Simulation Modelling Practice and Theory 18 (2010) 650–662

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Mediator approach to direct workflow simulation

Duckwoong Lee, Hayong Shin, Byoung K. Choi *

VMS Lab., Department of Industrial and Systems Engineering, KAIST, 335 Gwahak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea

ARTICLE INFO

Article history:
Received 26 November 2008
Received in revised form 29 July 2009
Accepted 12 January 2010
Available online 18 January 2010

Keywords:
Direct workflow simulation
Mediator
DEVS model
Participant simulator

ABSTRACT

This paper presents a *direct workflow simulation* method with which the future *enactment service processes* of a BPM system can be simulated directly (i.e., without a model conversion). The proposed method may easily be implemented on a commercial BPM system by plugging in a couple of software modules (no internal modification of the BPM system is required). Previous researches on workflow simulation relied mostly on *conversion methods* in which *process definition models* (PDMs) are converted to simulation models and the simulation is performed by a separate simulator. More recently, a direct workflow simulation method based on the concept of *listener* was proposed. However, with the *listener approach*, (1) some internal modification of the BPM system is required, (2) PDMs have to be modified slightly, and (3) reliable simulation is not guaranteed. The direct workflow simulation approach proposed in this paper, which we call the *mediator approach*, is free of these shortcomings. Moreover, the mediator approach is suitable for workflow simulation involving multiple BPM systems. In a 'direct' workflow simulation, (1) the work-list handler of each *participant* is replaced by a *participant simulator*, (2) simulation is carried out by the workflow engine of the BPM system, and (3) a software module called *synchronization manager* (mediator or listener) handles time synchronization during simulation. In this paper, the architecture and detailed logic of the mediator are described as DEVS models. The behaviors of participant simulators are also described as DEVS models. The proposed workflow simulation method has been implemented on a commercial BPM system as well as on an academic BPM system, and an illustrative workflow simulation example is provided.

1. Introduction

According to the Workflow Management Coalition, the *workflow management system* (WfMS) is a software system that completely defines and automatically executes workflows in order to manage the actual flow of work so that the right work is done at the right time with the right information by the right person in the organization [10]. Recently, a WfMS used for business process management (BPM) is often called a ***BPM system***, with an emphasis on orchestrating operational business processes that are driven by explicit process designs [2,9]. Along with this development is an increased awareness of need for workflow simulation in BPR (business process reengineering) [9], and the simulation is regarded as a key function of a BPM system [2,3].

The software module of a BPM system in charge of managing the actual flow of work is called a *workflow engine* or ***enactment server*** (we use the latter term throughout the rest of the paper), the service provided by the enactment server to automatically execute workflows is called *enactment service*, and the people involved with enactment service are called ***participants***. An explicit model of business processes to be managed is called **PDM** (process definition model), and an instance of PDM being executed is called ***process instance***. A process instance is a directed graph with each node representing an *activity* to be performed by a participant. Activities are sent to the participants via ***work-list handlers***.

* Corresponding author. Tel.: +82 42 350 3115; fax: +82 42 350 3110.
E-mail address: bkchoi@kaist.ac.kr (B.K. Choi).

1569-190X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.01.009

The process life cycle of BPM consists of a series of phases [9]: *Discovery* (clarification of how business processes work), *design* (building PDMs for business processes), *execution* (providing enactment services), *operation* (interacting with participants while monitoring the business processes), *optimization* (improving PDM), and *analysis* phases. There may be three levels of feedback in the BPM life cycle: (1) feedback from the operation phase to the execution phase at run time, (2) feedback from the optimization phase to the design phase at build time, and (3) feedback from the analysis phase to the discovery phase. Workflow simulation corresponding to each level of feedback is referred to as *run-time workflow simulation* (for dynamic rescheduling of process instances), *build-time workflow simulation* (for improving PDM), and *BPR workflow simulation*, respectively. A BPM system equipped with these feedback mechanisms (i.e., workflow simulations) is called a *closed-loop BPM system* [34]. Often the terms ***workflow simulation*** and *BPM simulation* are used interchangeably [18].

Early researches on workflow simulation focused mostly on BPR [17,19–21,23,24,33], where general business processes or workflows were simulated. These researches had little to do with BPM systems. Another line of workflow simulation researches focused on validating and/or optimizing PDMs of a BPM system, where the PDM is converted into a formal model such as Petri-net and DEVS or to a proprietary simulation language and then simulation is carried out with the converted model [1,14,15,26,28,30–32]. The build-time simulation concept was also employed in animation-based debugging of software systems [6]. With this conversion approach, there may be some information loss during the conversion, and the participants' behaviors may not be easily converted.

Recently, the authors' group presented a *listener approach* to *direct workflow simulation* with which process instances, together with the behaviors of participants, can be simulated directly (i.e., without a model conversion) using the enactment service mechanism of a BPM system [22,29]. In a direct workflow simulation, (1) the work-list handler of each *participant* is replaced by a *participant simulator*, (2) simulation is carried out by the enactment server of the BPM system, and (3) a software module called *synchronization manager* handles time synchronization during simulation. However, with the *listener approach*, (1) some internal modification of the BPM system is required, (2) PDMs have to be modified slightly, and (3) reliable simulation is not guaranteed. More details of the listener approach will be given in the next section.

This paper proposes a different method of *direct workflow simulation* in which the synchronization manager 'mediates' the communications between the enactment server and participant simulators. The direct workflow simulation approach proposed in this paper, which we call a *mediator approach*, is free of all the shortcomings of the listener approach. Another advantage of the mediator approach is that it is suitable for workflow simulation involving multiple BPM systems. In this paper, details of the mediator approach are described employing the DEVS formalism [13], and an illustrative implementation using a commercial BPM system [1] is presented.

The rest of the paper is organized as follows. In order to make the paper self-contained, basics of enactment service mechanism and a review of previous works are presented in the next section. DEVS models of the mediator approach are presented in Section 3 and an implementation example is given in the section that follows. Conclusion and discussions are provided in the final section.

2. Background and previous works

2.1. Enactment service mechanism of a BPM system

As the proposed workflow simulation method makes use of the enactment service mechanism of the BPM system, basics of enactment service are briefly described using Fig. 1. For each instance of workflow, a process instance (PI) is created from its process definition model (PDM). Depicted in Fig. 1 is a PI consisting of seven activities including start and end activities

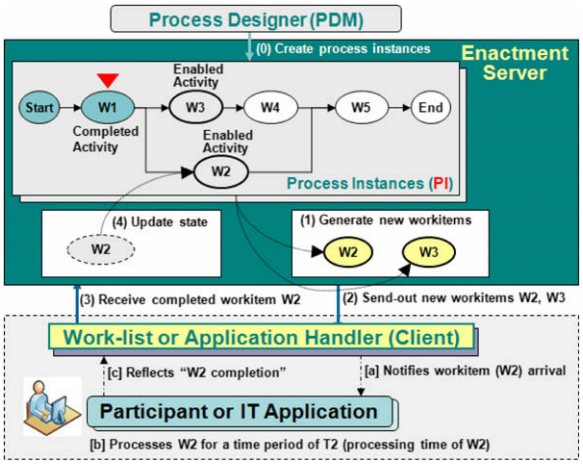


Fig. 1. Enactment service mechanism.

