# DEVS-based Software Process Simulation Modeling: Formally Specified, Modularized, and Extensible SPSM *

KeungSik Choi, Doo-Hwan Bae, TagGon Kim
*Department of EECS,*
*Korea Advanced Institute of Science and Technology, Daejon 305-701, Korea*
*{kschoi, bae}@se.kaist.ac.kr, tkim@ee.kaist.ac.kr*

## Abstract

*This paper proposes DEVS (Discrete Event System Specification)-based software process simulation modeling method which is a formally specified, modularized, and extensible simulation modeling approach. The proposed approach adopts DEVS formalism, a general purpose discrete event modeling and simulation framework, to the software process simulation modeling domain. This approach enables us to clearly understand the software process simulation model by formal specification, and provides explicit extension point to extend the simulation model for a specific purpose.*

*This approach also provides naturally hybrid software process simulation modeling environment, which embeds DESS (Differential Equation System Specification) and DTSS (Discrete Time System Specification) into DEVS formalism. This hybrid approach overcomes some limitations of system dynamics simulation models, such as difficulties of controlling process execution, representing explicit process activity, and modeling inherent uncertainty.*

## 1. Introduction

Many researchers have investigated why Software Process Simulation Models (SPSMs) are not widely used in industry and proposed several approaches. Raffo [1] argued that industry is not using SPSMs to their full advantages, because process models are difficult to build and maintain. As a solution, he proposed designing a Generalized Software Process Simulation Model (GPSM) that can be tailored quickly to a particular project scenario and deployed rapidly. He also reviewed three possible research areas applicable to GPSM: Modularization, Software Product-Line, and Cognitive Pattern. Angkasaputra and Pfahl [2] proposed to make SPSMs more agile by taking benefits from agile methods and design patterns to reduce product delivery time and budget. Ruiz [3] developed a Reduced Dynamic Model (RDM) which simplified Abdel-Hamid and Madnick's model [4] to easily learn and understand the model and to use the model at the initial phases of a project where the available or known information about the project is little.

Although these approaches alleviate some of the difficulties in using SPSMs in industry, there are other obstacles to apply SPSMs, especially system dynamics models, in software development projects. Traditional SPSMs are difficult to understand and extend, because most of researches concentrate on developing simulation models and displaying simulation results. For example, the stock flow diagrams of system dynamics models have complicated arrows, circles, rectangles, etc., which make it difficult to understand the process activity controls and variable interactions. Furthermore, those models don't have clear specifications and are rarely verified. Because simulation models are also software systems, we need a clear specification of the simulation model. Liu [5] claimed that to improve the maintainability we have to enhance the structural and behavioral specifications.

In an attempt to answer the aforementioned arguments we propose a DEVS (Discrete Event System Specification) [6]-based software process simulation modeling technique which is a formally specified, modularized, and extensible simulation modeling approach. With this approach, we can clearly specify and verify the simulation model and extend the model using Object-Oriented framework provided by the DEVS simulation engine [7]. Another benefit of this approach is this technique can solve the disadvantages of system dynamics models such as difficulties in controlling activity sequencing, describing discrete process steps, representing attributes of individual entities, and modeling un-

---

certainties [11].

The structure of this paper is as follows. In Section 2, we briefly introduce the DEVS formalism and the simulation environment. Section 3 describes the proposed DEVS-based software process simulation modeling method and demonstrates how to extend and tailor the simulation model. Section 4 compares existing hybrid simulation approaches and proposed DEVS-based SPSM in various aspects. Section 5 summarizes the main results of this paper and gives a plan for future work.

## 2. Background

### 2.1. DEVS formalism

DEVS is a general formalism for discrete event system modeling based on set theory [6]. It allows representing any system by three sets and four functions: Input Set, Output Set, State Set, External Transition Function, Internal Transition Function, Output Function, and Time Advanced Function. DEVS formalism provides the framework for information modeling which gives several advantages to analyze and design complex systems: Completeness, Verifiability, Extensibility, and Maintainability [7]. DEVS can also approximate continuous systems using numerical integration methods. Thus, simulation tools based on DEVS are potentially more general than other tools including continuous simulation tools [8]. With those properties, we applied DEVS formalism to software process simulation modeling.

