Available online at www.sciencedirect.com
SCIENCE DIRECT®
Decision Support Systems 38 (2004) 399–419
Decision Support Systems
www.elsevier.com/locate/dsw

Business-to-business workflow interoperation based on process-views

Duen-Ren Liu*, Minxin Shen

Institute of Information Management, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, ROC

Received 1 February 2002; accepted 23 July 2003
Available online 16 September 2003

Abstract

When cooperating with each other, enterprises must closely monitor internal processes and those of partners to streamline business-to-business (B2B) workflows. This work applies the process-view model, which extends beyond conventional activity-based process models, to design workflows across multiple enterprises. A process-view is an abstraction of an implemented process. An enterprise can design various process-views for different partners based on diverse commercial relationships and, in doing so, establish an integrated process that consists of internal processes and process-views that each partner provides. Participatory enterprises can obtain appropriate progress information from their own integrated processes, allowing them to collaborate effectively. Furthermore, B2B workflows are coordinated through virtual states of process-views. This work develops a uniform approach to manage state mappings between internal processes and process-views. The proposed approach enhances prevalent activity-based process models adaptable to collaborative environments.

Keywords: Process-view; Virtual workflow; Interorganizational workflow; Interoperation

1. Introduction

Enterprises cooperate strategically to develop a competitive business alliance or virtual enterprise. Cooperative relationships are formed by merging the operational processes of participants. Workflow technology enables an enterprise to construct a process-oriented organization efficiently. Business-to-business (B2B) workflows are streamlined if value-added activities are arranged from a process-oriented aspect as well as controlled automatically by integrating workflow management systems (WfMS) of participatory enterprises. However, managing workflows among multiple enterprises is more complex than doing so for an individual enterprise.

To remain competitive, a cooperating enterprise must conceal its internal process structures. However, collaborating enterprises must exchange business information. For example, a notebook computer manufacturer may submit a purchase order to a hard disk manufacturer. In addition to the data required for interaction, if an enterprise provides adequate progress status, its partners may respond as anticipated. For example, if the notebook company exposes appropriate progress data of its assembly workflow, then

* Corresponding author. Tel.: +886-3-5712121-57405; fax: +886-3-5723792.
E-mail address: dliu@iim.nctu.edu.tw (D.-R. Liu).

0167-9236/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-9236(03)00116-7

the hard disk manufacturer may deliver products just in time. Thus, the notebook company can reduce the stock cost required for storing materials such as hard disks. Moreover, distributed and heterogeneous WfMSs of cooperating enterprises must be integrated to automate B2B workflows.

From the perspective of decision support, a WfMS offers an integrated environment for decision makers to analyze, simulate, design, enact, control, and monitor the overall business processes of an enterprise. As the outsourced tasks grow, workflows extend across several enterprises. Workflow technology is expected to support decision-making within an interorganizational environment, just as it does within a single enterprise.

Our previous study [17] proposed a process-view model that enhances the capability of process abstraction in conventional activity-based process models [7]. A process-view, i.e., a virtual process, is abstracted from an actual process. According to distinct organizational roles' requirements, a process modeler can design various process-views, hence providing the appropriate process information to each participant. However, the preliminary process-view model does not consider managing workflows within interorganizational collaboration.

This work proposes a process-view-based coordination model that extends the preliminary process-view model [17] to effectively address the issues of managing B2B workflows. A process-view abstracts critical commercial secrets and is an external interface of an internal process. An enterprise can design various process-views, which are unique to each partner. Process-views of participatory enterprises comprise a collaboration workflow. Moreover, this work employs *virtual states* to coordinate B2B workflows. The virtual states (execution states) of a process-view represent progress status of an internal process. A uniform means is developed to manage state mappings between internal processes and process-views. An enterprise can monitor and control the progress of partners' processes through the virtual states of their process-views. Furthermore, data abstraction is proposed to derive meaningful *process-view relevant data* since our previous work focus only on the control flow of process-views. With these extensions, the enhanced process-view model contains a modeling tool that can accurately describe interorganizational workflows as well as an interoperation mechanism to coordinate autonomous, heterogeneous and distributed WfMSs.

The rest of this paper is organized as follows. Section 2 presents the process-view model and its applications within inter-enterprise cooperation. Section 3 then summarizes how to define an order-preserving process-view as presented in Ref. [17]. In addition, data abstraction is proposed for deriving process-view relevant data. Next, Section 4 presents the coordination of B2B workflows through the virtual states of process-views. Section 5 presents a prototype to demonstrate the effectiveness of the proposed approach. Section 6 then discusses some properties of the process-view-based approach and reviews related work in collaborative workflow management. Conclusions are finally made in Section 7.

2. Process-view-based coordination model

This section first introduces process-view model and then presents the process-view-based B2B coordination.

2.1. Basic definitions: base process and process-view

A process that may have multiple process-views is referred to herein as a *base process*. A process-view, i.e., an abstracted process derived from a base process, provides abstracted process information. From the users' perspective, a process-view resembles a typical process that consists of activities and dependencies although it is an abstracted form of an implemented process. Based on the process-view definition tool, a modeler can define various process-views to achieve different levels of information concealment. The following summarizes basic definitions of base process and process-view. Please refer to Ref. [17] for detailed definitions, semantics and examples.

Fig. 1 shows an example of base process, where the split and join structures are defined by the Workflow Management Coalition (WfMC) [29]. *AND-SPLIT*: An activity splits into multiple parallel activities that are all executed. *XOR-SPLIT*: An activity splits into multiple mutually exclusive alternative activities, only one of which is followed. *AND-JOIN*: Multiple parallel executing activities join into a single

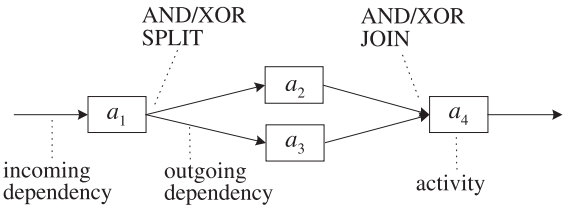


Fig. 1. Sample process.

activity. *XOR-JOIN*: Multiple mutually exclusive alternative activities join into a single activity. A base process is defined as follows.

