Deriving Software Process Simulation Model from SPEM-based Software Process Model *

SeungHun Park, KeungSik Choi, KyungA Yoon, Doo-Hwan Bae
Division of Computer Science,
Department of EECS, KAIST, Korea
{seunghun,choi,kayoon,bae}@se.kaist.ac.kr

Abstract

It is difficult to adopt a simulation technology for simulating a software process because of the difficulty in developing a simulation model. In order to resolve the difficulty, we consider the following issues: reducing the cost to develop a simulation model, reducing the simulation model complexity, and increasing the modularity of a simulation model. We propose an approach to deriving a Discrete Event System Specification (DEVS)-Hybrid simulation model from a Software Process Engineering Metamodel (SPEM)-based software process model. We provide the mapping between the elements of SPEM and the DEVS-Hybrid formalism and the transformation rules for automatically deriving a simulation model from a descriptive process model. Our approach resolves the issues by the transformation rules and the hierarchical and modularized modeling properties of UML and DEVS.

1. Introduction

With the increasing interest in improving the effectiveness and efficiency of a software process, software process management becomes one of the most important issues. Organizations have tried to adopt software process simulation to manage their own software processes. However, it is difficult to adopt a simulation technology. One of the reasons is that the simulation models tend to be difficult to build and maintain [1]. A software process simulation model consists of the process model elements which describe the structure and behavior of a simulation model and the quantitative model elements which contain the simulation parameters and the quantitative equations expressed in terms of mathematical equations. Therefore it needs to enable a process engineer easily and efficiently to specify the process model elements and define the quantitative model elements to ease the simulation technology adoption. Quantitative data should be obtained to define the quantitative model elements which result from the statistical or other inductive analysis techniques applied to the data [2]. Obtaining the quantitative data is another difficulty regardless of developing the simulation model. In this paper, we focus on the way how to specify the process model elements of a simulation model more easily and efficiently. In order to resolve the difficulty in developing the simulation model, we consider some key aspects of simulation modeling technology based on the issues that Raffo et al. presented [1]:

- **Reducing the cost to develop a simulation model**
  Simulation model development needs the knowledge for designing the process to be performed as well as detailed skills for simulation tools or languages. It results in communication difficulties between the stakeholders, eventually causing difficulties in technology adoption. The consistency issue between the process model to be performed, the descriptive process model, and the software process simulation model is another cost associated with simulation model development. This forces the software engineers to spend additional effort to develop process simulation models even if the organization has a detailed process model. We need to reduce the development cost by automating the simulation model development as much as possible, minimizing the intervention of human agents.

- **Reducing the simulation model complexity**
  Most simulation models are developed at once in the detailed level. It requires an amount of knowledge and experience to develop the model. We need to gradually develop the simulation model, lowering the abstraction level of the model.

- **Increasing the modularity of a simulation model**
Modularization of a simulation model makes the reusability of the model increase. Once we identify and generalize the reusable modules, these modules can be reused in the construction of a new simulation model. We need to allow a simulation modeling technique to support the modularization.

We propose an approach to deriving a software process simulation model from a descriptive process model. We use Software Process Engineering Metamodel (SPEM) [3] and Discrete Event System Specification (DEVS)-Hybrid formalism for our research [4]. SPEM is a metamodel for defining processes and their components as a standard for process modeling [3]. SPEM can be defined as a UML Profile. It allows SPEM to gain the benefit of the expressiveness of UML. The DEVS-Hybrid simulation model is based on the DEVS-Hybrid formalism, which is an extension of DEVS formalism to the hybrid software process simulation [4]. We provide the three steps for deriving a simulation model: Mapping, Modeling, and Transforming. In the Mapping step, we provide the mapping between the elements of SPEM and DEVS-Hybrid formalism. A descriptive process model is developed by using UML based on SPEM in the Modeling step. Finally, the descriptive process model is transformed into a DEVS-Hybrid simulation model by applying the transformation rules.

This approach enables the simulation model development cost to be reduced by automatically transforming a descriptive process model to a simulation model, minimizing the intervention of the stakeholders in developing the simulation model. We can develop the hierarchical and modularized simulation model to be reused in the construction of a new simulation model by using UML and DEVS-Hybrid simulation model.