DEVS has two kinds of models to represent systems. One is an atomic model and the other is a coupled model which can specify complex systems in a hierarchical way [6]. A DEVS model processes an input event based on its state and condition, and it generates an output event and changes its state. Finally, it sets the time during which the model can stay in that state. An atomic DEVS model is defined by the following structure [6]:

$$M = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$$

where:

- $X$ is the set of input values,
- $Y$ is the set of output values,
- $S$ is the set of states,
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function, where $Q = \{(s,e)|s \in S, 0 \leq e \leq ta(s)\}$ is the total state set, e is the time elapsed since last transition
- $\delta_{int} : S \rightarrow S$ is the internal transition function,
- $\lambda : S \rightarrow Y$ is the output function,
- $ta : S \rightarrow R_{0,\infty}^{+}$ is the set positive reals bet. 0 and $\infty$
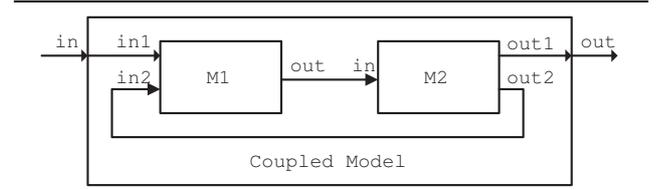


**Figure 1. Coupled DEVS model**

The behaviors represented by four functions of the atomic model are as follows:

- An atomic model can stay only in one state at any time

- The maximum time to stay in one state without external event is determined by $ta(s)$ function

- When an atomic model is in a state $(0 \leq e \leq ta(s))$, it changes its state by $\delta_{ext}$ function if it gets an external event

- If possible remaining time in one state is passed $(e = ta(s))$, it generates output by $\lambda$ function and changes the state by $\delta_{int}$ function

DEVS coupled model is constructed by coupling DEVS models. Through the coupling, the output events of one model are converted into input events of other models. In DEVS theory, the coupling of DEVS models defines new DEVS models (i.e., DEVS is closed under coupling) and then complex systems can be represented by DEVS in a hierarchical way [6]. Figure 1 shows a coupled DEVS model. M1 and M2 are DEVS models. The M1 model has two input ports ("in1" and "in2") and one output port ("out"). The M2 model has one input port ("in1") and two output ports ("out1" and "out2"). They are connected by input and output ports (e.g., "out" port of M1 is connected to "in" port of M2) internally, which is called Internal Coupling (IC). The M1 model is connected by external input, "in" port of Coupled Model, to "in1" port, which is called External Input Coupling (EIC). The M2 model is connected to output port "out" of Coupled Model (e.g., "out1" port of M2 is connected to "out" port of Coupled Model), which is called External Output Coupling (EOC). According to the closure property, the coupled model in Figure 1 can be used as a DEVS model and it can be coupled with other DEVS models.

### 2.2. Simulation environment

The DEVSimHLA [7] is a C++ based DEVS simulation environment which is integrated with Microsoft Visual Studio .NET. It, therefore, provides the advantages of Object-oriented framework such as encapsulation, inheritance, etc. The DEVSimHLA coordinates the event sched-

ules of atomic models in a system and provides classes and APIs for simulation.

With DEVS formalism and DEVSimHLA, we can get several advantages. First, we can specify the systems mathematically which gives straight forward verification method. Second, we can model hierarchical and modularized systems which enhance understandability and extensibility. Third, we can reuse simulation models by inheritance.

## 3. DEVS-based software process simulation modeling approach

This section describes how to model software development process using DEVS formalism. We develop a generic and simplified software process model to estimate the cost and duration of a project based on the initial information on the project such as project size and project duration. We will show the overall architecture of the proposed simulation model, and how to formally specify the software process simulation model using DEVS formalism. We also demonstrate how to extend and tailor the DEVS-based software process simulation model for a specific purpose. Finally, we will emphasize how our approach provides naturally hybrid simulation environment.

### 3.1. Overview

The purpose of this simulation modeling approach is to develop a formally specified, modularized, and extensible simulation model to make full advantages of SPSMs. We analyze published system dynamics models through literatures and books, and identify several limitations as follows [11]:

- Understanding and maintaining simulation models is difficult because of no clear specification

- Extending simulation models is difficult because of no explicit reuse mechanism and extending points provided

- Describing process steps is difficult because of no explicit mechanism to control the activity sequences

- Representing error prone modules or variable productivity of developers is difficult because of no individual entities and entity attributes

- Modeling uncertainties inherent in estimates of model parameters is difficult

We referenced system dynamics models provided by Vensim simulation tool [9] and Abdel-Hamid and Madnick [4], and simplified it for our purpose. We assume that we already analyzed all the dynamic interactions of software process variables using such as Causal-Loop Diagram (CLD).
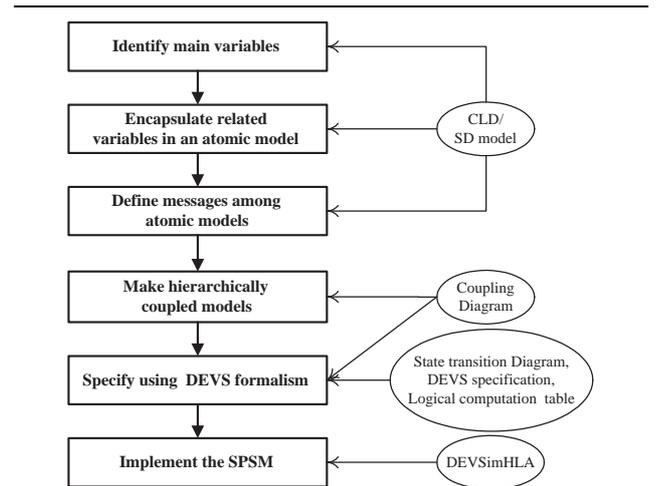