Definition 1 (Base process). A base process *BP* is a 2-tuple $\langle BA, BD\rangle$, where

1. *BD* is a set of dependencies. A dependency is denoted by $dep(x, y, C)$. Condition *C* represents the constraints, such as time or events, that determine whether routing can proceed from activity *x* to activity *y*.
2. *BA* is a set of activities. An activity is a 3-tuple $\langle SPLIT\_flag, JOIN\_flag, SC\rangle$, where (a) *SPLIT_flag*/*JOIN_flag* may be "NULL", "AND", or "XOR". NULL indicates that this activity has a single outgoing/incoming dependency (Sequence). Given multiple outgoing dependencies, AND indicates all succeeding branches are followed (AND-SPLIT), while XOR indicates only one succeeding branch is followed (XOR-SPLIT). Given multiple incoming dependencies, AND indicates that this activity can be started if all incoming dependencies have satisfied condition (AND-JOIN), while XOR indicates that this activity can be started if one of the incoming dependencies has satisfied condition (XOR-JOIN). (b) *SC* is the *starting condition* of this activity. A workflow engine evaluates *SC* to determine whether this activity can be started. If *JOIN_flag* is NULL, *SC* equals the condition associated with its incoming dependency. If *JOIN_flag* is AND/XOR, *SC* equals Boolean AND/XOR combination of the conditions of all incoming dependencies.
3. For $x, y \in BA$: (a) If there is a path from *x* to *y* in *BP*, then the ordering of *x* is higher than *y*, i.e., *x* precedes *y*. Their ordering relation in *BP* is denoted by $x > y$ or $y < x$. (b) If no path exists from *x* to *y* or from *y* to *x* in *BP*, then *x* and *y* are ordering independent, i.e., *x* and *y* proceed independently. Their ordering relation in *BP* is denoted by $x \infty y$.

The semantics of above definition is as follows. During run-time, an execution of a process is called a *process instance*. For $dep(x, y, C)$, *C* is not evaluated until *x* is completed. The value of *C* is either true of false. The fact that *x* is completed and *C* is true is one precondition of whether *y* can be started. For convenience, "a dependency is evaluated as false/true" represents that the dependency's condition field is evaluated as false/true.

For an activity *ba*, its *SC* field is evaluated when all incoming dependencies of *ba* have been evaluated. *ba* can be started when *SC* is evaluated as true. If *ba* is started, then its outgoing dependencies are evaluated after the completion of *ba*. If *SC* is evaluated as false, then *ba* is not executed in a process instance, and the outgoing dependencies of *ba* are evaluated as false.

An activity is called a *fired* activity in a process instance if its *SC* is evaluated as true and it is executed in the process instance. Contrarily, an activity is called a *non-fired* activity in a process instance if its *SC* is evaluated as false and it is not executed in the process instance. Notably, an activity that is non-fired in a process instance may be fired in other process instances. For convenience, "an activity is evaluated as fired/non-fired" represents that the activity's starting condition is evaluated as true/false.

A process-view is generated from either physical processes (base processes) or other process-views and is considered a *virtual process*. To differentiate the terminology used in base process and process-view, this work uses the terms *virtual activity*/*dependency*/*state* for the process-view while the terms *base activity*/*dependency*/*state* are used for the base process. A process-view is defined as follows.

Definition 2 (Process-view). A process-view is a 2-tuple $\langle VA, VD\rangle$, where (1) *VA* is a set of virtual activities. (2) *VD* is a set of virtual dependencies. (3) Analogous to base process, $\forall va_i, va_j \in VA$, the *ordering relation* between va_i and va_j may be ">", "<", or "∞".

A virtual activity is an abstraction of a set of base activities and corresponding base dependencies. A

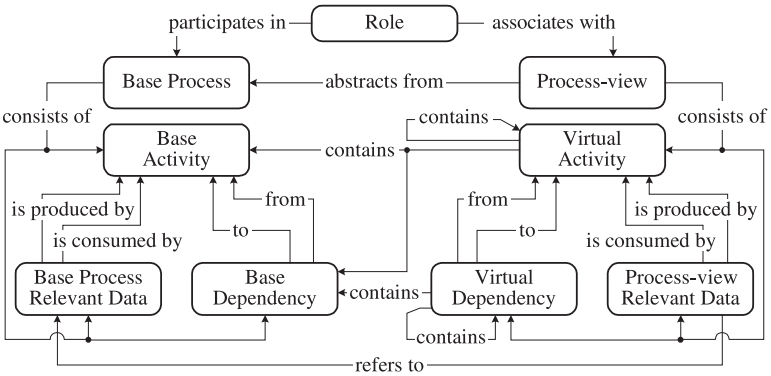


Fig. 2. Process-view model.

virtual dependency connects two virtual activities in a process-view. Fig. 2 illustrates how the components of our model are related. Section 3 demonstrates how to abstract virtual activities and dependencies from a base process. Notably, within an interorganizational environment, a participant's role represents an external partner.

2.2. Process-view-based coordination

Fig. 3 illustrates the interaction scenario, in which three systems cooperate through process-views. To enhance the interoperability through open techniques, process-views' interactions (solid bi-arrow lines) are implemented based on industrial standards, such as CORBA/IIOP and XML. However, each enterprise determines its proprietary implementation of the communication autonomously (blank bi-arrow lines) among base processes, process-views and integrated processes.

A process-view is an external view (or interface) of an internal base process and is derived through the procedure described in Section 3. An *integrated process* is a specific view of the interorganizational workflow that is based on a participatory enterprise's perspective, which consolidates internal processes and partners' process-views. Notably, an integrated process is also a virtual process. Each of its virtual activity/dependency is either a base one of an internal base process or a virtual one of partners' process-views. An integrated process is defined as follows.

Definition 3 (Integrated process). For a base process $BP = \langle BA, BD \rangle$ and a set of process-views $PV_i = \langle VA_i, VD_i \rangle$, an integrated process is a 2-tuple $\langle VA, VD \rangle$, where (1) members of VA are members of BA or

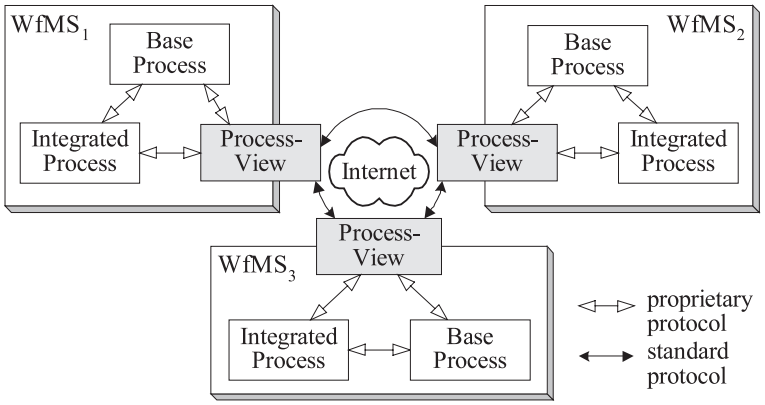


Fig. 3. Process-view-based coordination scenario.

members of VA_i. (2). Members of *VD* are members of *BD* or members of VD_i.

