Inf Syst Front (2007) 9:103–118
DOI 10.1007/s10796-006-9020-5

Enhancing the efficiency of supply chain processes through web services

Seung-Hyun Rhee · Hyerim Bae · Yongsun Choi

Published online: 22 November 2006

Abstract Among business enterprises, keen competition is accelerating the introduction of Supply Chain Management (SCM). SCM entails the utilization of cutting-edge information technology in elaborately designing, managing and integrating supply chain processes so that participating companies' processes are interoperational at the global level. Recently, new Business Process Management (BPM) technology has attracted much attention as an indispensable tool for managing the supply chain synthetically and systematically. This next-generation technology, a great advance on existing BPM systems, can greatly enhance overall process efficiency in run-time. The critical path(s) in a process, and the slack time of each task, being the typical determiners of supply chain process efficiency, are the basis of a method, proposed in this paper, of efficiently executing global supply chain processes. The proposed method acts as a dispatching rule that can guide prioritization of the tasks in order to improve the run-time efficiency of supply chain processes. We provide several simulated results to demonstrate the effectiveness of our method, propose a web-service-based system architecture for the communication of the run-time data of tasks among processes in heterogeneous environments, and present a prototype of the system implemented.

Keywords Business Process Management · Supply Chain Management · Process efficiency · Process interoperability · Dispatching rule · Web service

S.-H. Rhee · H. Bae (✉) · Y. Choi
Department of Industrial Engineering, Pusan National University, Pusan 635-709, Republic of Korea
e-mail: hrbae@pusan.ac.kr

1 Introduction

In rapidly revolutionizing business environments, collaboration with partners is considered an essential element of success (Hammer, 2001), because the competitiveness of a company is derived from the entire scope of business activity in delivering products to end users. Such collaboration is achieved by systematic interfaces among business partners, to the overall purpose of increasing customer satisfaction (Kobayashia, Tamakia, & Komodab, 2003; Seshasai, Gupta, & Kumar, 2005).

Supply Chain Management (SCM) is a systematic method of collaborative management among companies (Kobayashia et al., 2003; D. Simchi-Levi, Kaminsky, & E. Simchi-Levi, 2000). SCM requires a systematic approach to procedures, to the overall process, occurring in the supply chain. In this paper, that process is called the Supply Chain Process (SCP). Business Process Management (BPM), which is widely accepted as an effective managing and executing business process tool (Latimer, 2004), is an integrated approach to management of the SCP. BPM is a general methodology or a software system that increases company productivity through the systematic design, management, integration and improvement of business processes (Rhee et al., 2005; Smith & Fingar, 2003; Stohr & Zhao, 2001). BPM facilitates interoperations among heterogeneous systems, and sometimes serves as a platform for developing process-based applications (Rhee et al., 2005).

The introduction of BPM to manage an SCP provides several benefits. First, each stage of the SCP, and the interrelations among stages and participants, are defined clearly. In addition, BPM monitors and measures the performance of SCP execution (McCoy, 2004), making it possible to improve the SCP by analyzing the process

performance results (Smith & Fingar, 2003). Existing BPMs can assure that an individual process is executed in the proper order in exact accordance with the process model (Pozewaunig, Eder, & Liebhart, 1997; Zhao & Stohr, 1999). However, this does not guarantee that all of the processes proceed efficiently (Rhee, Bae, Ahn, & Seo, 2003), which is a very important issue in the SCM field. This is the BPM dilemma, and it demands a resolution.

Accordingly, in this paper we propose, from the perspective of overall process efficiency, a novel method of managing the execution of the supply chain process with BPM technologies in the web service environment. We expand the PERT/CPM (Program Evaluation & Review Technique/Critical Path Method) technique to handle the selective execution of path(s) and the nested processes. We suggest a prioritization rule of executing tasks based on the slack time of each task computed and demonstrate the effectiveness of our method with several simulated results. We propose a web-service-based architecture to deliver the run-time data of tasks among processes, and, finally, we introduce a prototype of the implemented system.

2 Background

2.1 BPM and process efficiency

Business Process Management Systems (BPMS) have been employed as a tool to overcome interoperability barriers among heterogeneous systems and to collaborate with partners effectively. That is, BPM technology, especially combined with web service architecture, is generally acknowledged to be useful in solving technological problems that occur when composing a global process from heterogeneous business processes, each of which is executed independently by a participating company (Kobayashia et al., 2003). With multiple companies interacting in such global processes, SCM is a typical and recurring problem area of BPM integration.

Generally, BPMS handle a business process as an object that can be managed in a digital form. The main functions of BPMS are conceived as being related to two distinct phases of build-time and run-time of these process objects (J. Bae, H. Bae, Kang, & Kim, 2004; Rhee, Bae, & Kim, 2004; Workflow Management Coalition, 1999a). In build-time, each *process model* is designed so that the process is ready for execution (van der Aalst, Weske, & Grunbauer, 2005; Workflow Management Coalition, 1999b). Activities including their attributes and precedence relations, among the activities in each process, are defined during this phase, and these will be explained further in the subsequent section. In run-time, *process instances* that will actually be performed are created based on the build-time process models (Workflow Management Coalition, 1999b). Once a process model is prepared, process instances are generated repeatedly whenever they are needed (van der Aalst et al., 2005). The controlled execution as well as the creation of process instances is carried out by the BPM *engine* (Y. Kim, Kang, D. Kim, Bae, & Ju, 2000). Since a typical BPMS deals with a large number of process models at the same time and each process model generates multiple instances, a task performer will have many tasks to do at a certain point in time. The average execution time of a task performed by a separate system is negligible compared with those of human-performed tasks. Therefore, we will consider only the set of tasks assigned to a human user, called a *worklist* (Workflow Management Coalition, 1999b).

The efficiency of business processes has two perspectives. One is the performance of the BPM engine that carries out management of the process lifecycle, and the other is the capacity of users that handle tasks assigned by the BPM engine. The former is related to scheduling a process model and assigning tasks considering the status of the model (Bae et al., 2004; Kumar & Zhao, 2002). The latter is related to providing information on the respective priorities of the user's tasks, and this is the issue to be treated in this paper.

2.2 Related studies on process efficiency

A large amount of research has been conducted in the area of efficient production processes in relation to scheduling and assigning problems (Baker, 1974; Holthaus & Rajendran, 1997; Hu, Minciardi, Paolucci, & Pesenti, 1992; Lu & Kumar, 1991; Park, Raman, & Shaw, 1997; Rajendran & Ziegler, 2001). However, in a business process environment, where processes are automated and controlled by software systems such as WfMS (Workflow Management Systems) or BPMS, the efficiency issue has not been a major topic (Pozewaunig et al., 1997; Rhee et al., 2003). Most research in the area of BPM including workflow has placed a disproportionate emphasis on process modeling, process interoperability, relevant applications, and related issues (Chang, Son, & Kim, 2002; Klein & Dellarocas, 2000; Pozewaunig et al., 1997; Rhee et al., 2003; Zhao & Stohr, 1999). However, as the areas in which automated business processes take place have expanded, the run-time efficiency of the business processes have received more attention (Son & Kim, 2001). In this section, we briefly review the previous work on process efficiency.

Pozewaunig et al. (1997) noting that time management is hardly supported in workflow systems, provide a concept for managing time in workflow processes. The proposed concept consists of calculating internal deadlines for all activities within a workflow process, checking time constraints, and monitoring time in run-time. It is notable that they extend the

PERT technique to support the structures usually found in workflows. Chang et al. (2002) from a similar point of view and with the purpose of contributing to the management of time and resources within a single workflow process, also support the identification and analyses of a critical path. However, both Pozewaunig and Chang simply deliver time information, and do not explain how this information is used systematically to improve the process.

Zhao, going further, introduces a concept of process efficiency in Zhao & Stohr (1999). He uses time-driven processes to predict turnaround time and to allocate expected processing time to each activity in the process, and then explores some traditional dispatching rules. Although he offers a sound conceptual foundation, he fails to fully demonstrate the effectiveness of the approach for process efficiency. Rhee et al. (2004), improving on Zhao, propose a practical approach to improving process efficiency by calculating slack time and finding a critical path based on modified PERT/CPM algorithms. They introduce the LST (Least Slack Time) rule and show how it can increase process efficiency. However, their approach is limited to single process models and cannot be applied to complex processes, such as SCP, that contain nested processes.

All of the previous research forms a robust foundation for this paper, in which we focus on the efficiency of SCPs managed by the BPM system. In so doing, we extend the LST Rule, originally proposed in Rhee et al. (2004) where it was applied only to a single process.

3 Representation of the supply chain processes

3.1 The SCP model

For the effective implementation of the SCP, it is prerequisite to represent the process in a computer-understandable form. In this section, we represent the SCP as a *process model*. The process model specifies the tasks and the relations among processes. The definition of the process model is presented below.

Definition 1 (Basic Process Model: Process, Activity, Link, Split and Merge)

A process structure is defined as a directed graph $P=(A_P, L_P)$ and a labeling function $f(\bullet)$ for split or merge types, such that

- $A_P=\{a_i \mid i=1,\ldots, N\}$ is the set of activities, where a_i is the ith activity and N is the total number of activities in P;
- $L_P \subseteq \{(a_i, a_j) | a_i \in A_P, a_j \in A_P$, and $i \neq j\}$ is the set of links, where an element (a_i, a_j) represents that a_i immediately precedes a_j.
- For a split activity a_j, such that $|S|>1$, where $S=\{a_k|(a_j, a_k)\in L\}$, $f(a_j)$='AND' if all a_ks should be executed; otherwise, $f(a_j)$='OR.'
- For a merge activity a_j, such that $|M|>1$, where $M=\{a_i|(a_i, a_j)\in L\}$, $f(a_j)$='AND' if all a_is should be executed; otherwise, $f(a_j)$='OR.'

