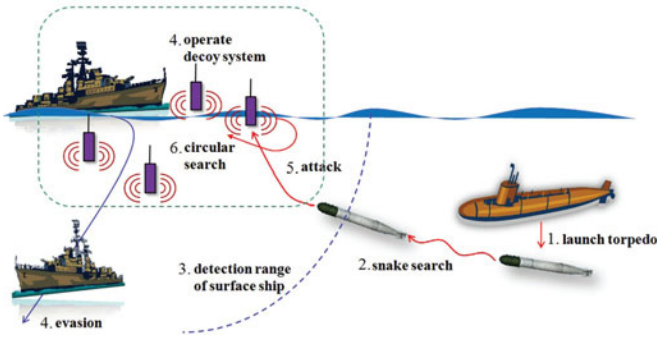

Figure 8. Brief scenario of anti-torpedo simulation [3]

We experimented with the anti-torpedo warfare simulation model to apply our proposed engine to the developed model for the real-field. The brief scenario is illustrated in Figure 8. The attacking platform is a submarine, and the target platform is a surface ship. The

[†] All the customers had three '✳' stamps, but on the graph, some customers only have two stamps. In these cases, generated time and service-start time are almost equal.

surface ship uses decoys for counter-measures. The surface-ship launches decoys according to stored strategies against the torpedo's possible paths. While the torpedo traces decoys, the surface-ship can evade the opponents.

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

In this paper, we experimented with the same model and parameters using DEVSim++ to compare to the results and execution time of DEVSim++. There are 600 scenarios from 4 parameters, detection range of surface ship, pattern of decoy system, operating time of decoy, and speed of mobile decoy. We tested 4 scenarios while changing the pattern of decoy system and not changing other parameters. The parameters are listed in Table 2 and Table 3.

Table 2. List of parameters

Parameter names	Parameter Value
Detection range	2500m
Operating time	130 seconds
Speed of mobile decoy	12 knts
Pattern	1, 2, 3 and 4

Table 3. Four patterns of the decoy operating system

#	Implications
1	4 static decoys (2 decoys at the front of surface ship and the others at the rear)
2	4 mobile decoys (2 decoys at the front of surface ship and the others at the rear)
3	2 static decoys at the front of surface-ship and 2 mobile decoys at rear
4	2 mobile decoys at the front of surface-ship and 2 static decoys at rear

All the scenarios are executed 10,000 times. The execution time for generating the above data is described in Figure 9.

The graph in Figure 9 shows considerable performance improvement. According to the patterns, our new simulation engine is 15–19 times faster. Seo et al. actually simulated the model 100 times and limited the changeable parameters [3] because of time limitation. The E-DEVSim++ provides more diverse analysis scenarios while increasing the number of execution times.

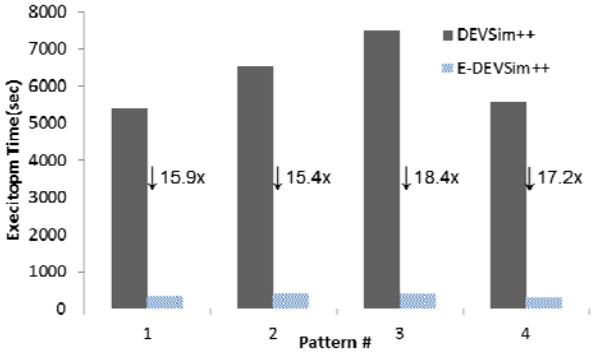

Figure 9. Execution time of the experiment

6. CONCLUSION

This paper proposed a new DEVS execution environment with event-scheduling for faster iterative execution. It applied an event scheduling worldview to DEVS execution and proposed an EO-DEVS model executed by event scheduler. However, the weakness of the EO-DEVS model is critical while our existing DEVSim++ models are useful for modeling and debugging.

Hence, the requirement of using DEVS-Compliant models with no change is arisen. The proposed simulation environment in this paper, 'E-DEVSim++', consists of existing DEVS-Compliant models and an event-based simulation engine with the pre-process and the mediation process, which enable the event-based engine to execute DEVS-Compliant models. The proposed engine is 5–20 times faster than DEVSim++ in the case study.

Finally, we can improve the M&S process of DEVSim++ toolkit. Users can use DEVSim++ in the development and testing process because the DEVSim++ is suitable for modeling and debugging. After that, using E-DEVSim++, they can execute the developed model and analyze the simulation results with more data in less time. To maximize the merits of event scheduling, the E-DEVSim++ needs to be as fast as the EO-DEVS model. Therefore, minimizing the mediation overhead is instrumental in future work.

Acknowledgement

This research is supported by Ministry of Culture, Sports and Tourism(MCST) and Korea Creative Content Agency(KOCCA) in the Culture Technology(CT) Research & Development Program 2009.