As a base activity/process, a virtual activity/process is associated with a set of (virtual) states to represent its run-time status. A *virtual state* is employed to abstract the execution states of base activities/processes contained by a virtual activity/process. To monitor and control the progress of an internal process through the virtual states of its public process-view, two rules are proposed in Section 4 to map the states between a base and a virtual process. Therefore, an enterprise can coordinate with its partners through virtual states of process-views.

2.3. Three-phase modeling

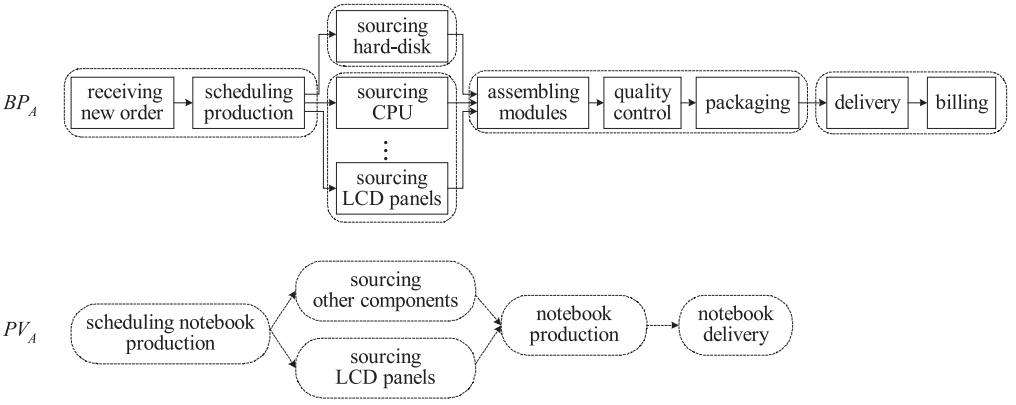
Collaboration modeling is a complex negotiation procedure. Process design is divided into three phases: *base process phase*, *process-view phase*, and *integration phase*.

- Base process phase is the traditional workflow build phase. A process modeler specifies the activities and their orderings in a business process, which is based on a top-down decomposition procedure that many activity-based process models support.
- Next, to facilitate the establishment of an alliance, an enterprise may provide adequate detail regarding its internal processes through process-views. Process-view design is a bottom-up aggregation procedure. A process modeler can define various process-views for the partners according to diverse cooperation relationships.
- Finally, a process modeler forms an integrated process that is a personalized view of an interorganizational workflow through consolidating internal base process and partners' process-views.

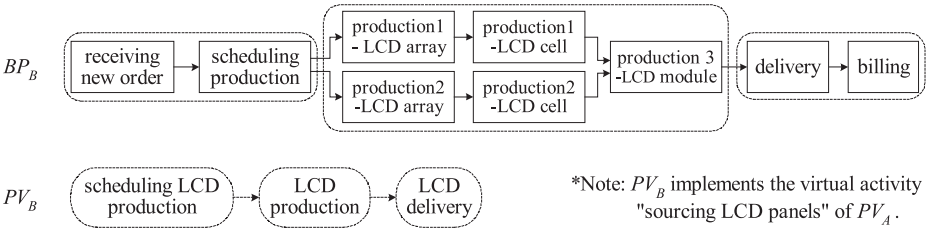
The integrated process not only provides a global view of workflows across cooperating enterprises, i.e., different enterprises may have distinct views regarding the interorganizational workflow, but also allows a process modeler to employ the integrated process definition in order to design action triggers and event notifications in a base process definition. The modeler incorporates appropriate information that is provided in partners' process-views into internal base processes. Then, a system engineer can implement the interaction points that connect B2B workflows based on the integrated process definition and the chosen interoperation standards.

Fig. 4 illustrates the three phases of cooperation between enterprises *A* (notebook computer producer) and *B* (LCD panel manufacturer). After designing a base process, enterprise *A* delegates activity "sourcing LCD panels" to other companies. Assume that enterprise *B* is chosen to complete this activity. Enterprises *A* and *B* then discuss how to merge their respective processes for workflow automation. Both parties must adequately detail their internal processes. They can employ process-view definition tools to determine appropriate abstractions of internal processes for negotiation. Assume that the process-views PV_A and PV_B are defined as depicted in Fig. 4a and b. Enterprise *A* wants *B* to undertake "sourcing LCD panels" in PV_A after "scheduling notebook production" has been completed. *B* must deliver products (i.e., complete "sourcing LCD panels") simultaneously with "sourcing other components" to avoid the stock cost of *A*. Enterprise *B* is also notified that a delay in "sourcing LCD panels" will block the virtual activity "notebook production", hence becoming a bottleneck in PV_A. Enterprise *A* can monitor and control the enactment of the base process of *B* via *B*'s process-view, and vice versa. When both parties agree on the information to be exchanged, the process-views become the interoperation interfaces between *A* and *B*. In the third phase, *A* and *B* define integrated processes, as shown in Fig. 4c and d, respectively.

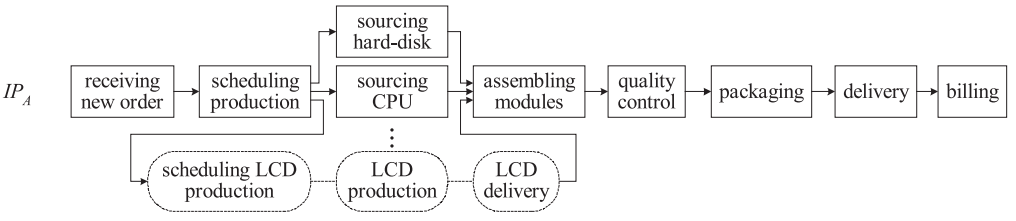
In the above example, enterprise *A* dominates the collaboration because *A* is *B*'s customer. Accordingly, after the base process of *A* is initialized, the virtual activity "scheduling notebook production" in Fig. 4a is notified and *B*'s base process is thus initialized. Fig. 5 depicts the run-time interactions between process definitions (the object name with a prefix "PM_") and process instances. The *event* is used to exchange information (e.g., state changes) between workflow objects. The upper gray area in Fig. 5 illustrates the fact that enterprise *A*'s process-view and integrated process are initialized (see messages 1.2, 1.3, 1.4, and 1.5) after the base process of *A* is initialized (see message 1.1). Consequently, the integrated process of enterprise *B* is notified (1.6). The