**Figure 2. Overall approach of DEVS-based software process simulation modeling**

Figure 2 shows our approach of proposed DEVS-based software process simulation modeling. First, we identify main variables which drive the simulation result variables such as an effort and a duration in software process simulation models. For example, we identified that the most important variable is a *workflow* rate which is determined by many factors such as productivity, total workforce, synergy, and communication overhead. We then construct an atomic model by encapsulating closely related variables centered on the main variable identified in the previous step, and identify interaction points (variables) among atomic models. There are two kinds of interaction points. One is a variable which is calculated by other atomic models and updated by input messages. The other is a variable which is computed in this model and becoming an output message. After constructing atomic models and interaction points, we define messages which transmit simulation information among atomic models. We then couple the models hierarchically. Hierarchical modeling enables us to modularize the simulation model and makes it simple and understandable.

When atomic and coupled models are defined, and their interactions are identified, we specify the simulation models using four methods which include a coupling diagram, a state transition diagram, DEVS specification, and a logical computation table. These methods we propose provide a clear, verifiable, understandable, and extensible specification for the software process simulation model. Finally, we implement the specification using the DEVSimHLA environment [7].

## 3.2. Overall architecture of the DEVS-based software process simulation model

Figure 3 illustrates the overall architecture of the simulation model, which is composed of a DevelopmentPhase and an ExperimentalFrame model. It shows the most basic structure of the software development project. We design this to make the model simple and easy to extend. The DevelopmentPhase model can represent the whole software development life-cycle and also can represent each phase of the software development life-cycle. The ExperimentalFrame model can be reused and extended for a specific simulation purpose.

The DevelopmentPhase model represents any phase of the software development life-cycle. It can be extended to any of the life-cycle such as requirements, design, or implementation. It also can be a Waterfall or Incremental life-cycle by coupling the DevelopmentPhase model each other. The WorkToBeDone, WorkDone, and Rework models are like a level variable in stock flow diagram. The WorkToBeDone model stores the amount of work to be done and sends it to the Work model. The WorkDone model integrates the workflow rate to compute the work done, and the Rework model computes the amount of rework to do again and sends the rework rate to the WorkToBeDone model.

The ExperimentalFrame model plays a role of a measurement system or observer like an oscilloscope in electronics. It generates inputs to the observed system, and accepts and analyzes the experiment data. It is a system that interacts with the system of interest to obtain the data of interest under specified conditions. In this simulation model, it generates WorkToDo message, which is an initial work to do. The WorkToDo message will be processed by the DevelopmentPhase model and sends the simulation data, as a WorkMonitoring message, to the ExperimentalFrame model.

The TimeIntervalGenerator model in the ExperimentalFrame is an executive which drive the simulation execution. It generates a Time event in a small-enough constant-time interval to make the WorkToBeDone model change its state and generate the WorkToDo message. The TimeInterval-Generator model enables the project variables to dynamically interact each other, which allows this model to become a naturally hybrid simulation model. The naturally hybrid simulation approach will be discussed in Section 4

The WorkMonitoring message contains simulation variables which are stored and analyzed by the SimAnalysis model in the ExperimentalFrame model. This message includes level variables, rate variables, and auxiliary variables in system dynamics representation. These variables are dynamically updated through the feedback loop, which shows the effects of complex dynamic software development process.

The Done message makes this simulation model stop. The WorkToBeDone model generates the Done message when the work is done. The interaction point of the DevelopmentPhase model (e.g., Time, Done, and WorkMonitoring) depicted in a small rectangle in Figure 3 is called an input or an output port. This is an explicit extension point in DEVS-based software process simulation model. For example, we can instantiate the DevelopmentPhase model to a Requirements model and a Design model, and then connect the Done port of the Requirements model to the WorkIn port of the Design model. The model extension issues will be further discussed in Section 3.4.

## 3.3. Formal specification using DEVS formalism

As we mentioned before, we use four methods (coupling diagram, state transition diagram, DEVS specification, and logical computation table) to specify software process simulation model. Using these methods we can develop a clear, verifiable, understandable, and extensible specification for a software process simulation model.

The coupling diagram shows the internal structure of the coupled model and the flow of simulation data. The state transition diagram specifies how the state changes as the system responds to its different inputs. This is the explicit mechanism to control the activity sequences. Based on the input and the state of the model we can change the behavior of the model. The DEVS specification defines the behavior of the model with 3 state sets and 4 functions. This provides the sound framework to verify the behavior of the simulation model, because the formal specification gives clear interpretation of that. The logical computation table describes how to calculate the project variables when the model changes its state by external input or internal time expiration.