However, to represent the SCP, it is not enough to define the above basic process model. A nested process model, as follows, must also be considered.

Definition 2 (Nested Process)

A second independent process is often associated with a process P. That second process is expressed as a set of constituent activities of P. Thus, the independent process is dealt with as an activity (a_{P_s}) as well as a process (P_s). This process is called a *nested process*, which is defined below.

- $P_s = (A_{p_s}, L_{p_s})$, $P_s \in A_P$ is a sub-process that belongs to process P, where A_{p_s} is the set of activities and L_{P_s} is the set of links.

The concept of the nested process is very useful to the design of a process model. If some parts of the process model already exist as an independent process, they can be modeled by reusing that independent process, and in the newly designed process they are expressed as a nested process. In a practical SCP where several partners interoperate, it is also reasonable to regard the internal process of the partner as a nested process. In information systems that digitize and manage processes, the nested process is actively employed. Nested processes strengthen companies' competitiveness by introducing the best practices of each partner industry through Business Process Outsourcing (BPO) or web services (Smith & Fingar, 2003). Effective collaboration among partners also requires the close integration of other processes (Hammer, 2001). The concept of the nested process can satisfy all such requirements from the viewpoint of a process structure.

When a process model is defined as above, multiple paths can exist between two different tasks, as in the following definition.

Definition 3 (Path)

Consider two activities a_i and a_j ($i \neq j$), in a process model $P=(A_P, L_P)$. It is possible that multiple paths exist between the tasks. When there is the pth path between them, r_p, which is a set of ordered activities such that

$$r_p = \{a_i, a_k, a_{k+1}, \ldots, a_{k+n}, a_j\}$$
$$(a_i, a_k), (a_k, a_{k+1}), \ldots, (a_{k+n}, a_j) \in L_P$$

a_j can be reached from a_i by visiting the activities in the order of the connecting links.

Generally, the complexity of the process model is determined by the form of AND structures, OR structures, and nested processes (Bae et al., 2004; Workflow Management Coalition, 1999a). For successful execution of the AND structure, all of the paths should be executed, but for the OR structure, it is enough to carry out only one of the paths. The OR structure is further classified into Normal OR (NOR), Priority OR (POR), or Conditional OR (COR) depending on the selection strategy of succeeding paths to execute (Bae et al., 2004). The types that will be performed are determined as follows.

- NOR: After the split activity, some of the branches can be executed. When one of those reaches the merge task successfully, the process proceeds to the subsequent task of the merge task while ignoring the other branches that have not yet reached the merge task.
- POR: A priority is assigned to each path. After the split activity, the path with the highest priority is performed first. If this successfully reaches the merge task, all of the other paths are ignored. Otherwise, the path with the next highest priority is tried until a successful one is found.
- COR: Each path is associated with a set of conditions. After the split task, the conditions are assessed, and only the paths that satisfy the conditions are carried out. The condition satisfaction is determined at or prior to the split task. Notice that this structure can be used as an XOR structure that allows only one path to execute.

3.2 A sample SCP (supply chain process)

Our method is devised for an SCP where two or more processes in heterogeneous environments interoperate with one another. In Fig. 1, a sample SCP is illustrated, and the process is used to describe our method presented in this paper. The SCP begins when a customer places an order to buy a large amount of product. When a customer order is received, a seller reviews the current available inventory and checks if it is sufficient to fulfill the customer order. If the current inventory amount is sufficient to fulfill the customer order, an ordinary order fulfillment is executed, and, additionally, the inventory data is updated. Based on this updated inventory data, the seller can determine when to reorder from an external manufacturer. When the seller reorders, the manufacturer, naturally, produces the product and conveys it to the seller. If, however, the manufacturer's current inventory amount is not sufficient, emergency production is initiated, the manufacturer producing the product and dispatching it to the seller. The seller, in turn, delivers the product to the customer.

As shown in Fig. 1, there are distinct processes corresponding to each of the three participating companies. The seller cooperates with two partner manufacturers who

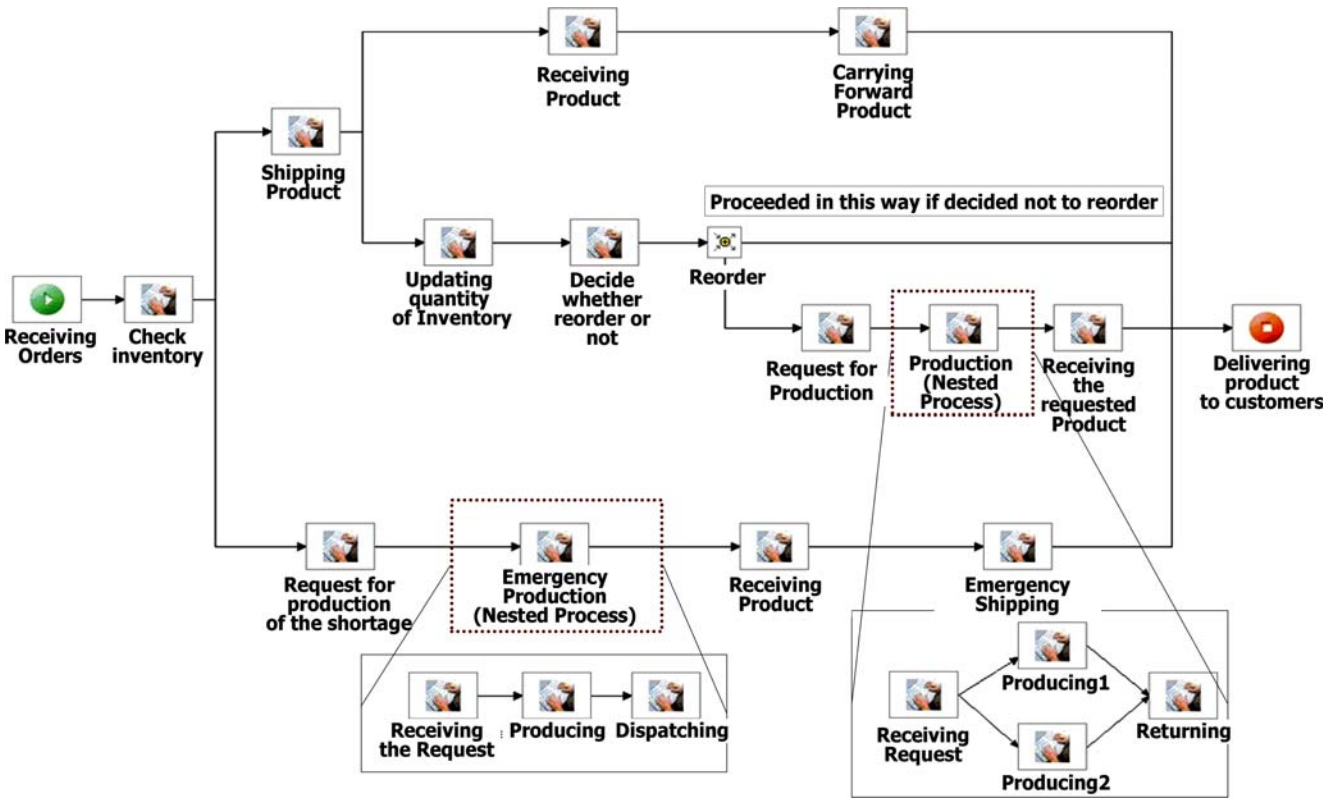


Fig. 1 A sample SCP

provide relevant product for that seller. Obviously, since the two partners' processes are performed on heterogeneous systems, they need to be integrated with the seller's process. Those two nested processes (Emergency Production, Production) are shown in the dotted square. 'Emergency Production' responds to the activity 'Request for production of the shortage' in the seller's process and requests the execution of the activity 'Receiving the Request' within the heterogeneous process. Once the activity 'Dispatching' is completed, data relevant to the execution of the nested process is delivered to 'Emergency Production,' and then 'Emergency Production' is completed. The other nested process, 'Production,' is executed in a similar way.

4 Efficient methods of SCP execution

This section presents a method of calculating the slack time of each activity in an SCP. A dispatching rule and exchanging information are also discussed.

4.1 Slack time and critical path in a basic process model

In order to furnish urgency information for each task, the critical path should be found, and the slack time should be computed. Computation of the slack time of each task is based on the PERT/CPM method, which was first proposed in a previous paper of ours (Rhee et al., 2004). Whereas a split in PERT/CPM networks always entails use of the AND semantic, workflow process structures have an alternative path (that is, an OR block is included), and we consider it as the single path presented in Fig. 2. Prior to finding the critical path and computing the slack time of tasks in an OR structure, the expectation time of each path is required. Since the AND structure is the same as that of the PERT/CPM network, we calculate the expectation time for only three representative types of OR structures; the computation of the expectation time of each type is summarized in Table 1. Any reader who requires a detailed description of the computation can refer to (Rhee et al., 2004).

The next section details the procedures for calculating the slack time in an SCP including nested processes.