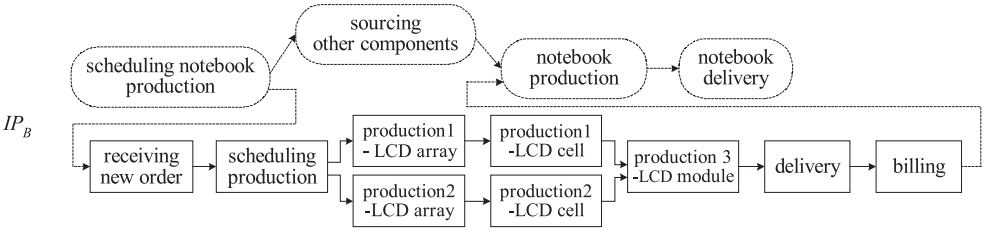
(a) a base process of enterprises A (BP_A) and a process-view provided by enterprises A (PV_A)



(b) a base process of enterprises B (BP_B) and a process-view provided by enterprises B (PV_B)



(c) the integrated process of enterprises A (IP_A)



(d) the integrated process of enterprises B (IP_B)

Fig. 4. Three phases of designing interorganizational workflows.

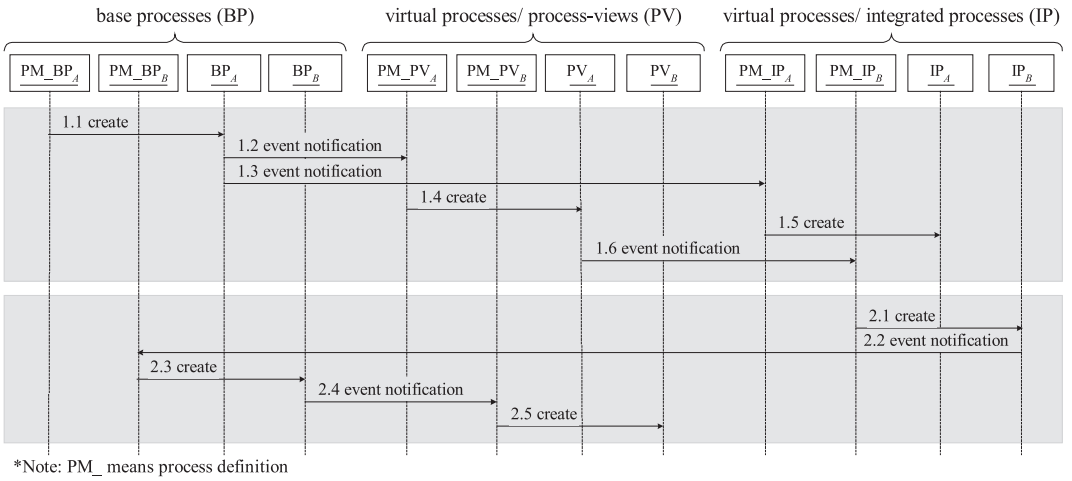


Fig. 5. Interactions between base processes and virtual processes.

lower gray area shows how enterprise *B* reacts to the event notification (1.6). *B*'s integrated process is initialized (2.1) after the event (1.6) is received. Next, the base process of *B* is notified (2.2) and initialized (2.3) to prepare service for *A*. Finally, the process-view of *B* is initialized (2.4 and 2.5) to publicize the progress of service enactment for *A*.

A modeler can define various process abstractions without being restricted by original process definitions since process-views are derived from a base process by bottom-up aggregation. Although the above scenario demonstrates cooperation between two enterprises, the proposed approach can be applied to multi-enterprise cooperation. Under such circumstances, the integrated process of an enterprise may contain internal processes as well as several process-views of partners.

3. Deriving interoperation interfaces: process-views

Enterprises cooperate through process-views. According to the different properties of a base process, various approaches can be developed to derive a process-view. A novel order-preserving approach to derive a process-view from a base process has been presented in [17] and is summarized in Section 3.1. The order-preserving approach ensures that the original execution order in a base process is preserved. Our previous work [17] focuses only on the control flow of process-views, thus data abstraction is further proposed herein. Section 3.2 introduces the operations that derive process-view relevant data from base processes according to access privilege, content, and presentation format.

3.1. Order-preserving process-view

The following summarizes the order-preserving approach for deriving a process-view from a base process. Notably, this summary only presents the case that a base process does not contain loops. Please refer to Ref. [17] for the case that a base process contains loops. A *legal* virtual activity in an order-preserving process-view must follow three rules:

Rule 1 (Membership). The member of a virtual activity may be a base activity or a previously defined virtual activity. The membership among base activities and virtual activities is defined transitively; that is, if *x* is a member of *y* and *y* is a member of *z*, then *x* is also a member of *z*.

Rule 2 (Atomicity). A virtual activity is an atomic unit of processing.

1. A virtual activity is started if one member activity is started, and is completed if all member activities have been evaluated and each fired member activity is completed.

2. In a process-view instance, a virtual activity is evaluated as fired if one member activity is evaluated as fired, and is evaluated as non-fired if all member activities are evaluated as non-fired.
3. The starting of a virtual activity implies that all its preceding virtual activities have been evaluated and each preceding fired virtual activity is completed in the process-view instance.

Rule 3 (Ordering preservation). Given a process-view *VP* as derived from a base process *BP*, for any two different virtual activities va_i and va_j, a_x is a member of va_i and a_y is a member of va_j: (1) "$va_i > va_j$ holds in *VP*" implies that "$a_x > a_y$" for all a_x and all a_y. "$a_x > a_y$" must hold in *BP* for all a_x and all a_y. (2) Statement 1 also applies to the ordering relations "$<$" and "∞".