The structure of this paper is as follows. In Section 2, we briefly introduce the DEVS-Hybrid formalism as a background. Section 3 describes how to derive a DEVS-Hybrid simulation model from a SPEM-based descriptive process model. Section 4 provides a case study as a validation for this approach. Section 5 introduces the related work and Section 6 summarizes the main results of this paper and gives a plan for future work.

2. DEVS-Hybrid formalism

DEVS-Hybrid simulation model is based on the DEVS-Hybrid formalism, which is extended to accommodate the hybrid characteristics of software development process [4]. DEVS-Hybrid simulation model uses the system dynamics modeling to convey the details concerning the activity behaviors and managerial policies, while the discrete event modeling controls the activity start/finish and sequence. DEVS-Hybrid can represent the discrete activities explicitly and consistently with the continuously varying project environments by fully incorporating the feedback mechanism of the system dynamics. Similarly DEVS [5], DEVS-Hybrid simulation model has two kinds of models to represent systems. One is an atomic model and the other is a coupled model. While the coupled model is the same as DEVS, the atomic model is extended. The DEVS-Hybrid formalism for the atomic model as follows [6]:

\[
\text{DEVS-Hybrid} = \langle X, Y, Y^{\text{phase}}, S, \delta_{\text{ext}}, \delta_{\text{int}}, C_{\text{phase}}, \lambda, t_a \rangle
\]

where:

- \(X\) is the set of input values.
- \(Y\) is the set of output values.
- \(Y^{\text{phase}}\) is the set of output values, which is the phase event triggered by phase event condition function (\(C_{\text{phase}}\)).
- \(S\) is the set of states.
- \(\delta_{\text{ext}} : Q \times X \rightarrow S\) is the external transition function, where \(Q = \{(s, e) \mid s \in S, 0 \leq e \leq t_a(s)\}\) is the total state set, \(e\) is the time elapsed since last transition.
- \(\delta_{\text{int}} : S \rightarrow S\) is the internal transition function.
- \(C_{\text{phase}} : Q \times X \rightarrow \text{Bool}\) is the phase event condition function for conditioning the execution of the phase event.
- \(\lambda : S \rightarrow Y\) is the output function.
- \(t_a : S \rightarrow [0, \infty)\) is the set positive reals between 0 and \(\infty\).

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 \(t_a(s)\) function. \(\delta_{\text{ext}}\) distinguishes the input into discrete or continuous one, and the input follows two different processing paths after that. When an atomic model is in a state \(0 \leq e \leq t_a(s)\), it changes its state by \(\delta_{\text{ext}}\) if it gets an external discrete event. If the model gets the external continuous input, which contains the flow(rate) and stock variables, the model updates the stock variables by using the equation in shown in [4]. If possible remaining time in one state is passed, it generates output by \(\lambda\) and changes the state by \(\delta_{\text{int}}\). If a phase event condition function becomes true, the phase output event (\(Y^{\text{phase}}\)) occurs.

A coupled model is constructed by coupling the atomic models or the coupled 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.

3. Deriving DEVS-Hybrid simulation model from SPEM-based process model

Figure 1 illustrates the overview of our approach to derive a DEVS-Hybrid simulation model from a SPEM-based descriptive process model. First, we map the elements of SPEM to be used for constructing a simulation model to the elements of DEVS-Hybrid formalism. The mapping shows how the information contained in a SPEM-based process...
model is reflected in the DEVS-Hybrid simulation model. Several UML diagrams can be used to present different perspectives of a software process model. We use Use case diagram, and Activity diagram. We assume the quantitative model elements are already defined. The quantitative equations and parameters are integrated in activity diagrams. The descriptive process model is transformed into a DEVS-Hybrid simulation model by sequentially applying the transformation rules: the structural and the behavioral. In our approach, the structural aspect includes the overall execution sequence of the simulation model, as well as the overall structure of the simulation model. The overall structure describes what coupled models or atomic models are in a simulation model. The overall execution sequence simply provides the information of how to couple the atomic models or coupled models with their input and output messages. On the other hand, the behavioral aspect means the behaviors for calculating the quantitative parameters in an atomic model. It describes how the states change to calculate the parameters. We also propose a method to automate the transformation using commercial tools. The DEVSHybrid simulation model is executed by DEVSim++, which is a C++ based DEVS simulation environment [7].