4.2 Calculation of slack time in SCP and a dispatching rule

An SCP usually includes nested processes, and the highest level process containing the nested processes is called a *global process*. The calculation of the slack time for an SCP as a global process is based on that for a basic process model, described in the previous section. In this section, we describe how to find the critical path and calculate the slack time in an SCP. Our dispatching rule, conceived in accordance with the slack time, is also provided at the end of this section.

In our method, the slack time is calculated in three different cases. In the first case, the initial critical path and slack time need to be calculated before a process is executed. In the second case, the calculation is required when each activity in the global process is completed. Since the actual completion time of the activity can contribute to correctness, once an activity is completed, the slack time of all of the activities is recalculated using the exact completion time. In the third case, when an activity in a nested process is completed, the time information of all of the constituent activities of the nested process in the SCP is updated using the actual processing time.

Let us explain the first case. We use a modified version of Fig. 1 (i.e., Fig. 3) to explain our method. As shown in Fig. 3, before executing an SCP, we find the critical path and calculate, based on the expected processing time, the initial slack time of all of the activities. The two numbers shown in the parentheses for each activity in Fig. 3 indicate the expected processing time and deadline. Whereas in this chapter the expected processing time is utilized, in the experimental chapter (Chapter 5) the deadline will be employed to measure the average number of completed process instances and the average delay time for each activity. In the nested process a_{P_1}, there exist two paths, $r_1 = \{a_{1-1}, a_{1-2}, a_{1-4}\}$, $r_2 = \{a_{1-1}, a_{1-3}, a_{1-4}\}$, between activity a_{1-1} and activity a_{1-4}. The nested process is regarded as a normal activity, a_{P_1}, in global process P, and the expected

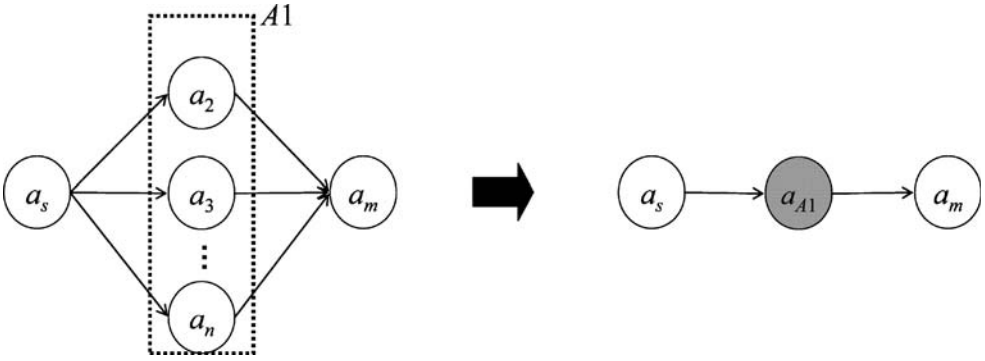


Fig. 2 A representative activity of an OR structure

Table 1 Expected processing time of each OR type

Type	Expected processing time	P_i (λ_i)
NOR	$\sum_{i=1}^{n}\prod_{j=1,j\neq i}^{n}\left(\frac{\lambda_i}{\lambda_i+\lambda_j}\right)\mathrm{ET}_i$	λ_i: The service rate with which the distribution of the actual processing time of the ith path follows an exponential distributionPOR
POR	$\begin{cases} P_1\times \mathrm{ET}_1 & ,n=1 \\ P_1\times \mathrm{ET}_1+\sum_{i=2}^{n}\left[\left\{\prod_{j=1}^{i-1}(1-P_j)\right\}P_i\left\{\sum_{k=1}^{i}\mathrm{ET}_k\right\}\right] & ,n\geq 2 \end{cases}$	$\boldsymbol{P_i}$: The probability that, when the ith path, r_I, is determined to be executed, the path is successfully completed
COR	$\sum_{i=1}^{n}(P_i\times \mathrm{ET}_i)$	$\boldsymbol{P_i}$: The probability that r_i is chosen for execution

ET_i: Expected processing time of activity a_i

processing time of the nested process at the level of the global process needs to be computed.

Let ET_{a_i} and ET_i denote the expected processing time of activity a_i and path r_i, respectively. Then, the expected processing time of a_{P_1} becomes

$$\mathrm{ET}_{a_{P_1}} = \mathrm{Max}\{\mathrm{ET}_1, \mathrm{ET}_2\}. \quad (1)$$

It is straightforward to generalize this for a nested process P_s with n paths, as follows:

$$\mathrm{ET}_{a_{P_s}} = \mathrm{Max}\{\mathrm{ET}_i | 1 \leq i \leq n,\ i \text{ is a natural number, and } n \text{ is the number of paths in } P_s\}. \quad (2)$$

Once the expected processing time of a nested process is acquired, the critical path of a global process and the slack time of its constituent activity can be calculated using the PERT/CPM algorithm. The slack time of the nested process is computed as that of a representative activity. Note that the slack times of the constituent activities in the nested process are also calculated using this slack time. Let $\mathrm{ST}_{a_i}^{P}$ be the slack time of activity a_i in process P. Then, the slack time of each activity in the nested process is computed at the global process level as follows:

$$ST_{a_{1i}}^{P} = ST_{a_{P1}}^{P} + ST_{a_{1i}}^{P_1} \quad i = 1, 2, \ldots n. \quad (3)$$

In Eq. 3, by applying the PERT/CPM algorithm, the value of $\mathrm{ST}_{a_{1i}}^{P}$ is exactly the same as the calculated slack

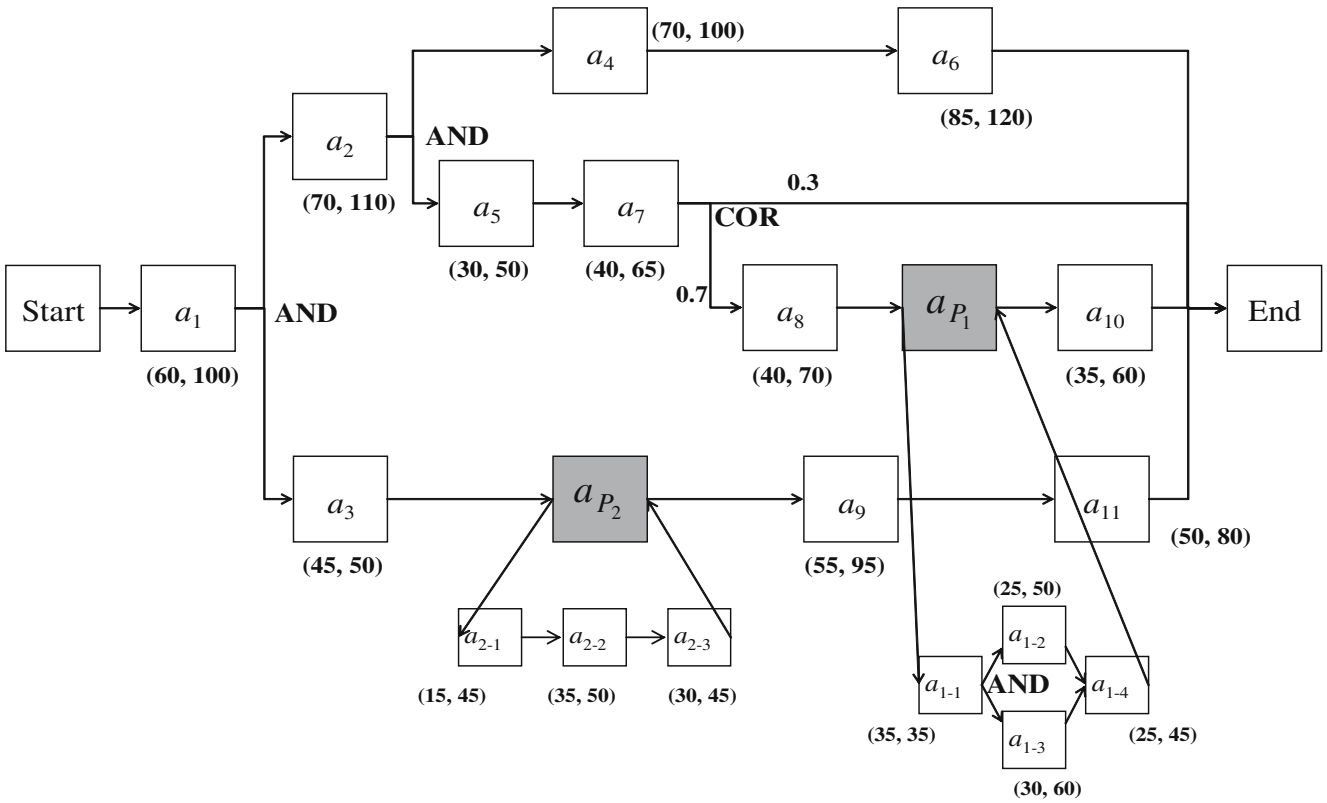


Fig. 3 The modified version for the calculation of slack time

time of each activity in P_1, assuming that that value constitutes the global process at the highest level.