(note that there may be a large number of PIs in the enactment server). Right after the completion of activity W1, two activities W2 and W3 are *enabled*, and then the enactment server provides the following sequence of enactment services:

(1) Generates new workitems for the enabled activities W2 and W3.
(2) Sends out the new workitems W2 and W3 to their respective work-list handlers to be processed by the participant who is either a single person or a group of people [12].
(3) Receives the completed workitem W2 from the work-list handler (W2 is completed first).
(4) Updates the state of the PI such that W2 becomes a completed activity.

Upon receiving the new workitem W2, the work-list handler notifies it to its participant, and the participant works on W2 for a time period of T2 and 'reflects' the results at the work-list handler so that the completed workitem is returned back to the enactment server.

The communications (i.e., sending new workitems and receiving completed workitems) between the enactment server and work-list handlers (or application handlers) are made based on the *interface standards* provided by Workflow Management Coalition [10,11]. For this purpose, the enactment server has a type of internal data called ***workflow relevant data*** that can be manipulated by *work-list handlers* and other applications via a set of standard **API** (application program interface) functions.

2.2. Previous works

Works on workflow simulation may be categorized from two perspectives: how the simulation model is obtained, and when the simulation is performed. From the first perspective, workflow simulation methods may be grouped into (1) *modeling and simulation method* where the simulation model is obtained by modeling real-life business processes or workflows, (2) *conversion method* where PDM in a BPM system is converted to a simulation model, and (3) *direct or non-conversion method* where PDM or process instances are used as simulation models. From the second perspective, as described in Section 1, they are grouped into *BPR simulation*, *build-time simulation*, and *run-time simulation*.

As mentioned in Section 1, early works on 'workflow simulation' are concerned with the ***modeling and simulation method*** and BPR simulation. The potential uses of simulation in BPR is elaborated in [20,21], and how to build the simulation model for BPR is shown in [17,33]. Also, the process of analyzing workflow is described in [19], and a C++ simulation system for business process planning is presented in [24]. However, these works have little to do with a BPM system and are not directly related to the subject of this paper.

Major works on workflow simulation involving a BPM system (or workflow management system) are concerned with the ***conversion method*** that may be used for build-time simulation as well as for BPR simulation. A common feature of these works is that a separate simulator is used for simulation execution of the converted model. In some works, PDM is converted into a formal model such as a Petri-net model [14] or DEVS model [26,30], while in others, PDM is converted into a proprietary simulation language such as Simul8 [1], SIMAN [15], ProcessMap [28], or iGrafx [31]. More recently, the same *conversion method* was applied to a run-time workflow simulation, where process instances at run time are converted into colored Petri-net models and simulation is performed by a Petri-net executor [32]. Compared to the direct or non-conversion method, an advantage of the conversion method may be that workflow simulation can be performed off-line (i.e., independent of the enactment server of the BPM system). However, drawbacks of the conversion method include (1) the converted model may behave differently from the original model (PDM or process instance), (2) the behaviors of participants are difficult to convert, and (3) making a BPM system equipped with simulation functions may take a lot of time and effort.

The ***direct method*** of workflow simulation recently presented by the authors' group is the first attempt to directly simulate the PDM and process instances by using the enactment server itself [22,29]. In this method of workflow simulation, (1) *work-list handlers* are replaced by ***participant simulators*** that locally simulate the behaviors of participants, (2) the *enactment server* of the BPM system performs workflow simulation by providing 'virtual' enactment services to participant simulators, and (3) the ***synchronization manager*** 'listens' to the communications between the enactment server and participant simulators in order to synchronize simulation times. The direct method of workflow simulation is a *parallel simulation* requiring a time synchronization mechanism [4]. The direct workflow simulation system requires three types of software modules: (1) an enactment server, (2) a synchronization manager (listener), and (3) a number of participant simulators representing human participants as well as IT applications.

Shown in Fig. 2 are interactions among the software modules in the listener-based workflow simulator [29]. Data (i.e., workitems) are exchanged between the enactment Server and the participant Simulators, and the time synchronization is handled by the synchronization Manager. At the start of each enactment service cycle, Server *notifies* to Manager the number (μ) of new workitems to be sent, and then starts sending out the new workitems (W_N) to Simulators. In the mean time, Manager waits while counting the number of new workitems received by Simulators, and when the count reaches μ it makes a series of message exchanges with the Simulators, and then sends an *OK signal* to the Server so that it can start a next cycle of enactment service. To implement the listener approach, the enactment server has to be equipped with functionalities for notifying μ to the synchronization manager and for receiving the OK signal from it, which requires some modification of the BPM system. Also, Manager has to wait at least a certain amount of time to achieve a reliable communication at the presence of a network delay, which may not be guaranteed. Another limitation of the listener approach is that it cannot handle

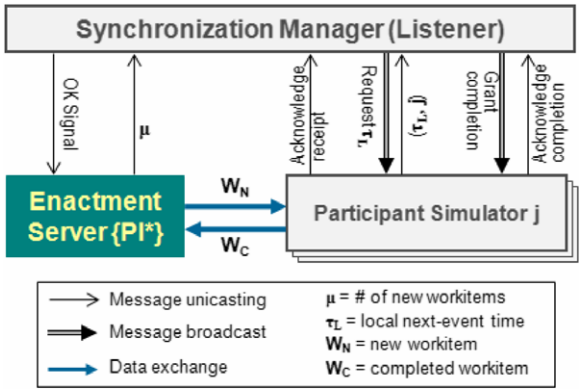


Fig. 2. Interactions among modules in the listener approach to direct workflow simulation [29].

workflow simulation involving more than one BPM system. In terms of the software design patterns [5], the direct workflow simulation method shown in Fig. 2 may be classified as the Observer pattern, but we use the term '**Listener**' borrowing from [27].

3. Mediator approach to direct workflow simulation

Now we present a novel approach to direct workflow simulation that overcomes all of the shortcomings of the listener approach mentioned above. Namely, with the proposed approach (1) no internal modification of the BPM system is required, (2) existing PDMs and process instances are used as the simulation model without any modification, (3) correct simulation is guaranteed, and (4) a direct workflow simulation may be performed with multiple BPM systems. Presented in this section are an overview of the mediator approach, DEVS models of individual components, and the architecture of our workflow simulation system. Summarized in Table 1 are all the parameters and variables used in the DEVS models.

3.1. Overview of the mediator approach to direct workflow simulation

Shown in Fig. 3 are interactions among the software modules (Server, Manager, and Simulators) in the mediator approach to direct workflow simulation. In this framework, data exchanges between the Server and Simulators are made through the Manager. Thus the role of the Manager is to mediate the communications between the Server and Simulators while managing time synchronization. This is a typical case of the **Mediator** pattern in software design patterns [5]. Here, the Manager has all connection (i.e., session) information of the Simulators that handle all new and completed workitems, and thus it can handle the 'standard' enactment services provided by the Server.