Based on the rules that a process-view should comply with, virtual activities and virtual dependencies in an order-preserving process-view are formally defined as follows:

Definition 4 (Virtual activity). For a base process $BP = \langle BA, BD\rangle$, a virtual activity *va* is a 5-tuple $\langle A, D, SPLIT\_flag, JOIN\_flag, SC\rangle$, where

1. A is a nonempty set, and its members follow three rules:
   (a) Members of A are base activities that are also members of BA or other previously defined virtual activities that are derived from B*P*.
   (b) *va* is started if one member activity is started, and is completed if all member activities have been evaluated and each fired member activity is completed.
   (c) For any $x \in BA$, $x \notin A$, $\Re \in \{>, <, \infty\}$: if existing a $y \in A$ such that $x \Re y$ holds in BP, then $x \Re z$ holds in BP for all $z \in A$. That means the ordering relations between x and all members (base activities) of A are identical in *BP*.
2. $D = \{\text{dep}(x, y, C_{xy}) | dep(x, y, C_{xy}) \in BD$ and $x, y \in A\}$.
3. *SPLIT_flag* may be "NULL" or "MIX". NULL suggests that *va* has a single outgoing virtual dependency while MIX indicates that *va* has multiple outgoing virtual dependencies.
4. *JOIN_flag* may be "NULL" or "MIX". NULL suggests that *va* has a single incoming virtual dependency while MIX indicates that *va* has multiple incoming virtual dependencies.
5. *SC* is the *starting condition* of *va*.

The *SPLIT_flag* and *JOIN_flag* cannot simply be described as AND or XOR since *va* is an abstraction of a set of base activities that may be associated with different ordering structures. Therefore, MIX is used to abstract the complicated ordering structures. A WfMS evaluates *SC* to determine whether *va* can be started. Members of A are called *va*'s *member activities*, and members of D are called *va*'s *member dependencies*. To save space, the abbreviated notation $va = \langle A, D\rangle$ is employed below to represent a virtual activity.

Definition 5 (Virtual dependency). For two virtual activities $va_i = \langle A_i, D_i\rangle$ and $va_j = \langle A_j, D_j\rangle$ that are derived from a base process $BP = \langle BA, BD\rangle$, a virtual dependency from va_i to va_j is $vdep(va_i, va_j, VC_{ij}) = \{dep(a_x, a_y, C_{xy}) | dep(a_x, a_y, C_{xy}) \in BD, a_x \in A_i, a_y \in A_j\}$, where the virtual condition VC_{ij} is a Boolean combination of C_{xy}.

Furthermore, *essential activity* is proposed to simplify process-view design. Prior to defining a virtual activity, a modeler must select the activities that are essential to this virtual activity. These chosen activities are called essential activities, and form an essential activity set *EAS*. Given an *EAS*, a modeler must identify a minimum virtual activity *min_va*(*EAS*) which contains only the essential and adequate activities that are required to preserve the original ordering relations in the base process. In Fig. 6, for example, a modeler only wants to discover an abstraction that contains a_3 and a_5, i.e., $EAS = \{a_3, a_5\}$. The three definitions, va_1, va_2, and va_3, are legal and va_1 is

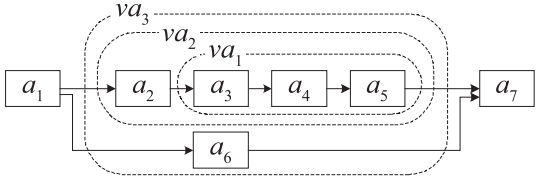


Fig. 6. An example of a minimum virtual activity.

the minimum virtual activity of *EAS*. In va_1, a_3 and a_5 are the activities the modeler wishes to abstract, and a_4 is required for ordering preservation.

Fig. 7 shows the procedure of defining an order-preserving process-view. A process modeler must initially select essential activities. The process-view definition tool then automatically generates a legal minimal virtual activity that encapsulates these essential activities. The above two steps are repeated until the modeler determines all required virtual activities. The definition tool then generates all virtual dependencies between these virtual activities as well as the ordering fields (*JOIN/SPLIT_flag*) and starting condition (*SC*) of each virtual activity. [Ref. 17] presents the algorithm that implements the process-view definition tool (the three gray blocks in Fig. 7).

3.2. Data abstraction

Base process relevant data defines the data objects created and used within a base process [30]. Similarly, the produced and consumed data of a process-view is called *process-view relevant data*. The produced/consumed data of a virtual activity may not be exactly the same as the data that is produced/consumed by the base activities that belong to the virtual activity, since a process-view provides abstracted and aggregated information of a base process. A process modeler must define process-view relevant data according to the requirements of abstraction. The process-view relevant data may be newly defined data that aggregates base process relevant data, or a selected portion of base process relevant data. Besides data contents, access privileges and presentation formats of data objects in process-views and base processes may also differ. This work abstracts process-view relevant data from base process relevant data by following four steps.

Step 1 (Determine data members). A base/virtual activity has an *input data* set and an *output data* set. The following notation is used for discussion.

- $d_x^{\mathrm{In}/a_i}/d_x^{Out/a_i}$ represents an input/output data object of a base activity a_i; $D_x^{\mathrm{In}/a_i}/D_x^{Out/a_i}$ represents the input/output data set of a_i.
- $d_x^{\mathrm{In}/va_i}/d_x^{Out/va_i}$ represents an input/output data object of a virtual activity va_i; $D_x^{\mathrm{In}/va_i}/D_x^{Out/va_i}$ represents the input/output data set of va_i.

Fig. 8a–c illustrates three policies for deriving the input data set of a virtual activity. Just as a virtual activity conceals its internal member dependencies, the *Conceal_Internal_Input* policy hides the data flow within a virtual activity. However, according to the *All_Input* policy, the input data set of va_1 comprises

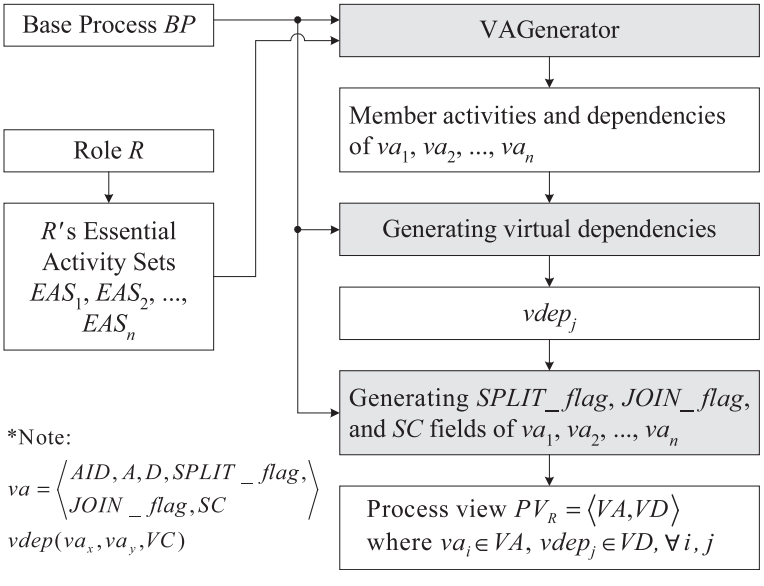


Fig. 7. Deriving a process-view.