Figure 4 shows the coupling diagram of the Work_Coupled model which shows the internal structure of the Work model in Figure 3. The Work_Coupled model comprises ScheduleMng, WorkforceMng, and WorkflowRateMng model. The ScheduleMng model calculates the remained schedule time to complete the job and based on this it calculates the required workflow. The required workflow can be KLOC/day or FP (Function Point)/day, etc. The required workflow is sent to the WorkforceMng model which returns the indicated workforce. The indicated workforce means required workers to complete the job based on the schedule. Based on this indicated workforce, the ScheduleMng model determines the amount of overwork. The WorkforceMng sends the total workforce and the WorkflowRateMng model calculates the workflow rate and work quality. The workflow rate means work done per day (e.g., KLOC/day or FP/day).
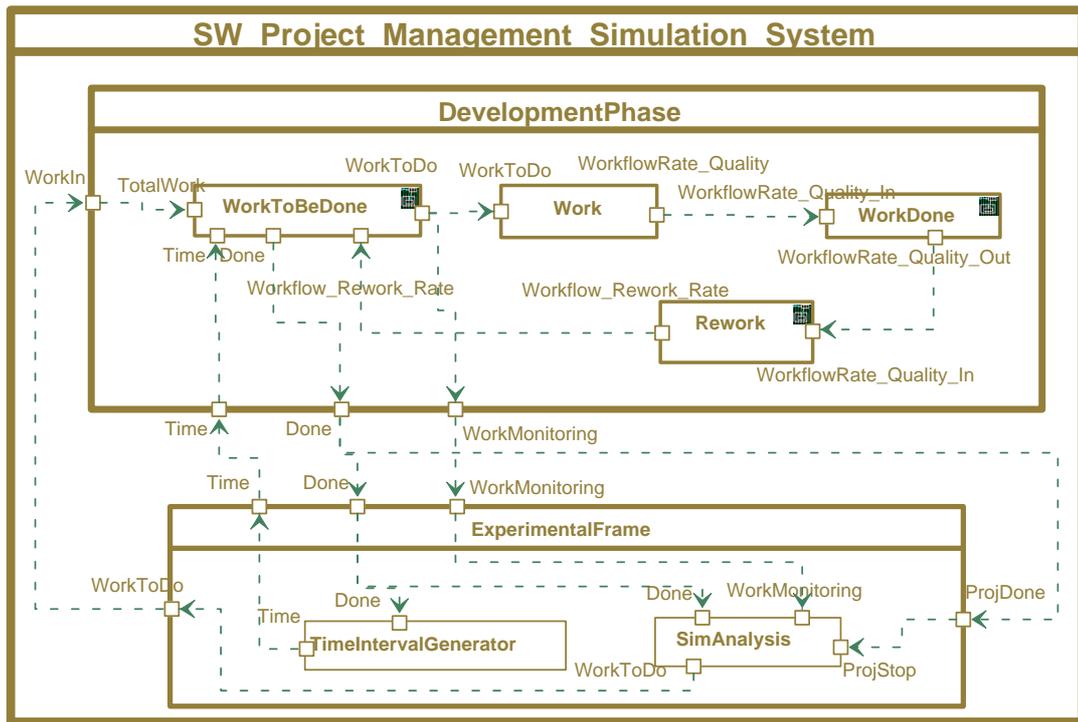
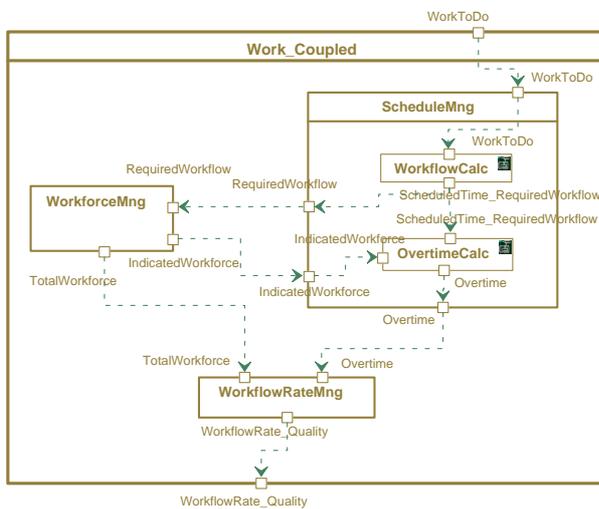**Figure 3. Overall architecture of DEVS-based SPSM**



**Figure 4. Coupling diagram of Work coupled model**

Figure 5 depicts the state transition diagram of the OvertimeCalc atomic model. This diagram shows that the initial state is a Wait state and it stays infinitely until it receives external input message. The model changes the state to the GetIndicatedWorkforce state when it receives external input message, ScheduleTime_RequiredWorkflow, and then the model changes the state to the OvertimeCalc state in response to the IndicatedWorkforce input. In the OvertimeCalc state, the model generates the Overtime output message and changes its state to the Wait state with 0 second delay.