The nested process and the COR structure must be transformed into their representative activities. The slack time is calculated using Eq. 2 and Table 1 as follows:

$$\begin{aligned} \mathrm{ET}_{a_{P_1}} &= \max\{35+25+25, 35+30+25\} \\ &= \max\{85, 90\} = 90; \end{aligned}$$

$$\begin{aligned} \mathrm{ET}_{\mathrm{COR}} &= \left(\mathrm{ET}_{a_8} + \mathrm{ET}_{a_{P_1}} + \mathrm{ET}_{a_{10}}\right) \times 0.7 + 0 \times 0.3 \\ &= (40+90+35) \times 0.7 = 115.5. \end{aligned}$$

$\mathrm{ET}_{a_{P_2}}$ can also have its value 80 following the same method. Accordingly, the expected processing time in build-time is computed as shown in Fig. 4.

Figure 4 presents a modified version of the original global process, where a_{P_2} is a nested process and a_{COR} is a representative of the COR block. Now, the process has three paths r_1, r_2, r_3, as follows:

$$\begin{aligned} r_1 &= \{a_1, a_2, a_4, a_6\}, \\ r_2 &= \{a_1, a_2, a_5, a_7, a_{\mathrm{COR}}\}, \\ r_3 &= \{a_1, a_3, a_{P_2}, a_9, a_{11}\}. \end{aligned}$$

Based on the expected processing time of each task, the expected processing time of the three paths is calculated as follows:

$$\begin{aligned} \mathrm{ET}_1 &= 60+70+70+85 = 285, \\ \mathrm{ET}_2 &= 60+70+30+40+115.5 = 315.5, \\ \mathrm{ET}_3 &= 60+45+80+55+50 = 290. \end{aligned}$$

Since r_2 becomes the critical path, the slack time of all of the tasks on r_2 is 0. The slack time of the representative activity a_{COR} also is 0. Although a_{COR} comprises the three activities including the nested process, the folded activity a_{COR} is maintained in build-time. Contrastingly, the slack time of all of the tasks on r_1 and r_3 is 30.5 and 25.5, respectively. We can give slack time to constituent activities in a_{P_2} using $\mathrm{ST}^P_{a_{P_2}}$, the value of which is 25.5 from the viewpoint of the global process. The slack time of activity a_{2-1} in the nested process a_{P_2} on the level of global process P is

$$\mathrm{ST}^P_{a_{2-1}} = \mathrm{ST}^P_{a_{P_2}} + \mathrm{ST}^{a_{P_2}}_{a_{2-1}} = 25.5 + 0 = 25.5.$$

In a similar way, the slack time of the remaining activities in a_{P_2} can be obtained. As a result, we can calculate the initial slack time of all of the constituent activities in build-time.

Now let us describe the second case. Once a global process P is launched by the BPM engine, a more exact slack time can be acquired using the actual processing time (Kim & Rhee, 2002). Suppose that activities a_1 and a_3 were completed earlier than expected, and that their actual processing times were 55 and 30, respectively. We also assume that activity a_2 is executing and that the other activities are not yet started. The actual processing time is employed to update the slack time of the uncompleted activities excepting the activities a_1, a_2, and a_3. We will now consider the slack time of activity a_{P_2} and its constituent activities. The expected processing time of the three paths in P is updated using the actual processing time as follows:

$$\begin{aligned} \mathrm{ET}_1 &= 55+70+70+85 = 280, \\ \mathrm{ET}_2 &= 55+70+30+40+115.5 = 310.5, \\ \mathrm{ET}_3 &= 55+30+80+55+50 = 270. \end{aligned}$$

We can obtain the updated slack time of activity a_{P_2}, the value of which is 40.5. The slack time of the constituent activities in a_{P_2} is also calculated from the viewpoint of the global process using Eq. 3. $\mathrm{ST}^P_{a_{2-1}}$ is updated to 40.5. In this way, we can provide updated slack time to internal activities of nested processes during the execution of a process.

Finally, let us take a look at the third case. When an activity in a nested process is completed, the slack time

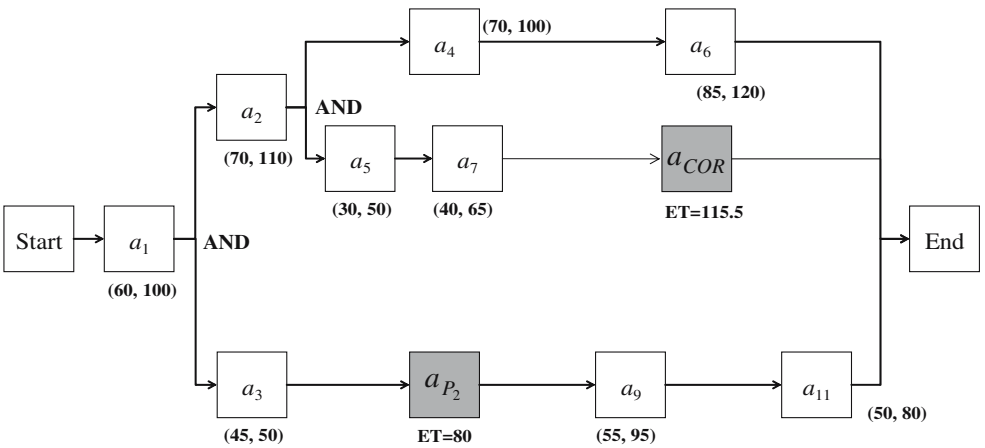


Fig. 4 Folded process for calculation in build-time

needs to be recalculated. That is, when an activity in a nested process is completed, the slack time of the activities in a nested process must also be updated. Suppose that the activities a_{1-1} and a_{1-2} in the nested process a_{P_1} are completed with their actual processing times, 30 and 35, respectively. We also suppose that all of the activities were completed obeying their expected processing times excepting the activities a_6, a_{10}, a_{11}, a_{1-3}, and a_{1-4}. We need to assign a new slack time to the uncompleted activities with which task performers participate, according to a dispatching rule that will be presented at the end of this chapter. Note that the activities in the nested process a_{P_1} do not possess their slack time in build-time because only the representative activity (a_{COR}) of the COR structure is considered. When an actual path is determined in the COR structure and the nested process a_{P_1} is executed, the constituent activities of a_{P_1} can have their own slack time. Now, the slack time of activity a_{1-3} being computed, $\mathrm{ST}_{a_{1-3}}^{a_{P_1}}$ is easily acquired, the value of which is 5, and the actual processing time of activity a_{P_1} is 90 according to Eq. 2. The slack time of the activities on three paths are recalculated, and the slack time of a_{P_1} is updated to 0. Finally, $\mathrm{ST}_{a_{1-3}}^{P}$ is 5 by Eq. 3.

When an activity is assigned to a task performer, that performer also receives the calculated slack time. These values are attached to every activity of every process instance. Hence, a task performer uses the slack time when he determines the order of performing his tasks from the viewpoint of the global process. The slack time, providing insight into which activity is most urgent, allows the user to prioritize. That is, according to the information, the user participating in an SCP deals with the most urgent task first. We call this dispatching rule the 'S-LST Rule.' We expect that the S-LST rule contributes to the increase of SCP efficiency, which is verified by the simulation experiments presented in Chapter 5.

4.3 Web service for SCP integration

Our method is applied to an environment in which heterogeneous processes of partners form a global SCP. For effective execution of such an SCP, important information such as inventory and final demand information as well as process description is required to be shared among participants in the supply chain in real time. Using that inter-communicated information, it is necessary to make decisions in real-time and to operate the supply chain synthetically, that is, by way of high-level process integration (Seshasai et al., 2005). For this purpose, a web service for facilitating interoperability among heterogeneous processes, which is already applied to various business areas such as inter-company negotiation (Kim & Segev, 2005) and more intelligent and freer e-marketplaces, is employed. A web service is used for two different purposes. One is to design and execute process models, and the other is to exchange relevant information used to fulfill the processes.

For designing and executing processes, various kinds of standard languages, such as BPEL4WS, WSCI, BPML (Business Process Modeling Language), and XPDL (XML Process Definition Language), have been developed. In the present study, we employed BPEL to represent processes among heterogeneous systems (Wohed, Aalst, Dumas, & Hofstede, 2002). Originally, BPEL was used to describe business processes based on web services, and it is known that the expressive power of the language is not poorer than that of system-specific PDL (Process Definition Languages). Most patterns of the BPM process model can be transformed into BPEL codes (Wohed et al., 2002). We use BPEL codes as an executable form of an SCP managed by BPM, which form includes the following attributes:

- The SCP includes processes of multiple companies participating in the supply chain whose internal operations are expressed as nested processes in a high-level process.
- The SCP is complex structurally and requires a design tool that provides GUI (Graphic User Interface) and transforms graphical representation of a process into BPEL codes.
- The nested processes of partners are executed in heterogeneous system environments.
- There is a lack of tools to execute and manage the SCP effectively and efficiently.

According to our method, all constituent activities of a global process have their slack time, which should be updated whenever activities are completed. This slack time information needs to be exchanged among the multiple partners that participate in a global SCP. The process itself is described using BPEL, and definition of additional elements for slack time information in the code is also required to be added as shown in Fig. 5. The information is dealt with as a kind of process variable, the variable containing both the slack time of individual processes and that of the global process.

A web service can also be used to exchange time information among partners. For exact calculations of slack time in real time, time information such as expected processing time, actual processing time, and slack time has to be shared continuously among a global process and its nested processes (Seshasai et al., 2005). Figure 5 presents a BPEL code for nested process a_{P_2} in Fig. 3. The two elements in the dotted square are designed to deal with two kinds of slack time. One variable, the name of which is 'pv_slacktime,' refers to the slack time of the constituent activities from the viewpoint of a_{P_2}. The other, 'pv_overall_slacktime,' is related to slack time with regard to the global process P. Once two process variables are declared and initialized, the values are updated repeatedly.