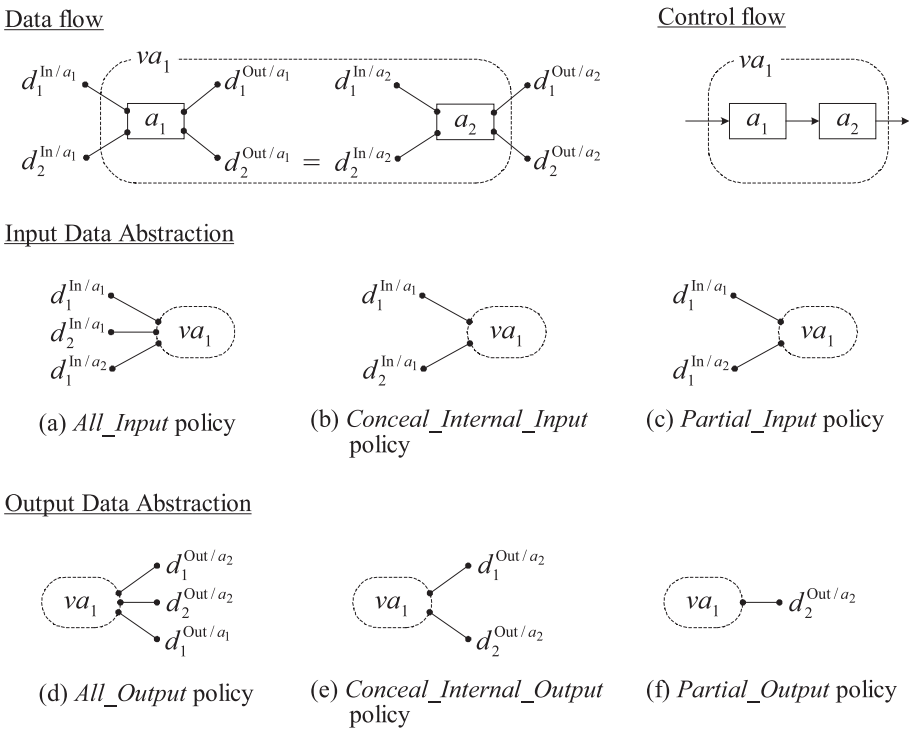


Fig. 8. Policies of data abstraction.

all input data objects that are not produced within va_1. Finally, a modeler may partially select d_1^{In/a_1} and d_1^{In/a_2} as the input data objects of va_1 (*Partial_Input* policy).

Notably, the *All_Input* policy derives the data needed for completing a virtual activity, but the *Conceal_Internal_Input* only derives the data needed for starting a virtual activity. For example, in Fig. 4a, which shows the process-view provided by a notebook company for a LCD manufacturer, the notebook company applies the *All_Input* policy to derive the input data set to complete the virtual activity "sourcing LCD panels". However, the LCD manufacturer only needs to know the data that will enable the virtual activity "sourcing other components"; thus, *Conceal_Internal_Input* is applied.

Formally, the following three policies are used to derive the input data set of a virtual activity $va_i = \langle A_i, D_i \rangle$.

- *All_Input*: $D^{In/va_i} = \{d_x^{In/a_i} | a_i, a_j \in A_i: d_x^{In/a_i} \neq {}^{dOut/a_j}_{y}$ for any $y\}$;
- *Conceal_Internal_Input*: $D^{In/va_i} = \{d_x^{In/a_i} | a_i \in A_i: \nexists a_j \in A_i$ such that $a_i < a_j\}$;
- *Partial_Input*: $D^{In/va_i} = \{d_x^{In/a_i} | a_i \in A_i: d_x^{In/a_i}$ satisfies the condition C that is defined by a process modeler$\}$.

Similarly, the output data set of $va_i = \langle A_i, D_i \rangle$ is derived as follows. Fig. 8d–f illustrates the following three policies, respectively.

- *All_Output*: $D^{Out/va_i} = \{d_x^{Out/a_i} | a_i, a_j \in A_i: d_x^{Out/a_i} \neq d_y^{In/a_j}$ for any $y\}$;
- *Conceal_Internal_Output*: $D^{Out/va_i} = \{d_x^{Out/a_i} | a_i \in A_i: \nexists a_j \in A_i$ such that $a_i > a_j\}$;
- *Partial_Output*: $D^{Out/va_i} = \{d_x^{Out/a_i} | a_i \in A_i: d_x^{Out/a_i}$ satisfies the condition C that is defined by a process modeler$\}$.

Step 2 (Determine data contents). The derived input/output data sets of virtual activity can be further processed according to their contents. The following three classes of operation provide high-level (abstract) specifications for deriving process-view relevant data. The abstract specifications may be associated with several implementations responsible for data trans-

formation and processing, since various data types exist in commercial environments.

- *Aggregation*: this class of operations aggregates given data objects, including *average*, *sum*, and *count*. For example, the total production time of a virtual activity is computed by summarizing the production time of its member activities. Moreover, this operation is applicable to non-numerical data by using the concept hierarchies of an ontology. An ontology offers a set of terms and concepts for describing a specific domain. According to the generation/specialization relationship, concepts in the ontology form a concept hierarchy. By using ontologies, for example, the sum of the capacities of factories in Taiwan and China Cities may be viewed as the capacity of factories in Asia.
- *Symbolization*: consolidated data can be transformed from numerical to symbolic data in order to increase comprehensibility. This operation also relies on ontology. Given a numerical datum, its unit (e.g., amount, length, and weight) is a concept in the ontology, and the value is the modifier of the concept. A higher-level concept in the concept hierarchy may be associated with symbolic modifiers. For example, "19 inch" LCD panels can be transformed into "large size" LCD panels since "size" is above "inch" in the concept hierarchy and is associated with symbolic modifiers—large, medium, and small.
- *Granularity*: consolidated data may require the granularity of the measurement scale to be coarse, as for example, in changing the scale on which production time is measured from days to months. Notably, the *aggregation* operation is applied to the value of the data object, while the *granularity* operation is applied to the scale of value of the data object.