At the beginning of an enactment service cycle, the Server generates new workitems $\{W_N\}$ and starts sending them one by one to the Manager. In the mean time, the Manager 'looks into' the Server to get the number (μ) of *newly generated workitems* by using the API function ListWorkitems() specified in the WfMC standard [11]. Then, the following sequence of actions is taken by the software modules involved:

(1) The Manager passes each new workitem (W_N) received from the Server to a pertinent Simulator while counting the number (m) of *new workitems it has passed*.
(2) When m (number of passed workitems) becomes equal to μ (number of newly generated workitems), the Manager broadcasts a message to every Simulator requesting to send its ***local next-event time*** (τ_L).

Table 1
Parameters and variables used in the DEVS models.

Name	Description	Name	Description
a	Arrival time of workitem	N	Total number of participant simulators
b	Number of busy people in a group participant	NEL	Next-event list
c	Completion time of workitem	q	Number of workitems in queue
clock	Local simulation time of participant simulator	t_d	Time delay
G	Group size (number of people in the group)	W_N	New workitem
j	ID number of a participant simulator	W_c	Completed workitem
j^*	ID number of the selected participant simulator	π	Processing time of workitem
LLT	List of local next-event times (τ_L)	τ_L	Local next-event time of each participant simulator
m	Number of new workitems received	τ_G	Global next-event time
n	Number of participant simulators replied	μ	Total number of newly generated workitems

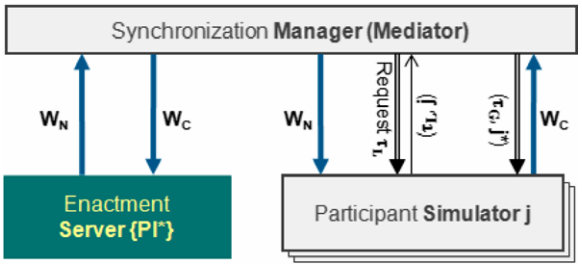


Fig. 3. Interaction among modules in the mediator approach to direct workflow simulation.

(3) Each Simulator j reports τ_L (its local next-event time) to the Manager.
(4) The Manager selects a Simulator j^* whose next-event time is smaller than others ($\tau_G = \text{Min}\{\tau_L \text{ for all } j\}$), and broadcasts τ_G (***global next-event time***) and j^* to all Simulators.
(5) The selected Simulator j^* completes its workitem (and advances its next-event time), and returns the completed workitem (W_C) back to the Manager.
(6) The Manager passes the completed workitem (W_C) to the Server which in turn updates the pertinent process instance to initiate the next cycle of enactment service.

3.2. DEVS modeling of enactment server and synchronization manager

As the internal behavior of a parallel simulation system is not easy to describe as a sequential algorithm, we use the DEVS modeling framework [13] to describe the internal structures and interactions among the modules involved (i.e., the enactment server, synchronization manager, and participant simulators). More specifically, we augmented the DEVS model primitives [35] with the Event Graph primitives [8,25]. Our version of the DEVS model primitives is summarized in Table 2, where the '?' symbol is used for *external input*, '!' for message output, 'Δ' for internal-transition elapsed time, and '∼'for condition.

Shown in Fig. 4 is a simplified version of DEVS model of the enactment server which is a simple finite state machine having three states. It is initially in the 'Wait' state and moves to the 'Processing' (update PI and generate $\{W_N\}$) state when the Start input is received. Once it has generated all the new workitems (for the enabled activities), it sends out the new workitems $\{W_N\}$ to the synchronization manager and moves to the 'Ready' state. Then, it waits in the 'Ready' state until it receives a completed workitem W_C, and then moves back to the processing state. It should be noticed that the processing state may not generate a new workitem, in which case, $\{W_N\}$ is a null set, meaning that the state is changed to 'Ready' without sending out any workitem.

Fig. 5 shows a DEVS model of our synchronization Manager together with its interactions with the enactment Server and participant Simulators. At the beginning of an enactment service cycle, the Manager stays in the 'Wait for first W_N' state with m (*number of new workitems received*) equal to zero. Then, (1) if the first $W_N(\pi, j)$ with its processing time π and the participant simulator ID j is received, the Manager moves to the 'Get μ' state after sending $W_N(\pi)$ to Simulator j and setting m to one, (2) otherwise it moves to the 'Get μ' state after a time delay of t_d. At the 'Get μ' state, the Manager obtains the value of μ (*number of newly generated workitems*) by using an API function (see Appendix A), and moves to the dummy state D1 after setting $\mu = \mu + m$. Then, at the dummy state D1, the Manager goes to the 'Wait for next W_N' state and comes back until m is equal to μ. At this point ($m \equiv \mu$), the Manager moves to the 'Wait for τ_L' state after sending the 'Request τ_L' message to all Simulators and setting n (*number of Simulators replied*) to zero.

Table 2
DEVS model primitives.

Primitives	Notation
External transition with condition	? (External-input) Condition — Value-update, ! (Message-output)
Internal transition with condition	Δ (Elapsed-time) Condition — Value-update, ! (Message-output)
State	Initial State; State; Dummy State

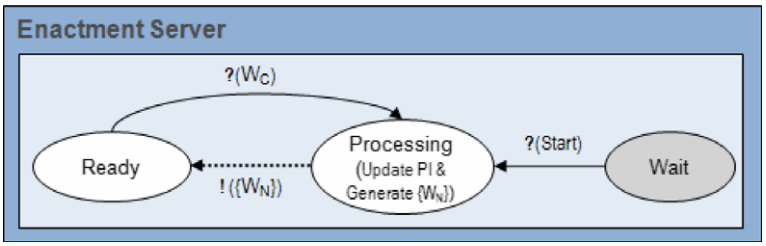


Fig. 4. DEVS model of the enactment server.

Now the Manager waits in the 'Wait for τ_L' state until it receives the *local next-event times* (τ_L) from all Simulators (i.e., $n \equiv N$) while storing the value of each τ_L in the list of local next-event time LLT[j]. Then, the Manager selects a Simulator j^* whose τ_L is the smallest designating it as the global next-event time (τ_G), and moves to the 'Wait for W_C' state after broadcasting τ_G and j^* to all Simulators. Finally, if a completed workitem W_C is received from the selected Simulator j^*, the Manager sends the W_C to the Enactment Server and moves to the 'Wait for first W_N' state to start a next cycle of enactment service.

3.3. DEVS modeling of participant simulators

A typical participant of a BPM system may be regarded as a **single server system** in which arriving jobs (i.e., new workitems) are put into a queue and a job is selected from the queue for processing based on a job selection rule (e.g., FIFO rule and shortest processing time rule). Shown in Fig. 6 is a *coupled DEVS* model of a single participant simulator consisting of three atomic DEVS models: ***Coordinator***, ***Queue***, and ***Processor***. As the DEVS model is self-explanatory, additional explanations are omitted.