Fig. 5 A process designed using BPEL

```
<process name="Urgent Production Process" targetNamespace="http://abc.com/supplychain"
suppressJoinFailure="yes" xmlns:tns="http://abc.com/supplychain"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/
business-process/" xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:ora="http://schemas.oracle.com/xpath/extension" xmlns:factory="http://abc.com/factory"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <partnerLinks>
    <partnerLinkname="globalSCM" partnerLinkType="tns:supplychain" myRole="supplychainProvider"
      partnerRole="supplychainRequester"/>
  </partnerLinks>
 <variables>
    <variable name="input" messageType="tns:supplychainRequestMessage"/>
    <variable name="output" messageType="tns:supplychainResponseMessage"/>
    <variable name="pv_slacktime" type="xsd:string"/>
    <variable name="pv_overall_slacktime" type="xsd:string"/>          Process Variable Declaration
 </variables>
 <sequence name="main">
    <receive name="Receive Request" partnerLink=" lobalSCM" portType="tns:supplychain" operation="initiate"
      variable="input" createInstance="yes"/>
    <invoke name="Production"/>
    <invoke name="Dispatch Products"/>
    <invoke name="callback SCM" partnerLink=" lobalSCM" portType="tns:supplychainCallback"
operation="onResult"
      inputVariable="output"/>
</sequence>
</process>
```

The BPEL code is required to deliver current values of slack time from a nested process to a global process or from a global process to a nested process.

In order to deliver the relevant time information, SOAP (Simple Object Access Protocol) is employed. The SOAP enables the sharing and delivering of accurate information in heterogeneous environments. A SOAP message is constructed to exchange the information. Figure 6 shows SOAP messages utilized in the SCP. Figure 6a shows a message of the global process *P*, and Fig. 6b illustrates that of the nested process a_{P_2}.

SOAP messages possess an important element, the delivering of slack time information to other processes. The element of messages differs according to the utility of the messages. If a SOAP message is delivered to a nested process, its element (GlobalSlackTime in Fig. 6a) contains the new slack time of the nested process (calculated from the viewpoint of the global process). On the other hand, if a SOAP message is delivered to a global process, its element (NestedSlackTime in Fig. 6b) includes the slack time result of constituent activities within a nested process at a point in time.

5 Simulation experiments

In this chapter, we describe simulation experiments, carried out for the sample process in Section 3.2 using Arena 6.0, to verify the effectiveness of the S-LST rule for the SCP. It was assumed that the processing time of a task and the arrival rate of the process instances follow an exponential distribution, and a simulation model including rules for assigning work and prioritizing tasks (Bae, Jeong, Seo, Kim, & Kang, 1999) was prepared. Each simulation scenario was simulated during replication time 30,000, and this is repeated 30 times. The warm-up period was set to 3,000, which is approximately the time needed for the total number of ongoing process instances to become constant.

5.1 Comparison with traditional methods

In existing BPMs, participants usually carry out their jobs according to rules such as SPT, FIFO, or Random Rule. When a deadline is provided for each activity, they employ EDD (Early Due Date), proposed in Lu & Kumar (1991). In our experiments, the S-LST Rule was compared with the traditional methods, and the result was analyzed for process efficiency. Since process efficiency can be influenced by the arrival rate of process instances, the arrival rate was initially set at 120. Process efficiency is measured in two ways: the average completion time of process instances (PE_1), and the average number of process instances completed during a simulation run (PE_2). The experimental results are summarized in Table 2.

Table 2 shows that the S-LST rule outperforms the others. The improvement rate of the S-LST rule over the second best rule, EDD, is about 30% for both PE_1 and PE_2. The comparison between the S-LST and the EDD is also statistically significant.

Another analysis was performed to measure the average delay time, which is the average of the subtraction of due date from completion time. A portion of the results are presented in Table 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-
instance">
<SOAP-ENV:Header>
<Request encoding="java"></Request>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:eReceiveGlobalSlackTime
xmlns:ns1="urn:com-bpm-adapter-SCMeIntegrator“
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/so
ap/encoding/">
<ProcessIDxsi:type="xsd:string">
497
</ProcessID>
<ProcessNamexsi:type="xsd:string">
SCM_Seoul_R00003
</ProcessName>
 <NestedProcessIDxsi:type="xsd:string">
498
</NestedProcessID>
<GlobalSlackTimexsi:type="xsd:string">
101:2^102:3.3^103:5.6^104:2
</GlobalSlackTime>
 </ns1:eReceiveGlobalSlackTime>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

(a) SOAP message to the nested process

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/“x
mlns:xsi="http://www.w3.org/1999/XMLSchema-
instance">
<SOAP-ENV:Header>
<Request encoding="java"></Request>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:eSendLocalSlackTime xmlns:ns1="urn:com-
bpm-adapter-SCMeIntegrator"
SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soa
p/encoding/">
<ProcessIDxsi:type="xsd:string">
498
</ProcessID>
<ProcessNamexsi:type="xsd:string">
SCM_Seoul_R00003_out
</ProcessName>
<ParentProcessIDxsi:type="xsd:string">
497
</ParentProcessID>
<NestedSlackTimexsi:type="xsd:string">
101:0^102:1.3^103:3.6^104:0
</NestedSlackTime>
 </ns1:eSendLocalSlackTime>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

(b) SOAP message to the global process

Fig. 6 SOAP messages to share slack time information among heterogeneous systems

As shown in Table 3, the difference of the average delay time between the two rules is not statistically significant for some of the constituent activities. This insignificant difference was also applied to the other activities. The EDD is usually considered to be useful for observing the deadlines. However, when the workload is larger than that in this experiment, it is uncertain whether the rule is still as effective as the S-LST rule. The effect of workload variation on process efficiency is detailed in Section 5.2.

5.2 Influence of workload

Workload, determined by the arrival rate of process instances, is an important factor affecting process efficiency. The smaller the arrival rate is, the larger the workload is. Whereas the analysis in the previous section assumed a fixed workload, in this section we study the effect that a varied workload has on the effectiveness of the S-LST rule.

We ran multiple simulation sessions, each of which had a different arrival rate. The simulation results are presented in Fig. 7. The X-Axis is the arrival rate of the SCP. The Y-Axis indicates the process completion time (PE1) and the number of completed instances (PE2). The Figure plots the significant *maximal difference* to compare S-LST with EDD. The maximal difference means the maximal difference that is statistically significant. We used the hypothesis of the *t*-test to acquire the maximal differences for each arrival rate as follows:

H0: (PE_i of the second rule)−(PE_i of the best rule)=c,
H1: (PE_i of the second rule)−(PE_i of the best rule)>c′,
i=1 and 2, c=0.

Table 2 Comparison of process efficiency

Performance	EDD	FIFO	Random	S-LST
PE1	750.90	1,008.25	1,102.77	537.61 (28.4)[a]
PE2	243.56	127.00	130.91	306.78 (26.0)[a]

[a] The improvement rate of the best rule over the second best rule

Table 3 Average delay time for each activity

Task	EDD	S-LST	FIFO	Random
a_1	12.33	14.35	23.67	20.12
a_2	13.68	13.13	21.01	19.97
a_3	9.66	8.16	15.64	13.88
a_6	17.45	16.23	27.42	28.64
a_7	10.62	10.66	15.31	13.75
a_{P_1}	35.57	32.69	45.10	47.33
a_{P_2}	28.83	26.14	39.18	41.12
a_{10}	7.14	6.67	11.89	11.51

The graphs show clearly how the improvements of process efficiency that S-LST produces are affected by the workload. As the arrival rate decreases, the improvement tends to increase. In other words, the larger the workload is, the more improvement is effected. The reason is that a large workload makes taking activity deadlines into consideration difficult for participants using EDD. This leads us to conclude that the LST rule is a more effective strategy in an SCP when the workload is sufficiently large.

5.3 Combinations of rules

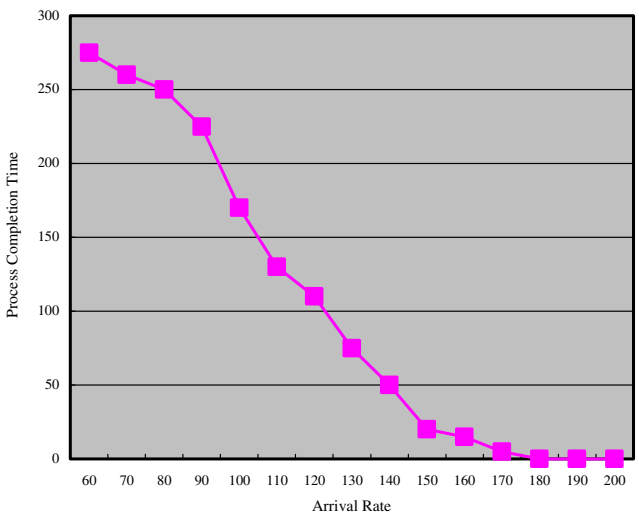
We have shown that the S-LST outperforms the traditional dispatching rules in overall process efficiency. This is because the S-LST takes task urgency into account during the entire process. However, this does not mean that S-LST is always the best rule. Additional information or measurements can improve the efficiency of the S-LST rule. The following analysis illustrates this possibility.