Table 1 shows the DEVS specification of the OvertimeCalc model. It represents the same information with the state transition diagram but defines it more declaratively.

Table 2 shows the logical computation table of the OvertimeCalc model. This table specifies the type of the variables and the equations. When this model receives a ScheduledTime_RequiredWorkflow message in the Wait state, it saves the input variables, and when it receives an IndicatedWorkforce message in the GetIndicatedWorkforce state, it calculates the NormalWorkflow, SchedulePressure, and Overtime variables. When this model receives the external inputs again by the feedback structure after a constant-time interval (e.g., one hour or one day), the variables are dynamically recalculated. Through this mechanism, we imple-
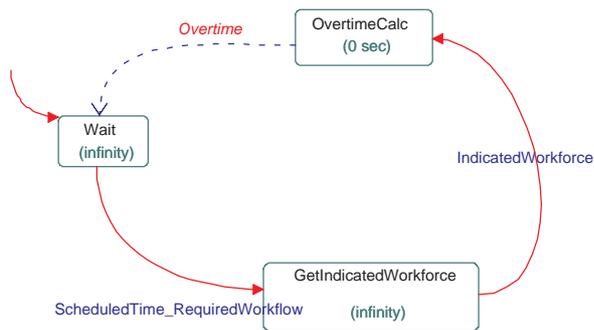
**Figure 5. State transition diagram of Over-timeCalc model**

ment feedback loop of the system dynamics. We don't specify the detailed variable types and equations in Table 2 because they are not our focus in this paper.

### 3.4. Extending and tailoring the simulation model

The model in Figure 3 is a generic and simplified one. It can estimate the effort and duration for intermediate size (21.3 Man-Month, 8 Months) [10] project. Based on this simulation model, we can extend this model rapidly and with low cost, because the base model has a clearly verified specification and explicit interfaces (ports or extension points) for extension.

For example, Figure 6 extends the base model to Waterfall life-cycle model by coupling the DevelopmentPhase model to each other. The Waterfall model starts when it re-

$$OvertimeCalc = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$$

$X$={ScheduledTime_RequiredWorkflow, Indicated-Workforce}
$Y$={Overtime}
$S$={Wait, GetIndicatedWorkforce, Overtime_Calc}
$\delta_{ext}$(Wait, ScheduledTime_RequiredWorkflow) = GetIndicatedWorkforce
$\delta_{ext}$(GetIndicatedWorkforce, IndicatedWorkforce)= Overtime_Calc
$\delta_{int}$(Overtime_Calc) = Wait
$\lambda$(Overtime_Calc) = Overtime
$ta$(Wait) = $ta$(GetIndicatedWorkforce) = Infinity
$ta$(Overtime_Calc) = 0

**Table 1. DEVS specification of OvertimeCalc model**

| State transition function | Logical description |
|---|---|
| $\delta_{ext}$(Wait, Scheduled-Time_RequiredWorkflow) | Save ScheduledTime and RequiredWorkflow |
| $\delta_{ext}$(GetIndicatedWorkforce, IndicatedWorkforce) | Calculate NormalWorkflow, SchedulePressure, Overtime |

**Table 2. Logical computation table of Over-timeCalc model**

ceives WorkIn input which is the initial project estimation size and it ends when it receives the ProjDone message. The Requirements phase model does the job and the model outputs the Done message when it completes the job, and this message is becoming an input message of the Design phase model. Of course, we have to modify the variables and the dynamic equations of each model to apply the characteristics of each phase. The ExperimentalFrame model is also reused and extended to store and analyze the project information of each phase.

Another example is shown in Figure 7. In this example, we add a ResourcePool model which manages shared human resources. If one organization has limited resources and performs two projects simultaneously, we suffer resource conflicts. Proj_A and Proj_B are modified version of the DevelopmentPhase model in Figure 3. The WorkforceMng model in the Proj_A is modified to request new workers to the organization's shared resource pool and it receives allocated workers. We reuse other models as it is except the WorkforceMng model. The ExperimentalFrame model is extended to store and analyze the project information of two projects.
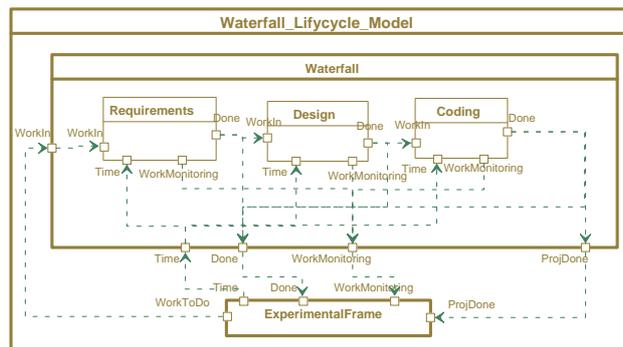