If a participant of a BPM system is a group of people, the participant simulator becomes a **multiple server system**. A coupled DEVS model for a group participant simulator is presented in Appendix B. In fact, the group participant simulator corresponds to one of the three cases of handling group participant that are specified by the Workflow Management Coalition [12]. Thus, we need yet to develop group participant simulators covering the remaining two cases. If all the participants are an infinite-capacity machine system, we may not need to define participant simulators, in which case the *conversion method of workflow simulation* may be a suitable choice. For example, a PDM (or PI) may be easily converted to a formal model such as Petri-net [7], and a sequential (i.e., non-parallel) simulation is carried out. For a sequential execution of a Petri-net model, the activity scanning method [13] is widely employed.

3.4. Architecture of workflow simulation system

An existing BPM system equipped with a synchronization manager and participant simulators may be used as a **build-time workflow simulator**. As depicted in Fig. 7(a), a *process instance generator* is also added to the simulation system, which is used in generating process instances {PI*} from the process definition models (PDM) defined at Process Designer of the BPM system. Here, Enactment Server provides *'virtual' enactment services* to Participant Simulators through Synchronization Manager. Architecture of **run-time workflow simulator** is shown in Fig. 7(b). At any point of time, a copy of the 'real' process instances {PI} is obtained and it is converted into 'virtual' process instances {PI*}. Enactment Server then provides 'real' enactment services to Work-list Handlers with {PI} and 'virtual' enactment services to Participant Simulators with {PI*} simultaneously. In the figure, only the work-list handlers for (human) participants are shown, but the application handlers

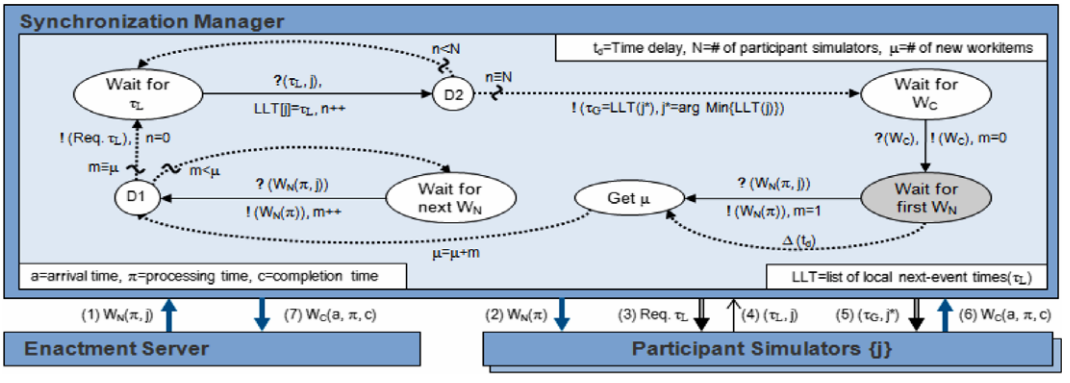


Fig. 5. DEVS model of the synchronization manager (mediator).

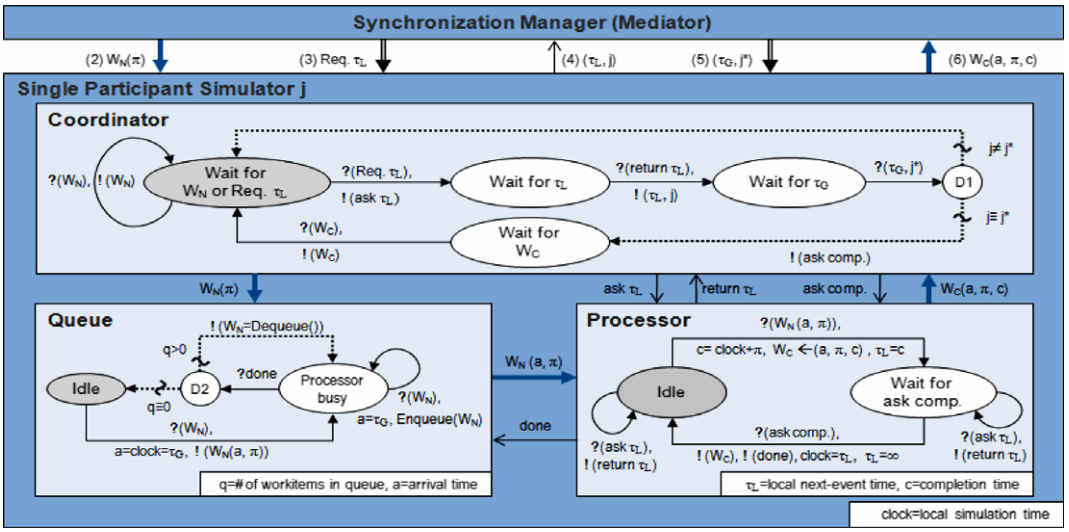
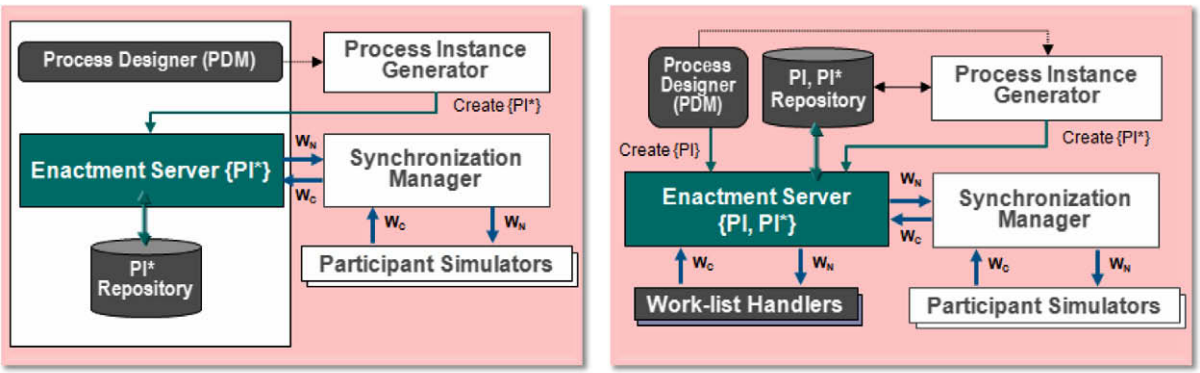


Fig. 6. DEVS model of 'single' participant simulator.



(a) Build-time workflow simulator (b) Run-time workflow simulator

Fig. 7. Architecture of direct workflow simulation systems.

[12] for IT applications can also be replaced by application simulators. In this case, the term 'performer simulator' may be used (instead of participant simulator) to cover both human participants and IT applications.

As will be described in Section 4, the build-time simulator architecture has been implemented. Implementing the run-time simulator architecture would require additional effort. Another approach to developing a run-time workflow simulator is to use two enactment servers, one for real enactment service and the other for virtual enactment service [29].