We devised the W-LST (Weighted-LST) rule using the suggestions in (Holthaus & Rajendran, 1997) and (Zhao & Stohr, 1999). Although S-LST focuses on process efficiency, it does not discriminate between important and unimportant process instances. Importance tends to depend on the customers' or the company's point-of-view. To improve process efficiency, then, S-LST must take the importance of each instance into consideration. Let ST_i and w_i denote the slack time and importance of the ith instance, respectively. For convenience in developing the complemented rule, we define importance by the smallness of the value. Participants using the W-LST rule prioritize activity instances with W-ST_i (=ST_i+w_i) and execute the activities with the smallest values first.

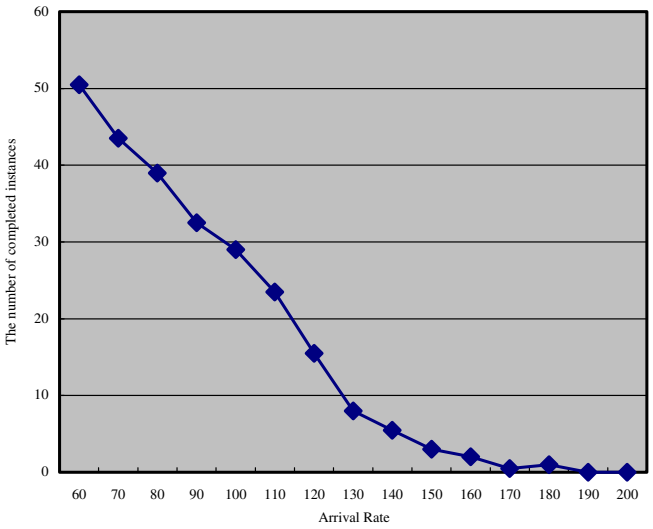
Table 4 presents the completed number of process instances (CN_0) whose importance is 0 during a simulation run. Clearly, W-LST is more effective than S-LST, the difference between S-LST a-nd W-LST being statistically significant at 0.05. This simple analysis leads us to conclude that, from the viewpoint of global SCP efficiency, the S-LST rule is a more effective dispatching rule when used in various combinations with elaborate indicators.

6 System implementation

The essence of our approach is a method of integrated execution and management of the SCM process using a BPM engine and guidance regarding which activity task performers work on first. For the purposes of a prototype with which to implement the proposed methodology to



(a) PE1: S-LST – EDD



(b) PE2: S-LST - EDD

Fig. 7 Influence of workload on process efficiency

Table 4 Comparison with S-LST and Weighted-LST

Performance	S-LST	W-LST
CN_0	23.41	52.76

enhance the SCP's efficiency, we used BizFlow 9.0, a product available worldwide to more than 300 companies.

6.1 Prototype design

Our prototype's design fully utilizes the advantage of BPM, integration on the basis of a process, and employs SOAP to construct interfaces among heterogeneous systems. BizFlow 9.0 provides interfaces designed for integration of legacy applications and facilitates process interoperability using web services or Enterprise Java Beans (EJB). Process interoperability is addressed by BPEL-based process models and SOAP messages, both specified in Section 4.3. The system architecture of the prototype, based on the sample SCP shown in Fig. 1, is presented in Fig. 8.

A global process of an SCP is modeled on a BPM server. The server possesses a process designer tool and an engine. The engine executes the process, delivers work items to participants, and requests the execution of external applications. It also calls on a 'Web Service Integration Module' (WSIM) to perform processes that are designed on other BPM servers or registered in web services. The WSIM delivers necessary information to the external processes by constructing SOAP messages, and then receives results from the execution of those processes. This module interprets the received SOAP message and delivers the actual processing time of activities to the engine when the activities are completed and slack time has to be recalculated. By using the time information, the engine calculates critical path and slack time, displays them on the monitor, and transfers them to the BPM DB using developed API. The BPM DB handles relevant data such as process,

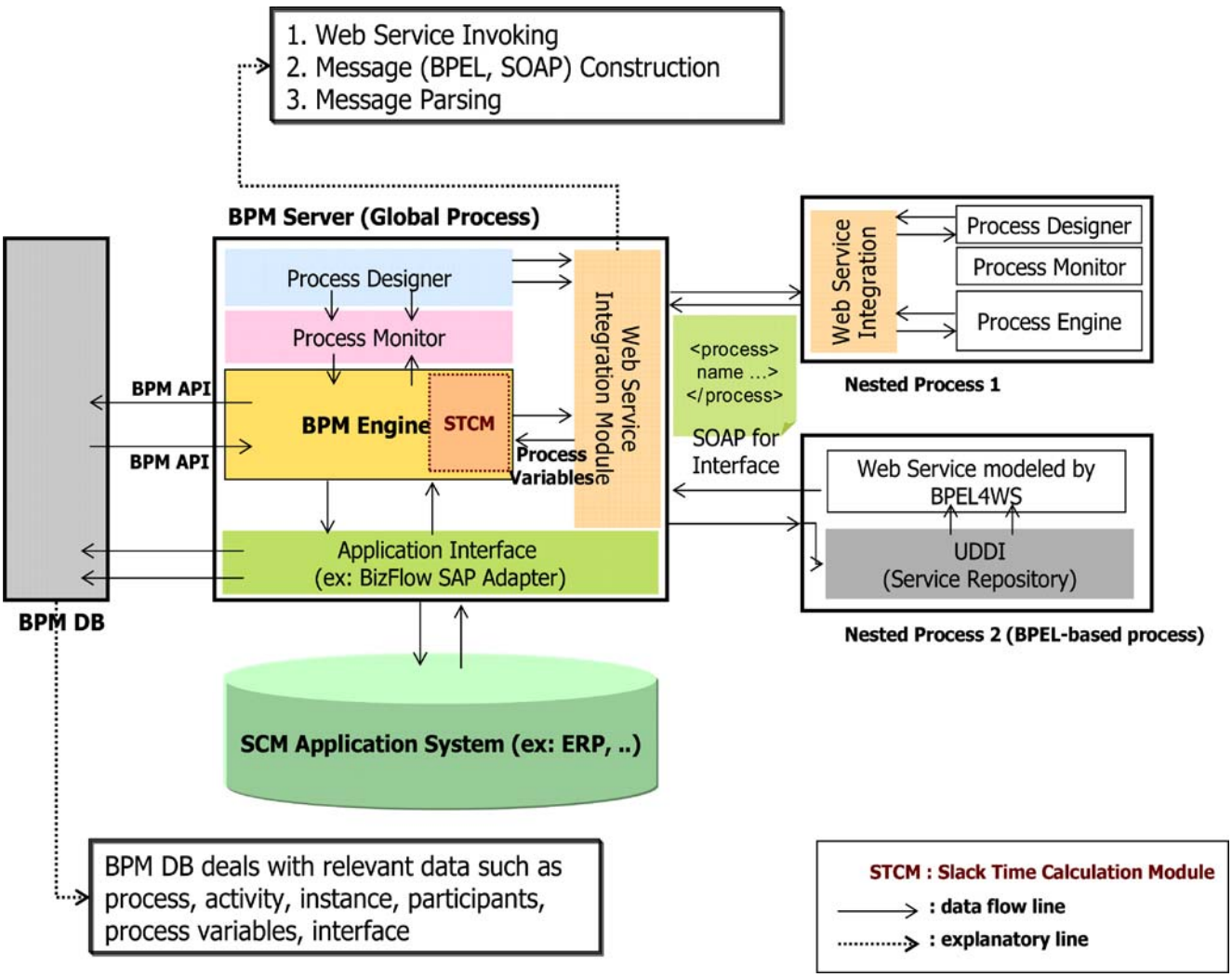
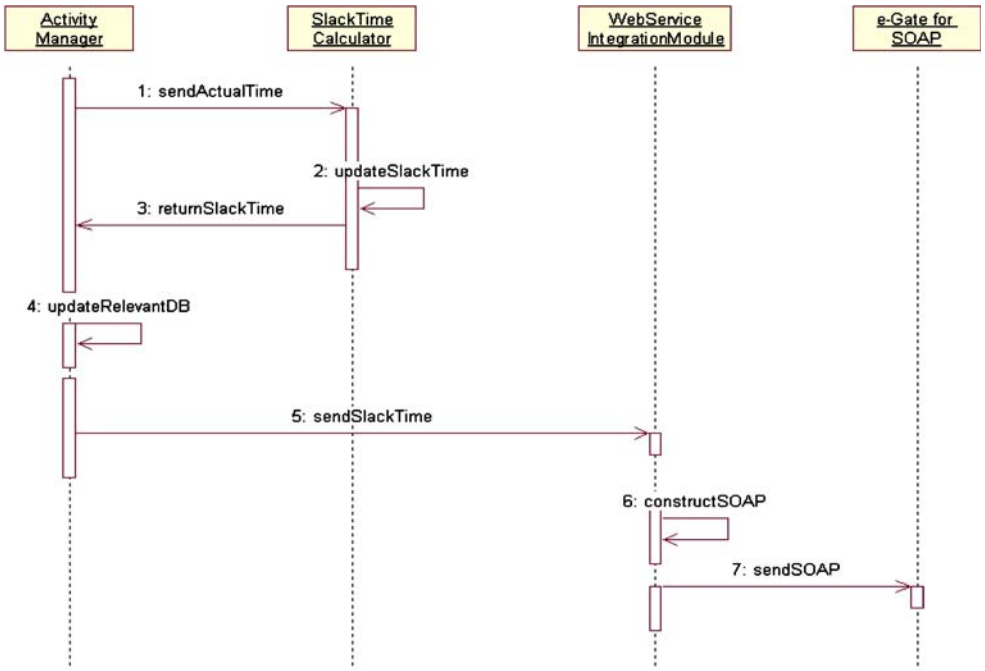
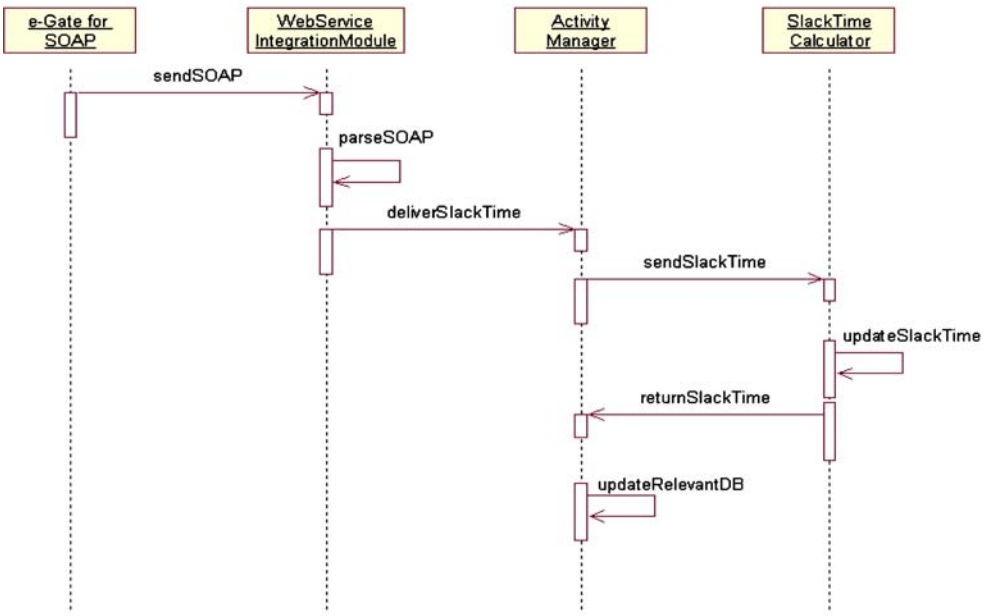