**Figure 6. Extended model: Waterfall model**
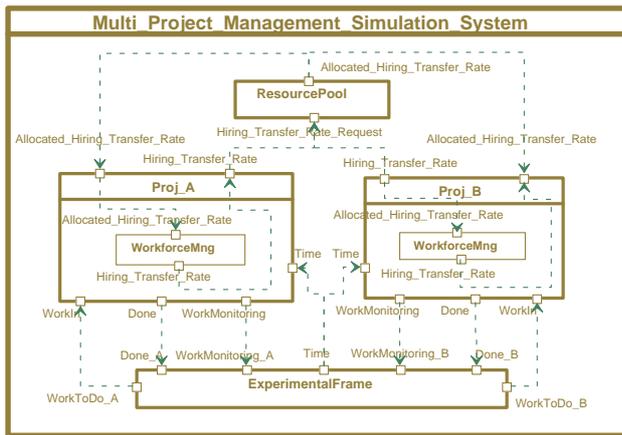
**Figure 7. Extended model: Multi-project model**

## 4. Naturally hybrid simulation modeling approach

We described DEVS-based software process simulation modeling approach, which overcomes some limitations of published system dynamics models: difficulties in understanding and extending simulation models, difficulties in describing discrete process steps, difficulties in representing individual entities and entity attributes of a variable, and difficulties in modeling uncertainties. The four methods for specifying software process simulation model provide a clear and understandable specification, and we can extend the simulation model through explicit extension points (ports). We also showed how to model discrete process steps in Section 3.4. Moreover, we can easily represent individual entities and entity attributes, and uncertainties because the proposed simulation model is basically a discrete event simulation model.

Because our approach incorporates feedback loop mechanism of system dynamics and represents discrete event, the DEVS-based software process simulation modeling technique is a hybrid simulation modeling approach. We named this approach to "Naturally hybrid simulation modeling approach" because DEVS formalism can naturally embed Discrete Time System Specification (DTSS) and Differential Equation System Specification (DESS), which will be discussed further in this section.

### 4.1. Characteristics of DEVS-based naturally hybrid simulation modeling approach

In this section, we provide theoretical backgrounds on how to model continuous systems in DEVS representation,
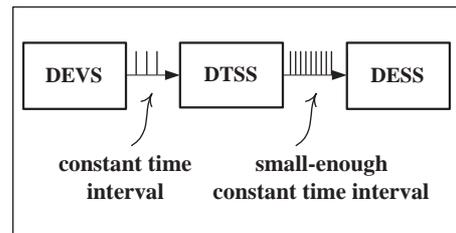


**Figure 8. Formalisms embedding**

and describes the characteristics of DEVS-based naturally hybrid simulation modeling approach.

Traditionally, differential equations have been solved with numerical integration in discrete time. In formal terms, Differential Equation System Specification has been embedded into Discrete Time System Specification [6]. The word "embedded" in this context means that any system in DESS can be simulated by DTSS. Of course, errors may be introduced in the DTSS approximation of the DESS model, but it is tolerable if the discrete time is small enough. Moreover, any DTSS can be simulated by the DEVS by constraining the time advance to be constant. Therefore, if we constrain the time advance of the DEVS to be small-enough constant-time, we can model DESS with DEVS formalism. This formalism embedding illustrated in Figure 8 makes DEVS-based software process simulation modeling technique to be a naturally hybrid simulation modeling approach. We coined "naturally hybrid" because this simulation modeling environment naturally includes both discrete and continuous simulation modeling techniques.

We also can easily model uncertainties in model parameters. For example, if we want to make a stochastic human resource transfer model, we can extend the DEVS formalism to a stochastic DEVS. The output function of DEVS, $\lambda$, can be extended like this:

$$\tilde{\lambda} : S \times [0, 1] \to Y$$

Therefore, the output is characterized by a possibility distribution of $[0, 1]$. We can apply this to the external transition function and internal transition function as well.