3.1 Mapping the elements between SPEM and DEVS-Hybrid formalism

The process model elements of a simulation model provide overall structure and execution sequence of the simulation. The simulation engine executes the process defined by the simulation model and calculates the quantitative parameters. With this observation, we classify the elements for simulation into the two types: the structural elements, and the behavioral elements. We map the elements of SPEM to the elements of DEVS-Hybrid formalism as shown in Table 1. The mapping is based on the similarity between the semantics of each element.

The main structural elements for process description are defined in ProcessStructure package of SPEM [3]. The structural elements include WorkDefinition, Activity, and WorkProduct. The Activity is the most basic element for process description. We map Activity to the atomic model, a basic modeling element in a DEVS-Hybrid simulation model. The WorkDefinition is the work performed in a process and consists of Activities. A Phase is a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria. Phase and WorkDefinition are mapped to coupled model, which couples atomic models together to model complex system hierarchically. The WorkProduct is anything produced, or consumed by an Activity in a process. An activity interacts with the other activities by sending and receiving the work products between activities. WorkProduct, therefore, can be mapped to a set of input and output events which trigger the execution of atomic models.

The behavioral elements include Step, and Transition. One Activity is decomposed into several Steps which describes the behavior of an Activity. In DEVS-Hybrid formalism, an atomic model consists of several states. We map Step to state by an analogy between Activity and atomic model. External transition function, internal transition function, output function, time advance function, and phase condition function of DEVS-Hybrid formalism are mapped to Transition of SPEM. External and internal transition functions specify a transition between two states. The external transition function is triggered by an external event, an input event \( X \) of DEVS-Hybrid formalism. On the other hand, the internal transition function is triggered after the time given by the time advance function has elapsed. The

<table>
<thead>
<tr>
<th>Simulation Elements</th>
<th>SPEM</th>
<th>DEVS-Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WorkDefinition, Phase</td>
<td>Coupled model</td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Atomic model</td>
<td></td>
</tr>
<tr>
<td>WorkProduct, Document</td>
<td>Input value((X)), Output value((Y)), Phase output value((Y^{\text{phase}}))</td>
<td></td>
</tr>
<tr>
<td>Behavioral Elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External transition function ((\delta_{\text{ext}}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal transition function ((\delta_{\text{int}}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output function ((\lambda))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time advance function ((\mu))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase event condition function ((C_{\text{phase}}))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TimeEvent specified with the keyword *after* which triggers a state transition is mapped to the time advance function. The output function which generates an external output just before an internal transition takes place is mapped to an action on a transition. Guard condition on a transition is mapped to the phase event condition function which decides whether a specific phase or an activity is complete.

### 3.2 Developing a descriptive process model using UML

The UML profile for SPEM gives benefits of using UML diagrams to present different perspectives of a software process model [8]. We use the two UML diagrams among them: Use case diagram, and Activity diagram. The examples are presented in Section 4. Use case diagram describes the assignment of ProcessRoles to Phases or Activities and represents the Work Breakdown Structure (WBS) of the process. For example, a Phase can include many Activities. Stereotyped ≪include≫ relation is used to specify the relationship between a Phase and the Activities which comprise the phase. With this use case diagram, we can construct hierarchically the coupled model of a DEVS-hybrid simulation model. Activity diagram represents the sequence of activities with their input and output WorkProducts. As shown in the Table 1, Activity is an atomic model and the WorkProduct is the input and output messages. Activity diagram completes the construction of structural aspect including the sequence of a simulation model. Activity diagram can illustrate the behavior of an activity with the ActionStates which represent Steps of an Activity. In general, the statechart diagram illustrates the behavior of an activity. SPEM specification defines that Step should be used only in the activity diagram. We restrict ourselves to stay as close as possible to standard SPEM to minimize the technical complexity. In this usage of the activity diagram, it defines how to change the state based on the input message and generates outputs after entering a specific state like the statechart diagram in order to calculate the quantitative parameters. All the parameters defined in the activity diagram are used as attributes of the atomic model and the attributes are instantiated by the scenario of DEVS-based simulation model. The quantitative equations and parameters are incorporated in the activity diagrams which describe the behavior of activities. The keyword *do* represents an ongoing activity that is performed as long as the modeled element is in the state. Using the keyword *do*, we specify the equations and parameters. For example, if we want to calculate the development rate of a specific activity, we describe the equation for the development rate with the expression, “*do* DevelopmentRate_A = ManpowerRate_A*Productivity_A".