3.5. Direct workflow simulation involving multiple BPM systems

The model of the synchronization manager presented in Fig. 5 may be extended to cover a workflow simulation involving a number of BPM systems. As BPM systems are widely adopted in various units of an enterprise as well as in different companies of a value chain, there might be a growing need for workflow simulation involving multiple BPM systems to support collaborations among participating organizations.

Fig. 8 shows a conceptual framework of direct workflow simulation involving multiple BPM systems. The enactment server (ES) and a set of participant simulators (PS) are connected to a single mediator. This framework is an extension of the build-time simulator architecture depicted in Fig. 7(a). Conceptually, the mediator (i.e., synchronization manager) for multiple BPM systems should be the same as the one given in Fig. 5, but in order to implement this framework, a detailed scenario for collaborative BPM has yet to be specified by considering all the interoperability cases defined in the WfMC standards [10]. Moreover, further research is needed to realize a run-time workflow simulator involving multiple BPM systems.

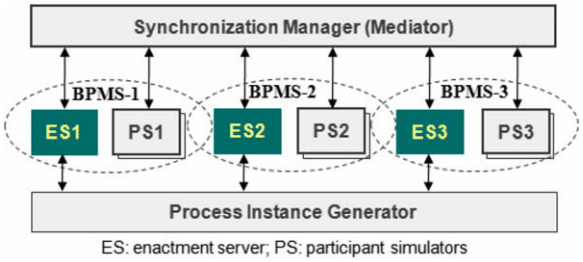


Fig. 8. Framework of direct workflow simulation involving multiple BPM systems.

4. Implementation and illustrative example

4.1. Implementation of build-time workflow simulator

The build-time workflow simulator depicted in Fig. 7(a) has been implemented as a prototype workflow simulator using a commercial BPM system [1] and an academic BPM system [34] both of which are in compliance with the WfMC standards [11]. The software structure of the prototype simulator is shown in Fig. 9. The synchronization manager and single and group participant simulators have been developed under a Microsoft .NET Framework 3.5 environment using the C# programming language, and they are plugged into the BPM systems via a *workflow engine connector* module. Also included in the prototype system are a *process instance generator* module for creating process instances (i.e., jobs) and a *progress monitor* module for monitoring the progress of simulation executions. Both installation files of the prototype workflow simulator are made public, and they may be downloaded from http://bpm.kaist.ac.kr. (The workflow simulator for the academic BPM system [34] is made available mainly for international readers because the commercial BPM system [1] does not fully support English.)

The class diagram of proposed workflow simulation system (excluding GUI classes) is presented in Fig. 10. The structure of the class diagram is exactly the same as that of the DEVS model shown earlier in Figs. 5 and 6. In the DEVS model, the *atomic model* synchronization manager (SM) is connected to the *coupled model* participant simulator (PS) consisting of atomic models coordinator (*C*), queue (*Q*) and processor (*P*). The state variables of the atomic models are defined as enumeration-type variables: SMState, CState, QState, and PState. Also, the external and internal events are defined as enumeration-type variables: SMExternalEventType, PSExternalEventType, and PSInternalEventType. Each model (atomic or coupled) is represented by a class whose main function is to handle the external/internal events and update its state variables: external events (i.e., messages from other atomic models) and internal events are handled by ExternalEventHanlder() and InternalEventHandler(), respectively. In order to communicate with the workflow engine (i.e., enactment server), the synchronization manager has an instance of class of *workflow engine connector* module which provides necessary API functions [11]. (Major API functions used in implementing the proposed workflow simulation system are listed in Appendix A.)

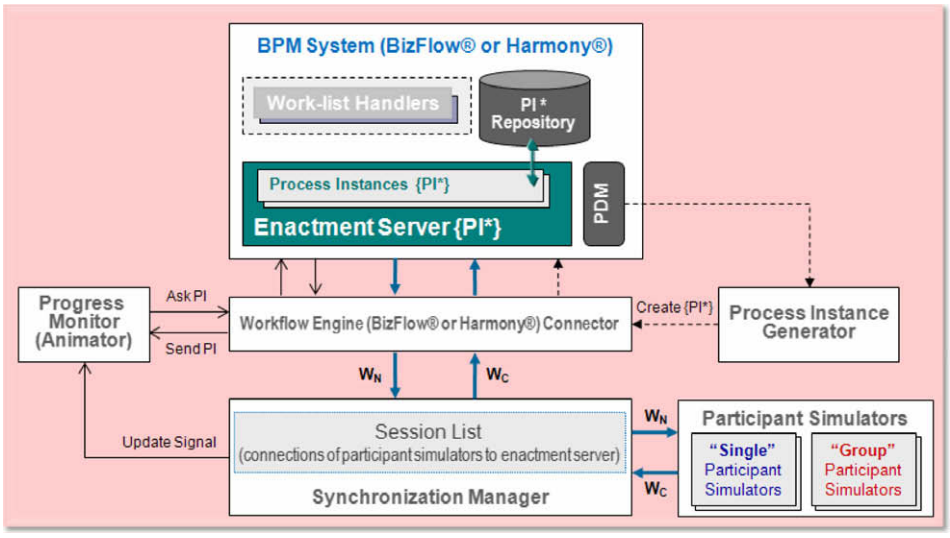


Fig. 9. Software structure of the build-time workflow simulator given in Fig. 7(a).

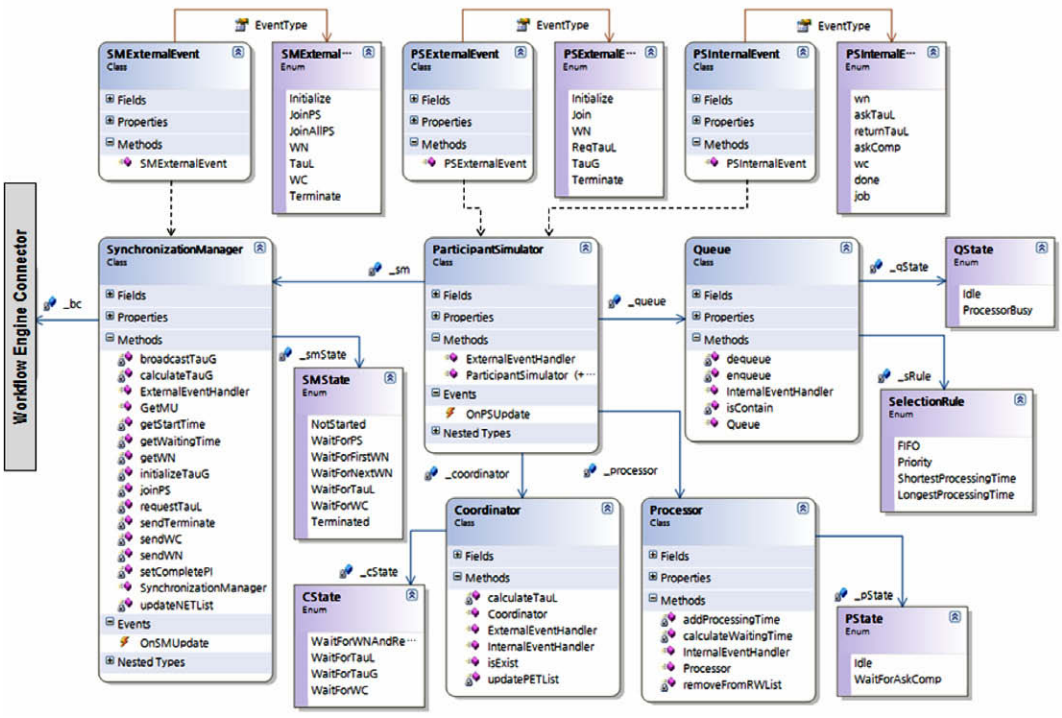


Fig. 10. Class diagram of proposed workflow simulation system.