Fig. 8 Overall system architecture

activity, instance, participants, process variables, and interface. The two squares on the right side of Fig. 8 present the common nested processes in an SCP from the viewpoint of the system architecture. We provide more systematic descriptions of this overall procedure using sequence diagrams in UML, which consist of some JAVA classes and their methods in the BPM modules such as the engine and the WSIM. Figure 9a shows the flow that takes place when an activity is completed in a global process and slack time is delivered to nested processes. Figure 9b illustrates that when an activity in a nested process is completed, the slack time of the nested process is returned to the global process and then updated as the slack time of a constituent activity.



(a) To send SOAP messages to a nested process



(b) To receive SOAP from a nested process

Fig. 9 Sequence diagrams describing the logical flows of our prototype

Four classes are considered in the sequence diagram. Figure 9a shows a procedure by which a message is delivered from a nesting process to a nested process. The class Activity Manager (AM) is a class of the BPM engine, and it usually handles time information such as actual processing time and slack time, and delivers the information to the table 'process variables' in the database. The AM sends the actual process time of an activity to the class Slack Time Calculator (STC) whenever any activity is completed. Once the STC receives the time, a new slack time is calculated using this information and is returned to the AM. The AM updates the newly calculated values to the DB table and then sends these values to the class WSIM. The WSIM, including methods to construct SOAP messages, constructs them and then sends the constructed messages to the nested processes. For the nested processes, the class e-Gate for SOAP is located in heterogeneous systems, and receives the SOAP messages delivered from the global process. For the global process, another e-Gate for SOAP is embedded in the BPM. As illustrated in Fig. 9b, a message is delivered in a reversed direction, the procedure of which is similar to that shown in Fig. 9a.

The proposed system architecture becomes a characteristic structure where legacy applications and independently executable services are rearranged on the basis of a global process. To allow for effective implementation of the global process, the structure leads each participant to take care of his/her own process and to execute it by referring to well-defined interfaces. It also enables the recalculation of the slack time information of the global process and its nested processes in real time as well as the sharing of that information among those processes.

6.2 Prototype implementation

The process designer of our system can be used to design SCPs. The sample process shown in Fig. 1 comprises normal activities, in which task performers work on their work items using SCM applications, as well as nested processes, in which some requests are carried out by external partners. The nested processes designed by BPEL are registered to UDDI for execution in the global process. Two process variables are also declared for dealing with slack time calculation. The first is to serve slack time calculated for a nested process. The second is to provide slack time calculated on the basis of a global process.

The prepared process is executed by a BPM engine. The BPM carries out the nested processes to facilitate interoperation with external processes: the engine delivers time information with SOAP messages to the external processes,

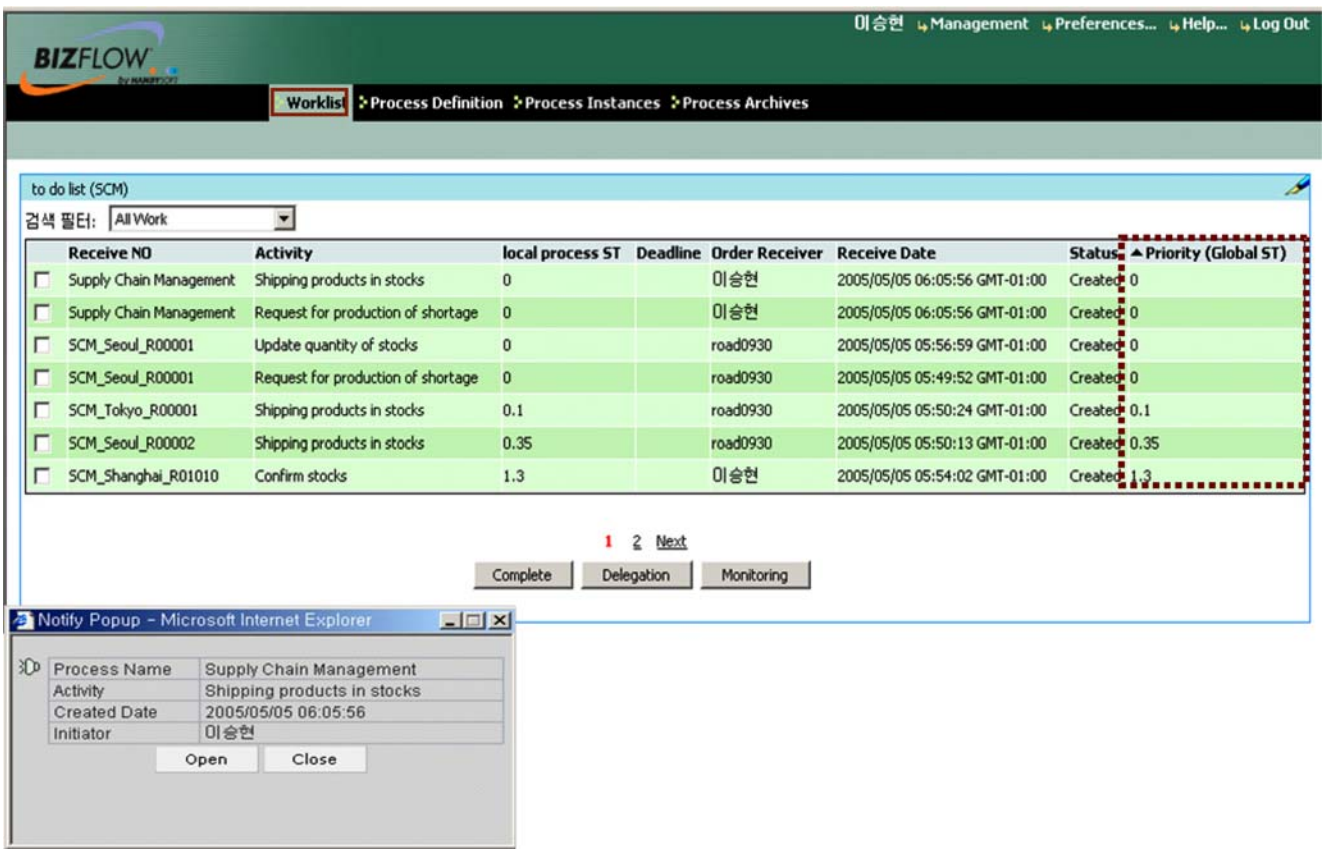


Fig. 10 Client interfaces of our prototype system

and then returns new information from them, as explained in Section 6.1. Once the nested processes are specified with BPEL, their details are registered to UDDI by the WSIM of the product. External partners perform the nested processes in the UDDI following the predefined procedure. The procedure is defined in BPEL, and its details are presented in Fig. 5 in Section 4.3.

Task performers can observe several activities to work on in their worklist. Each of the listed activities possesses its own slack time value calculated for the current time. This information helps task performers determine which activity must be completed first. Each performer can choose a work item to be executed depending on the slack time information displayed in the right-most column of his/her worklist, as shown in Fig. 10. On the bottom-left of Fig. 10, there is a pop-up window that informs the performer of the activities that reach his/her worklist.