### 3.3 Transforming a descriptive process model to a DEVS-Hybrid simulation model

We define the two transformation rules: the structural transformation rule, and the behavioral transformation rule. The structural transformation rule uses a use case diagram and the activity diagram describing the overall sequence of all the activities in a process as an input. Figure 2 shows the structural transformation rule. We assume the activities in the activity diagram are one to one mapped to the use cases stereotyped ≪Activity≫ in the use case diagram. The use cases stereotyped ≪Phase≫ are transformed into the coupled models. Activities included in a phase are transformed into the corresponding atomic models. The interactions between the activities in the same phase are transformed into Internal Couplings (ICs) of the phase. Inputs from the activities in the different phases are transformed into the input variables of the coupled model and External Input Couplings (EICs) between the atomic models which receive the inputs and the coupled model including the atomic models. Similarly, outputs toward the activities in the different phases are transformed into External Output Couplings (EOCs) and output variables of the coupled model. One of the output variables is transformed into the phase output variable.
Input: a UseCase diagram, Activity diagrams describing the behavior of the activities
Output: Atomic model
- UC: a use case in the UseCase diagram
- state: state in an atomic model
- ActionStates: Actions in the Activity diagram
- E; external events on all the transitions in an Activity diagram
- A; actions on all the transitions in an Activity diagram
- All: an action associated with the TimeEvent

For all use cases in the UseCase diagram
if UC’s stereotype is "Activity" 
  For the activity diagram which is mapped to UC1 
    ifguarded condition on a transition 
      
      \[ X_{source} = S \]
      \[ Y_{source} = A \]
      \[ Y_{source} = \text{guard condition} \]
    
    else
      \[ Y = A \]
  
  For all events e in the Activity diagram 
  if \( e = \) "after" 
    \[ \delta_{ext} = \text{(the source state, the target state)} \]
    \[ \tau_s = \text{time of after} \]
  
  else 
    \[ \delta_{int} = \text{(the source state, the target state, e)} \]
    \[ \tau_s = \infty \]

Figure 3. The behavioral transformation rule.

Figure 3 shows the behavioral transformation rule. All ActionStates representing the Steps are transformed into the states of the atomic model and all external events are transformed into input variables. Guard condition on a transition is used to check whether an activity is complete or not and transformed into the phase event condition function. The action with a guard condition is transformed into the phase output variable. The transition triggered by an external event is transformed into the external transition function. On the other hand, the transition triggered when the specified time is elapsed is transformed into the internal transition function and the action on the transition is transformed into the output function.

We need to confirm that the transformation rules operate correctly. We validate the transformation rules with the criteria proposed in [9]: termination, and behavioral equivalence. The termination means that the transformation rules should be applied finitely. The number of applications of the structural transformation rule and the behavioral transformation rule are limited to the number of use cases. Therefore, the transformation rules in our approach are applied finitely. The behavioral equivalence means the behavior of a source model is preserved in the transformed model. The behavior of models is described in the activity diagrams and atomic models. Table 2 shows the mapping between the elements of an activity diagram and an atomic model. The mapping is deduced from the fact that the two models are based on the state machine. As shown in Table 2, the behavioral properties of every element of an activity diagram is one to one mapped to atomic model of DEVS-Hybrid simulation model. The behavior of the source model, therefore, is preserved in the transformed model.