4.2. Illustrative example

In order to test the prototype system, workflow simulation runs were made with a workflow data reported in the literature [16]. It is a bidding/contract process workflow consisting of 13 activities (see Appendix C). Among the 13 activities, five are performed by three single participants (CP Manager, EVP, and CEO) and eight by two group participants (Bid/Contract Division and Estimation Division). There are two people in each group. Also documented in the literature are mean processing time (π) of each activity.

Fig. 11 shows the PDM (process definition model) of the bidding/contract process, which was defined with the Process Designer module of the BPM system. The organizational structure (i.e., participants) was defined by the Organization Modeler module of the BPM system (not shown). When defining each activity in the PDM, its processing time and participant are

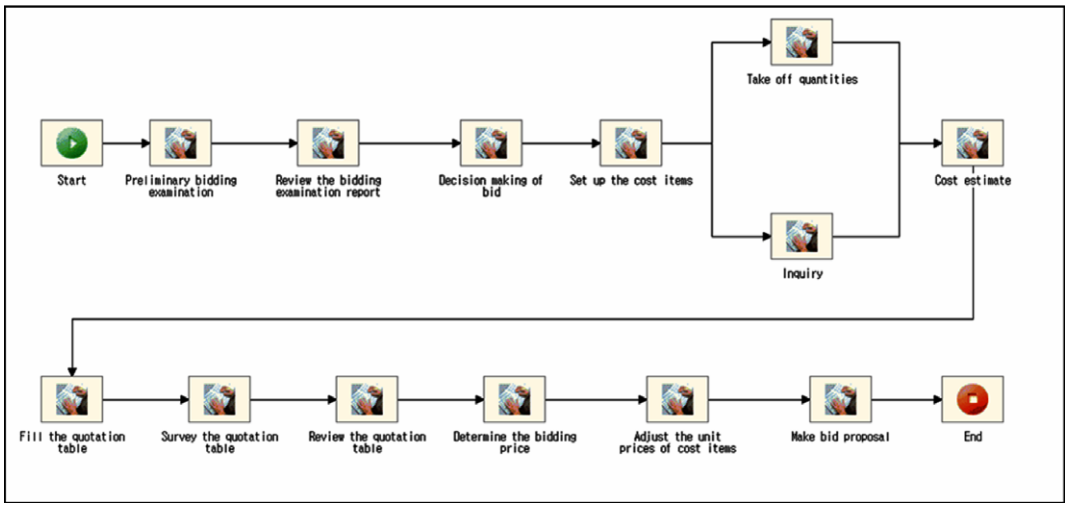


Fig. 11. PDM of the bidding/contract process.

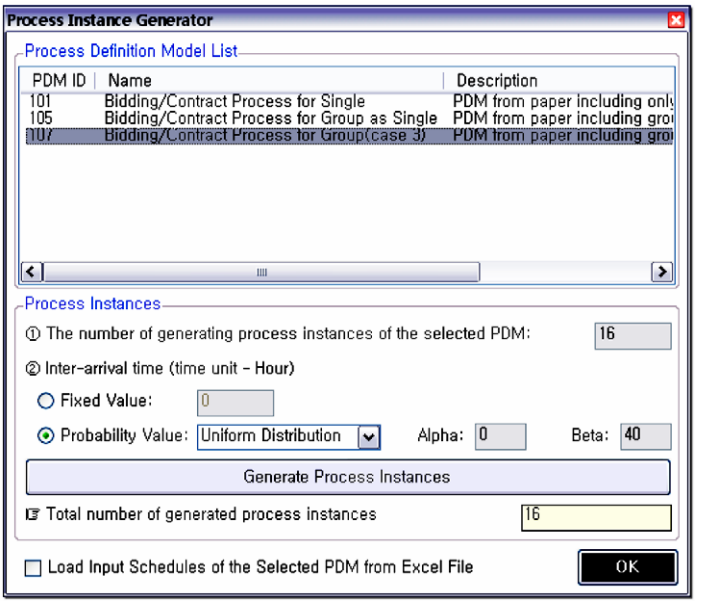
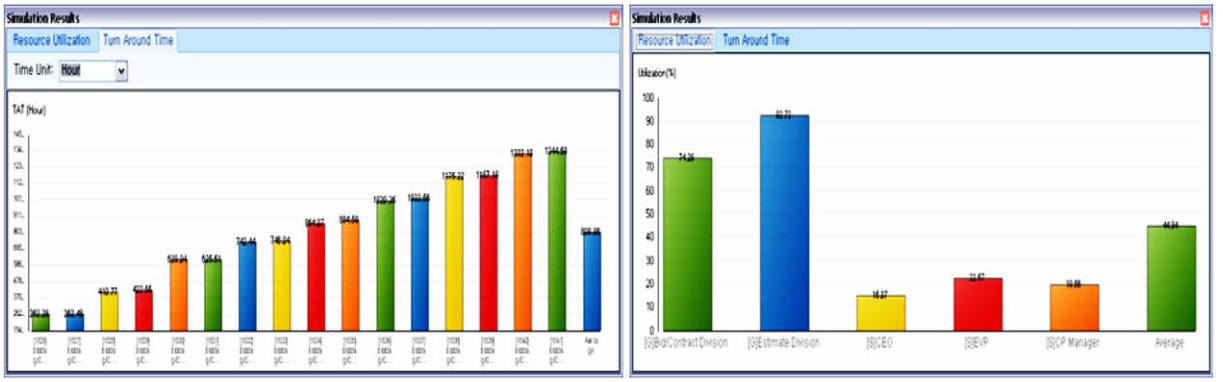


Fig. 12. GUI of process instance generator.

specified. For each participant (single or group), a participant simulator module is constructed as a session and it is connected to the Enactment Server via the Synchronization Manager with the Workflow Engine Connector. Also different types of PDMs may be generated and stored with their ID.

Fig. 12 is the GUI of the Process Instance Generator (see Fig. 9) in which three types of PDMs are shown with ID numbers 101, 105, and 107. With the Process Instance Generator, a number of process instances of a given PDM type may be generated with specified 'inter-arrival' times. In order to see if the workflow simulation is properly executed, 16 process instances for PDM-107 were generated with inter-arrival times following a uniform distribution with range from 0 to 40. The *critical path time* of the PDM is 243.49 h meaning that a job (i.e., process instance) can be completed in 243.49 h if there are unlimited resources (i.e., participants). Indeed, the turn-around time was found to be 243.49 h when a single process instance was executed.