The above operations can be combined to generate more complex abstractions. For example, the manpower can be averaged, and its scale changed from 10 to 100 workers.

Following the example presented in Fig. 4, Fig. 9 illustrates the derivation of input/output data of the virtual activity, "scheduling LCD production", which represents an abstraction of two base activities—"receiving new order" and "scheduling production". Only production data are described for clarity. First, the *Conceal_Internal_Input/output* policy is applied to derive the input/output data set of the virtual activity. Then, the output data objects are further aggregated. Therefore, partners only observe the "production amount" but do not know the production details, including production distribution and stock amount.

Step 3 (Determine access privileges). This step authorizes a participant's access to data objects in a process-view. A process modeler must specify permission according to the responsibilities of a participatory role to prevent unwarranted access. Basically, participant can *read* the data objects in views. However, *write* and *modify* permissions are restricted when these data objects are summarized from multiple data objects in the base process.

Step 4 (Determine presentation formats). The presentation of data objects in a process-view may differ from that in a base process. Differences may include input and output format, for example, by displaying a set of data objects in a pie chart to replace an original tabular form, or inputting data using check boxes instead of pull-down menus. A data object may be presented in many ways to adapt to various operating platforms and user interfaces (such as MS Windows, Mac OS, Motif, etc.), and users' preferences and needs. A process modeler must specify the appropriate presentation of data objects in process-views to facilitate the communication between participants.

Following Fig. 9, step 3 specifies that the access privileges of the data objects, "purchase order" and "production amount" are "modify" and "read", respectively. Then, step 4 decides that the presentation formats of the purchase order and production amount are "HTML.Form" and "HTML.Text_Field", respectively.

3.3. Abstraction of control flow and data flow

As mentioned early, a process-view is an abstraction of a base process. This section further discusses abstraction from the control and data flow aspects. Through bottom-up aggregation of base activities, a

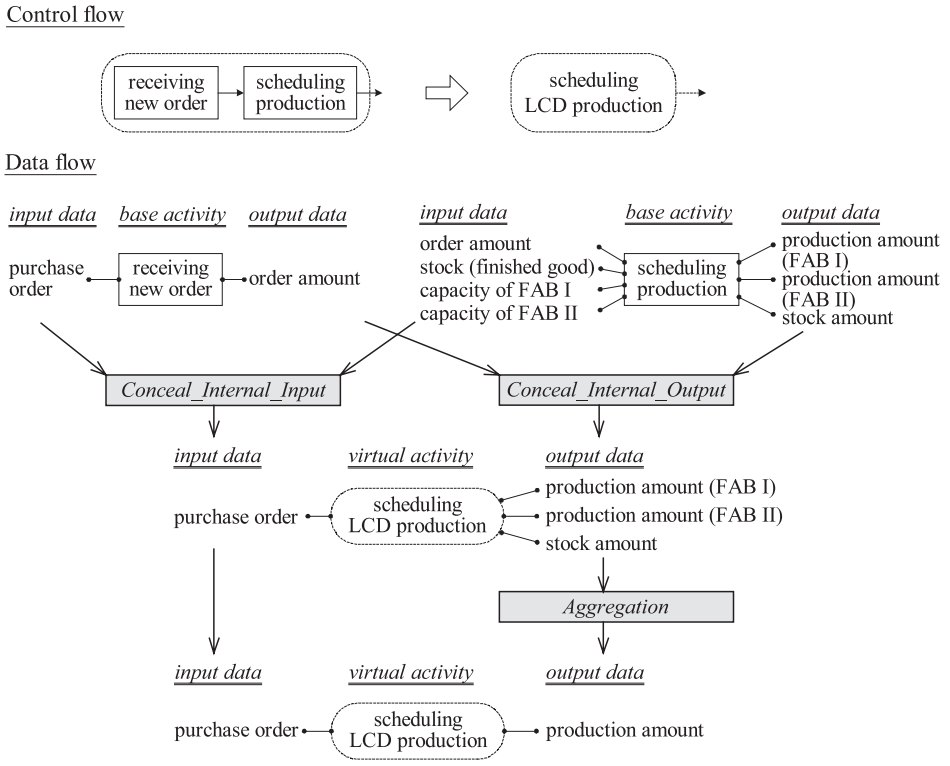


Fig. 9. Deriving input/output data of a virtual activity.

process-view yields a higher-level view of control flow of a base process. In several formal process models, including process algebras and Petri nets [1,19,24,27], the abstraction of a process refers to a selected portion of activities. More specifically, a desired abstraction is obtained by renaming some activities as *silent activities* that are unobservable from the outside. Conventional abstraction may be considered as *partial abstraction* since it provides partial observability of a process. Relative to partial abstraction, the proposed approach is considered to be *aggregate abstraction*, since it provides adaptable granularity of a process via bottom-up aggregation, i.e., a virtual activity may represent an aggregation of a set of base activities.

Partial abstraction does not reveal the progress status of the silent activities of a base process. For example, if a view "$a_1 \rightarrow a_3 \rightarrow a_7$" is partially abstracted from the process shown in Fig. 6, then the view does not expose the progress of silent activities (i.e., a_2, a_4, a_5, and a_6). However, if the proposed approach defines a process-view as "$\{a_1\} \rightarrow \{a_2, a_3, a_4, a_5, a_6\} \rightarrow \{a_7\}$", the progress of the virtual activity which contains $a_2 \ldots a_6$ expresses the aggregated progress of these five base activities. In addition to concealing sensitive details that partial abstraction can provide, process-views also provide aggregated information on a desired process.

Aggregate abstraction generally includes partial abstraction. For example, given a virtual activity va_1 which is an aggregation of base activities a_1, a_2, and a_3, if the weight of a_1 and a_3 are 0 and the weight of a_2 is 1, then va_1 can be viewed as a partial abstraction of a_1, a_2, and a_3. The binary weight implies the notion of visible/invisible or important/unimportant that is the core of partial abstraction. Thus, partial abstraction can be derived from aggregate abstraction by using the concept of weight.

Moreover, the proposed order-preserving approach provides one possible mechanism to derive aggregate abstraction. Under the constraints of Rules 1–3, the derived aggregation preserves original orderings of

base activities. Other approaches may be developed to derive appropriate process abstractions based on the different properties of business processes and commercial contexts.

With respect to data flow, Section 3.2 offers aggregate and partial data abstraction. Besides addressing syntactic issues such as determining privilege and selecting input/output data objects, this work alleviates semantic issues by using ontologies to derive meaningful data.