The transformation can be automatically implemented. Most of UML tools support XML Metadata Interchange (XMI) for interchanging models in a serialized form. We can analyze the exported XMI file and define the transformation rules of Extensible Stylesheet Language Transformations (XSLT). After extracting all the necessary information discussed in this section through XSLT processing, we can finally generate DEVSim classes by Document Object Model (DOM) processing.

4. A case study

In this section, we provide a case study for validating our approach in the industry. We develop the simulation model supporting for deciding a government acquisition policy. It is difficult to decide how to apply the acquisition policies in dynamically changing project environment. The simulation model can help project managers to analyze the effects of the acquisition policies. We describe the procedure for developing the simulation model and show the result of simulation.

4.1 Project characteristics for simulation

Boehm et al. present the application of several acquisition policies such as Cost As an Independent Variable (CAIV), and Schedule As an Independent Variable (SAIV) to manage the risks in acquisition environment [10]. We evaluate the appropriateness and effectiveness of those policies in managing requirements creeping using simulation. The project introduced in this paper is performed by an acquirer and the supplier consisting of several small organizations. The project type is the perfective maintenance for an existing system. The characteristics of the project for the simulation are shown in Table 3.
Table 3. The characteristics of the project.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Size</td>
<td>30 KSLOC</td>
</tr>
<tr>
<td>Life-cycle model</td>
<td>Waterfall</td>
</tr>
<tr>
<td>Creeping requirements</td>
<td>50% of the initial job size</td>
</tr>
<tr>
<td>Pattern of creeping</td>
<td>Design (77%), Code (13%), Test (10%)</td>
</tr>
<tr>
<td>Cost upper limit</td>
<td>10% of the initial cost</td>
</tr>
<tr>
<td>Schedule upper limit</td>
<td>10% of the initial schedule</td>
</tr>
</tbody>
</table>

4.2 The descriptive process model represented by UML

The use case diagram shown in Figure 4 describes the part of the decomposition of the process into several phases and activities applied to the project. The process component “Develop Business Process Management (BPM)” consists of the four phases: “Requirements”, “Design”, “Code”, and “Test”. Each phase also consists of several activities such as “Identify the customer requirements”. Figure 5 shows the inputs, outputs, and sequences of activities and the swimlane represents ProcessRoles assigned to each activity. This activity diagram gives the overall sequence of the process. Figure 6 describes the behavior of the activity “Identify the customer requirements”. This activity diagram consists of the two steps: “Interview with customer”, and “Develop requirements list”. When the activity “Identify the customer requirements” receives the work product “Customer needs” as an input, the activity enters to the state “Interview with customer” and calculates the effort and the amount of the performed work. After entering the state “Develop requirements list”, TimeEvent occurs after the amount of time given to the parameter “TimeForDevelopList” elapses. With this event, if customer needs are identified over than 90%, then the work product “Requirements list” is produced as an output and transferred to the next activity “Analyze the requirements”.

The simulation equations and parameters are defined based on the quantitative model in [6]. For example, we can calculate the development rate of each activity by the equation, \[ \text{DevelopmentRate} = \text{ManpowerRate} \times \text{Productivity}. \]

4.3 DEVS-Hybrid simulation model

Figure 7 shows the part of the derived simulation model by applying the transformation rules to the descriptive process model. Each phase is transformed into the corresponding coupled models and includes the activities which are transformed into the corresponding atomic models. For example, the phase “Requirements” includes the two activities in the use case diagram: “Identify customer requirements”, and “Analyze the requirements”. The coupled model “Requirements” also includes the two atomic models in the DEVS-Hybrid simulation model: “Iden_Cust_Reqs”, and “Analy_Reqs”. The name of those atomic models has the compressed form of the activity “Identify customer requirements” and the activity “Analyze the requirements”, respectively, because of the convenience of displaying. The work products of the activities are transformed into the corresponding EICs, EOCs, and ICs. The work product “Requirements list”, for example, is sent from the activity “Identify customer needs” and received by the activity...
“Analyze requirements” in the descriptive process model. Because the two activities are in the same phase, the work product should be transformed to IC followed by the structural transformation rule. The IC between the atomic model “Identify Customer requirements” and the atomic model “Analyze requirements” is “Requirements list”. The transformation, therefore, is correctly performed.