References

[1] Tag Gon Kim, Chang Ho Sung, Su-Youn Hong, Jeong Hee Hong, Chang Beom Choi, Jeong Hoon Kim, Kyung Min Seo, and Jang Won Bae, 2011, "DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 8, No. 3, (July): 129 – 142.

[2] Tag Gon Kim and Sung Bong Park, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++," 1992 European Simulation Multiconference, York, United Kingdom, pp. 152 - 156, June, 1992.
[3] Kyung-Min Seo, Hae Sang Song, Se Jung Kwon and Tag Gon Kim, "Measurement of Effectiveness for an Anti-torpedo Combat System Using a Discrete Event Systems Specification-based Underwater Warfare Simulator," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 8, No. 3, pp. 157 - 171, July., 2011.
[4] Jeong Hoon Kim, Chang Beom Choi, and Tag Gon Kim, "Battle Experiments of Naval Air Defense with Discrete Event System-based Mission-level Modeling and Simulations," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Vol. 8, No. 3, pp. 173 - 187, July., 2011.
[5] Jae Hyun Kim, Su Youn Hong and Kim TG, "Design and implementation of simulators interoperation layer for DEVS simulator", In: Proceedings of M&S-MTSA'06, Ottawa, July 2006, pp.195–199.
[6] Fujimoto, R.M. Parallel Discrete Event Simulation. Comm. of the ACM 33, 10, 1990, 30-53.
[7] Q. Liu, and G. Wainer, "Accelerating Large-scale DEVS-based Simulation on the Cell Processor", Proceedings of the 2010 Symposium on Theory of Modeling and Simulation – DEVS Integrative M&S Symposium, Orlando, FL, pp. 191-198, 2010.
[8] B.P Zeigler, T.G Kim and H. Praehofer, *Theory of Modeling and Simulation*. Academic Press, Orlando, FL, USA, 2000.
[9] B.A. Cota and R.G. Sargent, "A Modification of the Process Interaction World View," ACM Transactions on Modeling and Computer Simulation. Vol. 2, No.2, pp 109-129, April 1992.
[10] Wan Bok LEE and Tag Gon KIM, "Performance Evaluation of Concurrent System Using Formal Model: Simulation Speedup," IEICE Transactions on fundamentals of electronics, Communications and computer sciences, Vol. E86-A, No. 11, pp. 2755-2766, 2003.
[11] Tag Gon Kim and Sung Bong Park, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++," In: Proceedings of the 1992 European Simulation Multiconference, York, UK, June 1992, pp.152–156.
[12] Yeong Geol Kim, Myung Soo Ahn and Tag Gon Kim, "Optimization of Model Execution Time in the DEVSim++ Environment," Eurpean Simulation Symposium, Serial. 9, Passau, Germany, pp. 215 - 219, Oct., 1997.
[13] Alexandre Muzy and J. Nutaro, "Algorithms for Efficient Implementations of the DEVS & DSDEVS Abstract Simulators," 1st Open International Conference on Modeling & Simulation (OICMS), 2005.
[14] Lewis Ntaimo, Bithika Khargharia "A Cellular DEVS Dynamic Fire Spread Simulation Model With An Optimized Cell Space", Project Report, Electrical and Computer Engineering Department, University of Arizona, May, 2003.
[15] G. Wainer, and N. Giambiasi, "Application of the Cell-DEVS Paradigm for Cell Spaces Modeling and Simulation", Simulation, v. 76, p. 22-39, 2001.
[16] Xiaolin Hu and B. P. Zeigler, "A High Performance Simulation Engine for Large-scale Cellular DEVS Models," High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference (ASTC), p. 3-8, 2004.

Biography

Se Jung Kwon received his BS degree in Department of Computer Science from the KAIST in 2009 and MS degree in Department of Electrical Engineering. He is currently a PhD student in the Department of Electrical Engineering at the KAIST. His research interests include simulation algorithms for DES, DEVS execution environment and framework for describing virtual world.

Tag Gon Kim received his PhD in computer engineering with specialization in systems M&S from the University of Arizona, Tucson, AZ in 1988. He was an Assistant Professor in Electrical and Computer Engineering at the University of Kansas, Lawrence, KS from 1989 to 1991. He joined the Electrical Engineering Department, KAIST, Daejeon, Korea in Fall, 1991 and has been a Full Professor in the EECS Department since Fall, 1998. He was the President of The Korea Society for Simulation (KSS). He was the Editor-In-Chief for Simulation: Transactions for the Society for Computer Modeling and Simulation International (SCS). He is a co-author of the text book 'Theory of Modeling and Simulation', Academic Press, 2000. He has published about 200 papers in M&S theory and practice in international journals and conference proceedings. He is very active in defense M&S in Korea. He was/is a consultant for defense M&S technology at various Korea government organizations, including the Ministry of Defense, the Defense Agency for Technology and Quality (DTAQ), the Korea Institute for Defense Analysis (KIDA), and the ADD. He is a Fellow of SCS and a Senior Member of the Institute of Electrical & Electronics Engineers (IEEE).