Shown in Fig. 13 are some statistics collected from the simulation run with 16 process instances. Fig. 13(a) is a GUI of the Progress Monitor module showing the turn-around times of the 16 jobs (i.e., process instances). Fig. 13(b) shows the utilization of each participant. In this particular case, the average turn-around time is 808.86 h (with a minimum of 262 h for PI#1 and a maximum of 1344 h for PI#16) the average resource utilization is 44.84%. Considering the critical path time of 243.49 h, one may conclude that the bidding/contract process is heavily unbalanced. That is, there are severe job delays while the resources are underutilized. However, it should be noticed that the simulation results show only the behavior of the system during an initial transition period.



(a) Turn-around time of process instances

(b) Resource utilization of participants

Fig. 13. GUI of the workflow simulation system: simulation results.

For a rigorous simulation analysis, steady-state simulation runs need to be made to observe key system performances such as the distribution job turn-around times, the profile of pending jobs, the profile of pending workitems for each participant, and the utilization of each participant. We skip this part because it is not the main concern of this paper.

5. Conclusions and discussion

In this paper, we proposed a *mediator approach* to direct workflow simulation so that software modules required for workflow simulation can be plugged in an existing BPM system without any modification of workflow engine. In the proposed approach, the synchronization manager mediates all interactions between the enactment server and the participant simulators. Detailed DEVS models of the enactment server and (single and group) participant simulators are presented. Also presented are architectures of build-time and run-time workflow simulators, together with a class diagram for the build-time simulator. The build-time workflow simulator has been implemented with a commercial BPM system (as well as with an academic BPM system) and tested with a workflow data reported in the literature. The paper has demonstrated that (1) workflow models (i.e., PDMs and process instances) of a BPM system can be directly simulated by using the workflow engine (or enactment server) itself without any modification of the workflow models and (2) the workflow simulation system can be built without any internal modification of the BPM system. Further, it is postulated that the proposed approach may easily be extended to 'collaborative' workflow simulation involving multiple BPM systems.

However, further developments are needed to make the proposed approach fully operational in the commercial BPM world. First, as mentioned earlier, we need to develop group participant simulators covering the remaining two cases of handling group participant that are specified by the Workflow Management Coalition [12]. Second, in order to provide the run-time simulation functionality to a commercial BPM system, the run-time simulator architecture of Fig. 7(b) needs to be implemented.

Another line of workflow simulation research that deserves further investigation is 'collaborative' workflow simulation involving multiple BPM systems. For this purpose, a detailed scenario for collaborative BPM has to be specified by considering all the interoperability cases defined in the WfMC standards [10]. The mediator approach proposed in the paper is expected to play a key role in collaborative workflow simulation, but this has yet to be demonstrated.

Acknowledgment

This work was supported by the Korea Science and Engineering Foundation (KOSEF): No. R01-2006-000-11118-0.

Appendix A:. API functions used in the direct workflow simulation system

Major API functions [11] used in implementing the proposed workflow simulator (Fig. 9) are given in Table A1. In particular, the WAPI-2 function ListWorkitems was used in building the function GetMu() with which the value of μ (*the number of newly generated workitems*) is obtained. As described in Section 3.2, the synchronization manager has to know the value of μ at the beginning of each enactment service cycle. The function ListWorkitems(*j*) returns the number of workitems generated for Participant (Simulator) *j*. A pseudo code of the GetMu function is listed below.

```
public int GetMu()
{
  int mu = 0;
  for (int i=0; i<sessionList.Count; i++) {
    Filter worklistFilter;
    Collection workitems = sessionList[i].ListWorkItems(worklistFilter);
    for (int j=0; j<workitems.Count; j++) {
      //check if the workitem has already been passed to the pertinent simulator
      bool isSent = false;
      for (int k=0; k<sentWorkitems.Count; k++) {
        if (sentWorkitems[k] == workitems[j]) {
          isSent = true;
          break;
        }
      }
      if(isSent == false) mu ++;
    }
  }
  return mu;
}
```

Table A1
Standard API functions used in implementing the workflow simulator in Fig. 9.

Category	API name (WfMC interface 2)	Description
Connection	WMConnect() WMDisconnect()	Session establishment: return session information Disconnect session
Process control	WMCreateProcessInstance() WMGetProcessInstanceAttributeValue() WMAssignProcessInstanceAttribute()	Create a new process instance Get relevant data Update relevant data
Process status	WMOpenProcessInstanceList() WMGetProcessInstance()	Query process instances Get a process instance
Worklist	WMGetWorkItem() WMCompleteWorkItem()	Get a wonVitem Complete a workitem
WAPI-2	ListWorkitems()	Used for getting μ value

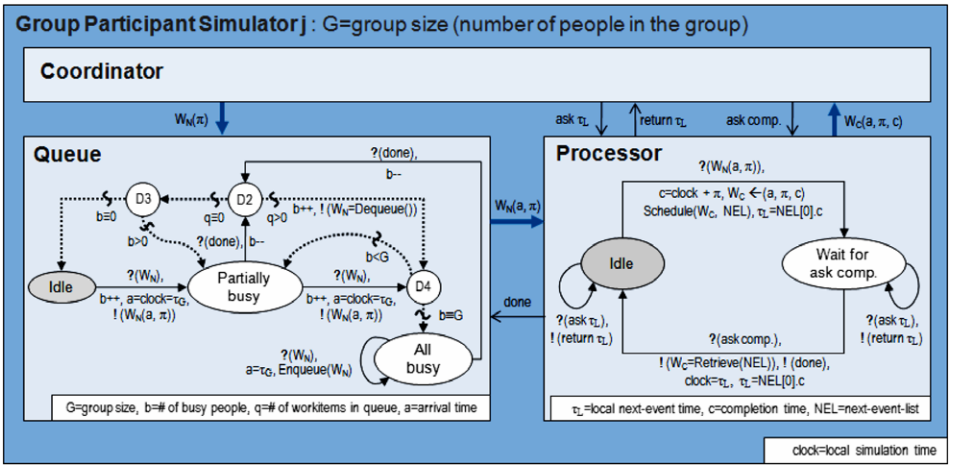


Fig. B1. DEVS model of a 'group' participant simulator.

Appendix B:. DEVS model of a group participant simulator

Fig. B1 shows a coupled DEVS model of the group participant simulator mentioned in Section 3.3. The **Coordinator** model of the group participant simulator is the same as that of the single participant simulator of Fig. 6. The **Queue** model now has three states Idle (no one in the group is busy), Partially-busy and All-busy (all the people in the group are busy), and state transitions are modified accordingly. The atomic DEVS model of the Queue should be self-explanatory (if the reader understood the single participant Queue model of Fig. 6), where G = group size (number of people in a group) and b = number of busy people.