### 4.2. Comparisons of published hybrid simulation approaches and DEVS-based SPSM

System dynamics models describe the interaction between project factors, but do not easily represent discrete process steps. On the other hand, discrete event models may not have enough events to represent feedback loops accurately [11]. Many researches, therefore, have tried to combine discrete event simulation and continuous simulation to

| Author | Martin [11] | Lakey [12] | DEVS-based SPSM |
|---|---|---|---|
| **Application** | Evaluate potential process changes | Project estimation & Project management | Project estimation & project management |
| **Implementation tool** | Extend | Extend | DEVSimHLA |
| **Basic approach** | Discrete event models are combined in system dynamics framework | Feedback mechanisms are approximated in discrete process | System dynamics models are implemented with DEVS which embeds DESS |
| | Process activities: DES | Process activities: DES | Process activities: DES |
| | Project environment: SD | Divide a discrete activity into multiple sub-activities and iterate multiple times | Implement feedback mechanism by constraining time advance into small-enough constant-time interval |
| **Discrete event model** | Calculates duration, total effort, errors which are passed out to system dynamics model | Product size and quality are passed on to next activity | Information on the work, process control events, and simulation results data are passed on to next activity |
| **Continuous model** | System dynamics model (human resource, manpower allocation, productivity) passes out the data to discrete event model | Each sub-activity (feedback loops of development, review, rework) iterates multiple times to incorporate dynamic feedback loops while calculating a number of equations for product, process, project factors | Workflow, human resource, manpower allocation, productivity, work quality, etc. are dynamically calculated |
| **Limitations** | Duration time of each activity can not be dynamically calculated | Coarse approximation of system dynamics model | No limitation |
| **Timing issues** | Each activity computes the duration time but advances the clock only by the specified delta time | Time advance doesn't mean anything, schedule model in each activity calculates calendar weeks | Time advance is constrained to small-enough constant-time interval, duration time of each activity is dynamically calculated |

**Table 3. Comparison of hybrid simulation modeling approaches**

model software process more realistically. Table 3 compares existing hybrid simulation approaches with our approach.

Martin et al. [11] develop a combined model that represents the software development process as a series of discrete process steps executed in a continuously varying project environment to evaluate the potential process changes. The process activities are represented in discrete event model and the project environments, such as human resource, manpower allocation, and productivity, are modeled in system dynamics. In their approach, the discrete event models are combined in the system dynamics framework as shown in Figure 9. To support hybrid model, they advance the clock by the specified delta time while preserving the discrete scheduling time computed by discrete event model. At each delta time increment, they integrate all necessary equations until the next scheduled event time

is reached. The limitation of this approach is that the duration time of each activity is not calculated dynamically. In most system dynamics approach, the activity time is dynamically influenced by many factors such as pressure, fatigue, and workforce, but this approach calculates the duration time of each activity once before executing the activity.

Lakey [12] proposes a hybrid simulation model for project estimation and management. The feedback loops are incorporated in the discrete event process by dividing a single discrete event process block into multiple iterations so that certain factors are allowed to change several times within the discrete activity. The development process has discrete four major activities as shown in Figure 9, and each of the activity incorporates four discrete sub-blocks which represent feedback loops of system dy-
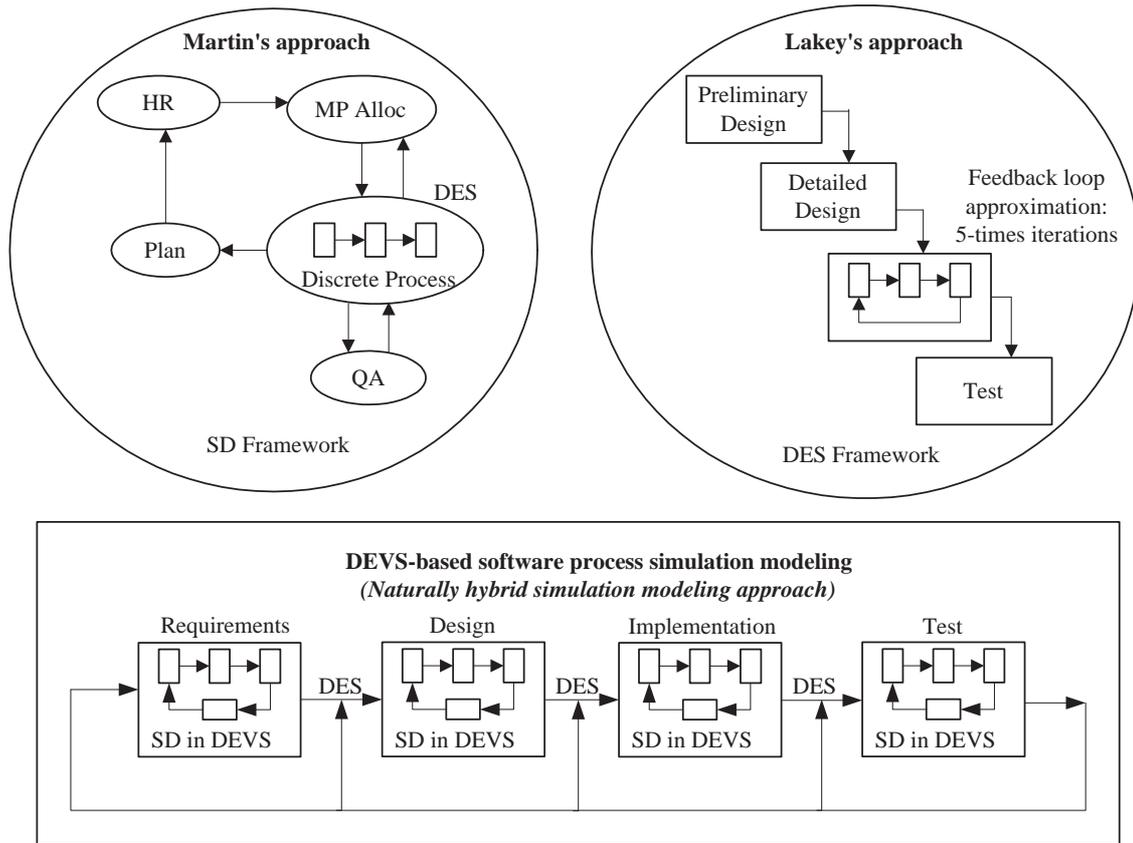
**Figure 9. Characteristics of each hybrid simulation approach**