7 Conclusions and further research issues

This paper proposes a method of increasing the efficiency of Supply Chain Process (SCP) execution that enables interoperability among heterogeneous information systems. In SCP, multiple partners having their own processes form a global process. Therefore, SCP requires functions for process execution that have particularized features such as monitoring of the global process, interoperability among partners, and information sharing. These requirements can be satisfied by employing a BPM system. However, traditional BPM systems do not provide an efficient method of SCP execution, since they focus mainly on process integration, automation, and correctness. A dispatching rule, as explained in this paper, is provided for users, with which they can prioritize tasks in their worklists according to the extent of urgency. In formulating this rule, we determined the critical path based on expected processing time, and computed the slack time of each task in a process. While executing the SCP, partners participating in the process need to exchange information, which includes urgent information. We use a service-oriented system architecture for effective sharing of information, as well as a BPEL4WS standard to describe processes communicated among heterogeneous parties. Our method not only improves the overall efficiency of the SCP but also enables each partner involved in an SCP to monitor the process either in whole or in part. Henceforth, we expect that our approach will contribute to the competitive power of a supply chain in which multiple nested processes form a global SCP.

Some interesting further research issues remain, of which there are two aspects: one is efficient process execution for more complex processes, and the other is solid process interoperability. From the perspective of the first aspect, we need to consider for our method more complex SCPs than the basic-structured SCP presented in this paper. We expect that workflow patterns developed in Wohed et al. (2002) can be good comprehensive structures for our method. In addition, whereas our method is originated from the traditional project management, more effective theories can be introduced to improve process efficiency. Techniques developed in such areas as scheduling and TOC (Theory of Constraints), for example, might be employed. Finally, it is very important to take into account various factors that affect process efficiency. In this paper, arrival rate and combinations of other rules are considered. However, there are more influential factors in process efficiency, such as participants' skill level, the number of participants, involvement of automated tasks, dynamic variation of arrival rates, and others.

Our work is also associated with process interoperability or integration. We first need to escape a platform-dependent environment and apply our method to diverse supply chain environments that use standard e-commerce frameworks such as ebXML and RosettaNet. Then, new interface methods and analysis tools to compare the environments will be required. Finally, it is necessary to pay attention to the four dimensions of the process interoperability description framework proposed in Zhao (1999): connectivity, expressivity, visibility, and flexibility. Especially, our method must enable easier alteration of global or nested process specifications. Consequently, we will have to consider an expanded framework that enables calculation and modification of slack time independently regardless of the alteration of the processes.

Acknowledgements This work was supported by the Regional Research Centers Program (Research Center for Logistics Information Technology), granted by the Korean Ministry of Education & Human Resources Development.

References

Bae, J., Bae, H., Kang, S., & Kim, Y. (2004). Automatic control of workflow processes using ECA rules. *IEEE Transactions on Knowledge and Data Engineering, 16*(8), 1010–1023.

Bae, J., Jeong, S.-C., Seo, Y., Kim, Y., & Kang, S. (1999). Integration of workflow management and simulation. *Computers & Industrial Engineering, 37*(1–2), 203–206.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Chang, D.-H., Son, J. H., & Kim, M. H. (2002). Critical path identification in the context of a workflow. *Information and Software Engineering, 44*(7), 405–417.

Hammer, M. (2001). *Agenda: What every business must do to dominate the decade*. New York: Random House.

Holthaus, O., & Rajendran, C. (1997). Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics, 48*(1), 87–105.

Hu, Y., Minciardi, R., Paolucci, M., & Pesenti, R. (1992). Techniques for dynamic scheduling in a manufacturing environment. In *Proceedings of the 31st IEEE Conference on Decision and Control* (pp. 404–408).

Kim, Y., Kang, S., Kim, D., Bae, J., & Ju, K. (2000). WW-flow: Web-based workflow management with runtime encapsulation. *IEEE Internet Computing, 4*(3), 55–64.

Kim, Y., & Rhee, S.-H. (2002). Algorithm of real time computation of slack time for workflow processes. In *SNU Information Systems Lab. Technical Report, SNUIS-TC-1017*.

Kim, J., & Segev, A. (2005). A web services-enabled marketplace architecture for negotiation process management. *Decision Support Systems, 40*(1), 71–87.

Klein, M., & Dellarocas, C. (2000). A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work, 9*, 399–412.

Kobayashia, T., Tamakia, M., & Komodab, N. (2003). Business process integration as a solution to the implementation of supply chain management systems. *Information and Management, 40* (8), 769–780.

Kumar, A., & Zhao, J. L. (2002). EROICA: A rule-based approach to organizational policy management in workflow systems. In *Proceedings of the 3rd conference on web-age information management* (Beijing, China, August 11–13).

Latimer, N. (2004). Business process management preliminary market size and forecast. SWSI-WW-MT-0123, Gartner.

Lu, S. H., & Kumar, P. R. (1991). Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control, 36*(12), 1406–1416.

McCoy, D. (2004). The convergence of BPM and BAM. Gartner Research Note, SPA-20-6074.

Park, S. C., Raman, N., & Shaw, M. J. (1997). Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach. *IEEE Transactions on Robotics and Automation, 13*(4), 486–502.

Pozewaunig, H., Eder, J., & Liebhart, W. (1997). ePERT: Extending PERT for workflow management systems. In *The First European Symposium in ADBIS* (pp. 217–224).

Rajendran, C., & Ziegler, H. (2001). A performance analysis of dispatching rules and a heuristic in static flowshops with missing operations of jobs. *European Journal of Operational Research, 131*(3), 622–634.

Rhee, S.-H., Bae, H., Ahn, D., & Seo, Y. (2003). Efficient workflow management through the introduction of TOC concepts. In *Proceedings of the 8th annual international conference on industrial engineering theory, applications and practice (IJIE2003)*, Las Vegas, America.

Rhee, S.-H., Bae, H., & Kim, Y. (2004). A dispatching rule for efficient workflow. *Concurrent Engineering—Research and Applications, 12*(4), 305–318.

Rhee, S.-H. et al. (2005). Process-oriented development of job manual system. In *Lecture notes in computer science*, vol. 3482 (pp. 1259–1268). Berlin Heidelberg New York: Springer.

Seshasai, S., Gupta, A., & Kumar, A. (2005). An integrated and collaborative framework for business design: A knowledge engineering approach. *Data & Knowledge Engineering, 52*(1), 157–179.

Simchi-Levi, D., Kaminsky, P., & Simchi-Levi, E. (2000). *Designing and managing the supply chain*. Singapore: McGraw-Hill.

Smith, H., & Fingar, P. (2003). *Business process management—The third wave*. Tampa, FL: Meghan-Kiffer.

Son, J. H., & Kim, M.-H. (2001). Improving the performance of time-constrained workflow processing. *Journal of Systems and Software, 58*(3), 211–219.

Stohr, E. A., & Zhao, J. L. (2001). Workflow automation: Overview and research issues. *Information Systems Frontiers, 3*(3), 281–296 (September, special issue on workflow automation and business process integration).

van der Aalst, W. M. P., Weske, M., & Grunbauer, D. (2005). Case handling: A new paradigm for business process support. *Data & Knowledge Engineering, 53*(2), 129–162.

Wohed, P., Aalst, W. M. P., Dumas, M., & Hofstede, A. H. M. (2002). Pattern Based Analysis of BPEL4WS. Technical Report FIT-TR-2002-4, QUT.

Workflow Management Coalition: Process Definition Interchange, Document no. WfMC-TC-1016-P (1999a). http://www.wfmc.org.

Workflow Management Coalition: Terminology & Glossary, Document no. WfMC-TC-1011 (1999b), http://www.wfmc.org.

Zhao, J. L. (1999). Interoperability requirements for cooperative workflows in electronic commerce. In *Proceedings of the 2nd international conference on telecom and electronic commerce (ICTEC99)*, Nashville, TN.

Zhao, J. L., & Stohr, E. A. (1999). Temporal workflow management in a claim handling system. *SIGSOFT Software Engineering Notes, 24*(2), 187–195, March.

Seung-Hyun Rhe is a researcher in the Department of Industrial Engineering at Seoul National University. Before becoming a researcher, he had been with Handysoft Corporation, an international business process management (BPM) vendor located in Korea, for two and a half years. In Handysoft, he was responsible for designing its client companies' processes, integrating their legacy systems and developing a software product. Now, he is interested in business process design, e-marketplace design using web service technology and industry redesign by IT. His current research activities include the improvement of BPM using theory of constraints and the development of personalized content service on the internet.

Hyerim Bae is an assistant professor in the Industrial Engineering Department at Pusan National University (PNU), Korea. He received Ph.D., M.S., and B.S. degrees from the Industrial Engineering Department at Seoul National University, Korea. He had been a manager for information strategic planning at Samsung Card Corporation before he joined PNU. He is interested in the areas of business process management (BPM), process-based B2B integration and ubiquitous business computing. His current research activities include analysis of business process efficiency, controlling of logistics processes with context awareness and convenient modeling of business processes.

Yongsun Choi is an associate professor in the Department of Systems Management and Engineering, Inje University, Korea. He received his B.S. degree in Industrial Engineering from Seoul National University and his M.S. and Ph.D. degrees in Industrial Engineering from Korea Advanced Institute of Science and Technology. He has been to Stanford University, Tokyo University and University of Arizona as a visiting scholar. His research interests include workflow and business process management, Web services, Semantic Web, etc.