Figure 8 shows the acquisition project management simulation tool which is implemented using the DEVSim++ and available at [11] for our research. We can design various simulation scenarios by changing the project size, simulation parameters for creeping requirements, and acquisition policies. We have to select the acquisition policies we want to apply and set the appropriate effort/schedule upper limit to analyze the effects of the policies on creeping requirements. The simulation tool can show the simulation results with the numerical values and a chart.

Table 4 shows the simulation results. The Base Case means no requirements creeping. The Req. Creep shows the result when the requirements creeping occurred without acquisition policies. CAIV and SAIV show the results of the application of each policy. The analysis of the results is out of scope of in this approach. The project manager can decide what policy is appropriate the current project depending on the results.

5. Related work

There are researches to develop a simulation model more systematically and efficiently. Pfahl et al. developed a methodology for integrated measurement, modelling and simulation (IMMoS) [2]. IMMoS helps industrial practitioners to build system dynamics (SD) models efficiently and effectively. IMMoS integrates system dynamics modelling with goal-oriented measurement and descriptive process modeling. In order to integrate the descriptive process model into a SD model, authors map the descriptive process model elements to SD model flow graphs representation. Quantitative models by goal-oriented measurement are used to define start conditions, implicit management decision rules, and explicit management decision rules of SD models. Rus et al. proposed a methodology for developing discrete event simulation models of software development processes [12]. Authors present the activities related to the development of a simulator: simulator requirements identification and specification, analysis and specification of the process to be modeled, design and implementation of the simulation model, and verification and validation of the model. In this approach, a simulation model designer needs to understand and analyze the descriptive process model to be simulated, and develop a simulation model based on the result of analysis. The two approaches provide the systematic methodologies to develop a simulation model based on the descriptive process model. However, those approaches do not provide the concrete techniques to resolve the issues we raised: reducing the cost, reducing the complexity, and increasing the modularity. Raffo et al. proposed the generalized process simulation model (GPSM) concepts [13]. Authors introduce generic building blocks which enable modularization, and reuse of components. A GPSM is developed by tailoring the existing generic GPSM blocks. This work makes it easy to reduce the time and cost of building a simulation model by reusing the modularized building blocks. However, it still requires the communication over-
head between stakeholders to select and tailor the building blocks.

There are approaches to using UML as simulation modeling tool [14],[15],[16]. Several UML diagrams such as use case diagram, activity diagram are required to present different perspectives of a software process. Most of those approaches, however, use sequence diagram, which represents the behaviors of a system, to develop a simulation model because they do not focus on the software processes to be performed but the systems to be developed. They also do not provide the mechanism to integrate the quantitative equations in a descriptive process model.

6. Conclusion and future work

We proposed an approach to deriving a DEVS-Hybrid simulation model from a SPEM-based descriptive process model by applying the transformation rules. This approach can resolve the issues that we raised.

- **Reducing the cost to develop a simulation model**
  Once a descriptive process model is developed, a simulation model can be automatically generated from the descriptive process model. It minimizes the interventions of stakeholders resulting in reduction of communication cost. It also ensures the consistency because the descriptive process model is directly transformed into a simulation model.

- **Reducing the simulation model complexity**
  UML supports the hierarchical modeling with several diagrams. DEVS-Hybrid formalism also supports the hierarchical modeling with the coupled models, and the atomic models. We can develop the overall structure of a simulation model at first and then build up the detailed behavior.

- **Increasing the modularity of a simulation model**
  UML is an object oriented modeling language and supports enough the modularization of a model. A DEVS-Hybrid simulation model consists of several building blocks such as the coupled model, and the atomic model. The building blocks can be reused in the new simulation model.

As a future work, we will apply our approach to a variety of methods for quantitatively analyzing a descriptive process model. At first we will apply our approach to analyzing the impact of process changes during process enactment in Process-centered Software Engineering Environments (PSEEs). A PSEE contains the descriptive process model to be enacted. By transforming the process model to be changed into a simulation model and simulating the model, it can help a project manager decide whether the change is adopted or not. We will apply our approach to process construction or process tailoring by using the existing process modules used in constructing a simulation model.

References