namics. Within each of the sub-blocks are a number of equations that relate the product, process, and project factors to output parameters: Schedule, size, quality, manpower, overhead, skill level, tool support, process maturity, and functional growth. The limitation of this approach is that this may not fully represent overall dynamic feedback relationships in software development project because it approximates feedback loops by limited five iterations in each process activity.

Martin et al. [11] advance the clock at a small steady increment and integrate all necessary equations to implement system dynamics concept, and Lakey [12] iterates Development, Review, and Rework sub-activities five times in each activity to incorporate dynamic feedback loops. Martin combines discrete event model into the system dynamics framework, and Lakey approximates feedback loop in discrete event model. On the other hand, our approach embeds DESS and DTSS into DEVS formalism, and implements each activity phase (e.g., Requirements) with system dynamics model with DEVS as shown in Figure 9. This simulation modeling environment naturally includes both discrete and continuous components.

## 5. Conclusion and Future Work

We proposed DEVS-based software process simulation modeling technique which is a formally specified, modularized, and extensible simulation modeling approach. We modularized the simulation model by encapsulating closely related variables in one atomic model, and used four methods to specify a software process simulation model. Our approach enables us to make a clear, verifiable, understandable, and extensible specification for a software process simulation model. DEVS-based software process simulation modeling technique also overcomes some limitations of existing system dynamics models such difficulties as describing discrete process steps, controlling the activity sequences, and modeling uncertainties, by providing naturally hybrid simulation modeling environment.

This simulation modeling approach is unique in that it adopts DEVS formalism, a general purpose modeling and simulation framework, to a software process simulation modeling domain and implements a naturally hybrid simulation environment by embedding DESS and DTSS into DEVS formalism. We can implement the same system dynamics models as models developed by system dynamics

modeling tools (Vensim, Extend, or iThink) with discrete event simulation technique, and at the same time we can utilize the advantages of discrete event simulation because this technique is fundamentally a discrete event simulation one.

We have described a base model using DEVS-based software process simulation modeling technique and shown some examples of extended model. We, however, haven't experimented in industrial environment. We have plan to experiment a Waterfall life-cycle model in industrial setting, which will makes this simulation modeling approach more concrete.

## References

[1] D. Raffo, G. Spehar, and U. Nayak, "Generalized Simulation Models: What, Why and How?", *ProSim '03*, 2003.

[2] N. Angkasaputra and D. Pfahl, "Making Software Process Simulation Modeling Agile and Pattern-based", *ProSim '04*, 2004.

[3] M. Ruiz, I. Ramos, and M. Toro, "A simplified model of software project dynamics", *The Journal of Systems and Software*, Elsevier, 2001, pp. 299-309

[4] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1991

[5] Ling Liu, "EVOLVE: adaptive specification techniques for object-oriented software evolution", *Thirty-First Hawaii International Conference*, 1998

[6] B. Zeigler, H. Pracehofer, and TagGon Kim, *Theory of Modeling and Simulation, Second Edition*, Academic Press, New York, 2000

[7] TagGon Kim, *DEVSimHLA v2.2.0 Developer's Manual*, Korea Advanced Institute of Science and Technology (KAIST), 2004

[8] E. Kofman, M. Lapadula, and E. Pagliero, "PowerDEVS:A DEVS-Based Environment for Hybrid System Modeling and Simulation, *Technical Report LSD0306*, LSD, Universidad Nacional de Rosario, 2003

[9] *Vensim Modeling Guide*, Ventana Systems, Inc., 2004

[10] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981

[11] R.H. Martin and D. Raffo, "A Model of the Software Development Process Using Both Continuous and Discrete Models", *Software Process Improvement And Practice*, John Wiley & Sons, NJ, 2000, pp. 147-157

[12] Peter B. Lakey, "A Hybrid Software Process Simulation Model for Project Management", *ProSim '03*, 2003.