The **Processor** model now maintains a *next-event list* (NEL) in which all the *'scheduled' completed workitems* $\{W_C(a, \pi, c)\}$ assigned to the group are stored. NEL is a priority queue where $\{W_C\}$ are sorted in an ascending order of their completion time c. Every time a new workitem $W_N(a, \pi)$ is received from the Queue, a completed workitem $W_C(a, \pi, c)$ is generated after computing its expected completion time c and then it is 'scheduled' into NEL. If a 'ask completion' message is received from the Coordinator, the first record of NEL is 'retrieved' and sent to the Coordinator. The rest of the operations in the Processor model should be easily understood.

Appendix C:. Details of the activities in the PDM shown in Fig. 11 [16].

No.	Name	Participant type	Participant name	Processing time: π (Time unit: hour)
1	Preliminary bidding examination	Group	Bid/contract Division	10.00
2	Review the bidding examination report	Single	CP Manager	9.92
3	Decision making of bid	Single	EVP	6.00

(*continued on next page*)

Appendix C (*continued*)

No.	Name	Participant type	Participant name	Processing time: π (Time unit: hour)
4	Set up the cost items	Group	Bid/contract division	19.00
5	Take off quantities	Group	Estimate Division	68.00
6	Inquiry	Group	Bid/contract division	94.81
7	Cost estimate	Group	Estimate division	21.00
8	Fill the quotation table	Group	Estimate division	3.36
9	Survey the quotation table	Single	CP Manager	7.00
10	Review the quotation table	Single	EVP	13.52
11	Determine the bidding price	Single	CEO	13.04
12	Adjust the unit prices of cost items	Group	Bid/contract division	2.68
13	Make bid proposal	Group	Bid/contract division	1.97

References

[1] Bizflow®, Handysoft, 2008. <http://www.handysoft.com>.
[2] Delphi, BPM 2005 Market Milestone Report, 2005. <http://www.delphigroup.com/research/whitepaper_request_download.htm>.
[3] Gartner, Magic Quadrant for Business Process Management Suites, 2006.
[4] R.M. Fujimoto, Parallel and Distributed Simulation Systems, John Wiley & Sons, 2000.
[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[6] F. Leymann, D. Roller, Production Workflow Concepts and Techniques, Prentice Hall, 2000.
[7] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981.
[8] D.L. Schruben, L.W. Schruben, Event Graph Modeling Using SIGMA, Custom Simulations, 2001.
[9] H. Smith, P. Fingar, Business Process Management: The Third Wave, Meghan-Kiffer Press, 2003.
[10] WfMC-TC-00-1003, The Workflow Reference Model, Workflow Management Coalition, 1995. <http://www.wfmc.org>.
[11] WfMC-TC-1009, Workflow Management Application Programming Interface Specification, Workflow Management Coalition, 1998. <http://www.wfmc.org>.
[12] WfMC-TC-1025, Process Definition Interface: XML Process Definition Language, Workflow Management Coalition, 2008. <http://www.wfmc.org>.
[13] B. Zeigler, H. Praehofer, T. Kim, Theory of Modeling and Simulation, Academic Press, Boston, 2000.
[14] W.M.P. Aalst, A.H.M. Hofstede, Verification of workflow task structures: a petri-net-based approach, Information Systems 25 (1) (2000) 43–69.
[15] J.S. Bae, S.C. Jeong, Y.H. Seo, Y.H. Kim, S.H. Kang, Integration of workflow management and simulation, Computers and Industrial Engineering 37 (1999) 203–206.
[16] M.Y. Cheng, M.H. Tsai, Z.Y. Xiao, Construction management process reengineering: organizational human resource planning for multiple projects, Automation in Constructions 15 (6) (2006) 785–799.
[17] M.N. Davies, A generic model for simulating office process flows, European Journal of Operational Research 99 (2) (1997) 267–277.
[18] G. Gans, M. Jarke, G. Lakemeyer, D. Schmitz, Deliberation in a metadata-based modeling and simulation environment for inter-organizational networks, Information Systems 30 (2005) 587–607.
[19] G.M. Godefridus, G. Siebren, J.M. Henk, Workflow management: changing your organization through simulation, Accreditation and Quality Assurance 4 (9/10) (1999) 438–442.
[20] A. Greasley, S. Barlow, Using simulation modeling for BPR: resource allocation in a police custody process, International Journal of Operation and Production Management 18 (1998) 978–988.
[21] A. Greasley, Using business-process simulation within a business-process reengineering approach, Business Process Management Journal 9 (4) (2003) 408–420.
[22] H.C. Hwang, B.K. Choi, Workflow-based dynamic scheduling of job shop operations, International Journal of Computer Integrated Manufacturing 20 (6) (2007) 557–566.
[23] V.K. Pandya, S. Nelis, Requirements for process redesign tools, International Journal of Computation and Applied Technology 11 (1998) 409–417.
[24] V. Peer, W. Brigitte, A simulation-based decision support system for business process planning, Fuzzy Sets and Systems 125 (2002) 275–287.
[25] L.W. Schruben, Simulation modeling with event graphs, Communications of the ACM 26 (1983) 957–996.
[26] G. Zacharewicz, C. Frydman, N. Giambiasi, G-DEVS/HLA environment for distributed simulations of workflows, SIMULATION: Transactions of the Society for Modeling and Simulation International 84 (5) (2008) 197–213.
[27] A.H. Buss, Component-based simulation modeling, in: Proceedings of the 2000 Winter Simulation Conference, 2000, pp. 964–971.
[28] D.Y.K. Chan, Design an effective workflow in simulation, in: The Eighth International Conference on Computer Supported Cooperative Work in Design Proceedings, vol. 2, 2004, pp. 324–328.
[29] B.K. Choi, D.W. Lee, D.H. Kang, DEVS modeling of run-time workflow simulation and its application, in: Proceedings of the 22nd European Conference on Modeling and Simulation, Cyprus, 2008.
[30] K.J. Hong, J.K. Lee, D.H. Kim, T.G. Kim, DEVS framework instrumented with database for web-based workflow modeling simulation, Society for Computer Simulation (Simulation Councils, Inc.) 31 (3) (2003) 113–118.
[31] J.J. Li, F. Casati, M.C. Shan, Business process simulation with HP process manager, Simulation Series 34 (1) (2002) 291–295.
[32] A. Rozinat, M.T. Wynn, W.M.P. Aalst, A.H.M. Hofstede, C.J. Fidge, Workflow simulation for operational decision support using design, historic and state information, in: Proceedings of the Sixth International Conference on Business Process Management, Italy, 2008.
[33] A. Swami, Building the business using process simulation, in: Proceedings of the 1995 Winter Simulation Conference, 1995.
[34] B.K. Choi, H.C. Hwang, Architecture of A 3-Layered Closed-loop BPMS Harmony®, KAIST VMS Lab Technical Report, 2005. <http://vms.kaist.ac.kr/publications/TechnicalReport/3layer.pdf> (in Korean), The S/W is Available at <http://bpm.kaist.ac.kr>.
[35] G.P. Hong, DEVS-Based Framework for Logical and Performance Analysis of Discrete Event Systems, Doctoral Thesis, Department of Electronic Engineering, KAIST, 1997.