4. Coordinating B2B workflows through virtual states

This section describes the mechanism that coordinates B2B workflows through activity/process states. The state of a process or activity instance represents the execution status of the instance at a specific point. A virtual state abstract the execution states of base activities/processes contained by a virtual activity/process. During run-time, cooperative partners monitor and control the progress of B2B workflows through the virtual states of virtual activities/processes. First, the states and operations of base activity/process are described. Then, the state mapping rules to coordinate base processes, process-views and integrated processes during run-time are proposed.

4.1. Generic states and operations

A state transition diagram depicts the possible run-time behavior of a process/activity instance. Currently, both Workflow Management Facility (WMF) [22] and Wf-XML [31] support the generic states as shown in Fig. 10, in which WfExecutionObject is a generalization of a process or activity instance [22]. Furthermore, the hierarchy of states imposes super-state/sub-state relationships between them.

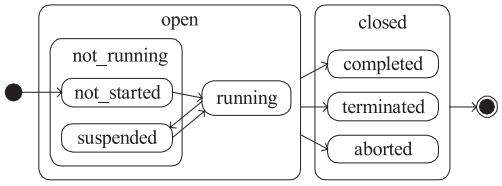


Fig. 10. States of a WfExecutionObject [22].

After a WfExecutionObject is initiated, it is in the *open* state; however, upon completion, it enters the *closed* state. The *open* state has two sub-states: *running* indicates that the object is executing, and *not_running* suggests that the object is quiescent since it is either temporarily paused (in the *suspended* state) or recently initialized and prepared to start (in the *not_started* state). The state *completed* indicates that the object has been completed correctly. Otherwise, the object stops abnormally, i.e., in the *terminated* or *aborted* state.

A state transition diagram also includes a set of operations for monitoring and controlling the progress of a base activity/process instance. The operations (e.g., *suspend*, *terminate*, and *change_state*) that are used to control a WfExecutionObject change the state of a WfExecutionObject as well as its associated WfExecutionObjects. When a request of state change fails, an *InvalidState* exception [22] throws. The operation *get_current_state*, as defined in WMF, returns the current state of a WfExecutionObject instance. In the following section, state function f_s is used to substitute this operation for brevity.

4.2. State mapping

Both base and virtual activities/processes support the same set of the previously mentioned generic states and operations. In this section, consistent mapping of the execution states between virtual processes/activities and its member processes/activities is discussed. Two cooperation scenarios can trigger state mapping. First, virtual activities/processes must respond to the state change that occurred in base activities/processes, i.e., the mapping occurs from base activities/processes to virtual activities/processes. Second, base activities/processes must react to the request to change the state as triggered by virtual activities/processes, i.e., the mapping occurs from virtual activities/processes to base activities/processes. Moreover, state mapping can be considered from process level or activity level.

4.2.1. State mapping between a base process and a process-view

The virtual state of a process-view simply equals the state of its base process. For example, a

process-view is in the suspended state if its base process is also in the same state. However, state mapping between virtual activities and its member activities must follow atomicity rule (Rule 2) as follows.

(i) **Open state.** Atomicity rule implies that a virtual activity is in the open state if at least one member activity is in the open state. *Active degree* (or grade) is introduced to extend the atomicity rule for state mapping. The open state is more active than the closed state. Moreover, according to active degrees, sub-states of the open state are ranked as follows: running>suspended>not_started. Since an activity has been executed for a while prior to suspension, but never runs before the not_started state, the suspended state is more active than the not_started state. Therefore, the atomicity rule is extended as follows. If two or more member activities of a virtual activity *va* are in the open state and states of member activities compose a state set Q, then the (virtual) state of *va* equals the most active state in Q.

(ii) **Closed state.** Atomicity rule also implies that a virtual activity is in the closed state if all members have been evaluated and each fired member is in the closed state. According to the definition of the closed state and its sub-states in Refs. [22,31], an execution object, WfExecutionObject, is stopped in the completed state if all execution objects contained within it are in the completed state. Second, an execution object is stopped in the terminated state if all execution objects contained within it are either in the completed or terminated state, and at least one is in the terminated state. Finally, an execution object is stopped in the aborted state if at least one execution object contained within it is in the aborted state. Therefore, based on these definitions, whether a virtual activity is stopped in the closed state or its sub-state can be determined.

In sum, a virtual activity responds to the state change of member activities according to the following rule. Notably, $f_s(a)/f_s(va)$ denotes the state/virtual state of a member activity *a*/virtual activity *va*.

Rule 4 (State abstraction). Given a virtual activity $va=\langle A, D\rangle$, let $A'=\{a|a\in A$ and a is fired$\}$.

1. If $\exists a\in A$, $f_s(a)=$ open or its sub-state, then let the state set $Q=\{f_s(a)|a\in A\}$, $f_s(va)$ equals the most active state in Q.
2. If a have been evaluated for all $a\in A$, and $f_s(a')=$ completed for all $a'\in A'$, then $f_s(va)=$ completed.
3. If a have been evaluated for all $a\in A$, and $f_s(a')=$ terminated or completed for all $a'\in A'$ and at least one is terminated, then $f_s(va)=$ terminated.
4. If $\exists a\in A'$, $f_s(a)=$ aborted, then $f_s(va)=$ aborted.

Table 1 lists examples of state mapping between a virtual activity and its member activity(s). Fig. 11 depicts the state transitions of a virtual activity that are triggered by the state transitions of its member activities.

When invoking an operation of a virtual process/activity object, the object propagates the operation to underlying base process/activity object. A process-view affects the entire base process, while a virtual activity only affects its member activities. For example, invoking *create_process* operation on

Table 1
State mappings between a virtual activity and its member activities

Virtual activity	Member activities
Open	At least one member activity is in the open state.
Closed	All member activities have been evaluated, and each fired member is stopped in the closed state.
Running	At least one member activity is in the running state.
Not_running	Each member activity's state is either closed or not_running, and at least one member activity is in the not_running state.
Not_started	Each member activity's state is either closed or not_started, and at least one member activity is in the not_started state.
Suspended	No member activity's state is running, and at least one member activity is in the suspended state.
Completed	All member activities have been evaluated and each fired member is stopped in the completed state.
Terminated	All member activities have been evaluated, all fired member are either stopped in the terminated or completed state, and at least one member activity is stopped in the terminated state.
Aborted	At least one fired member activity is stopped in the aborted state.

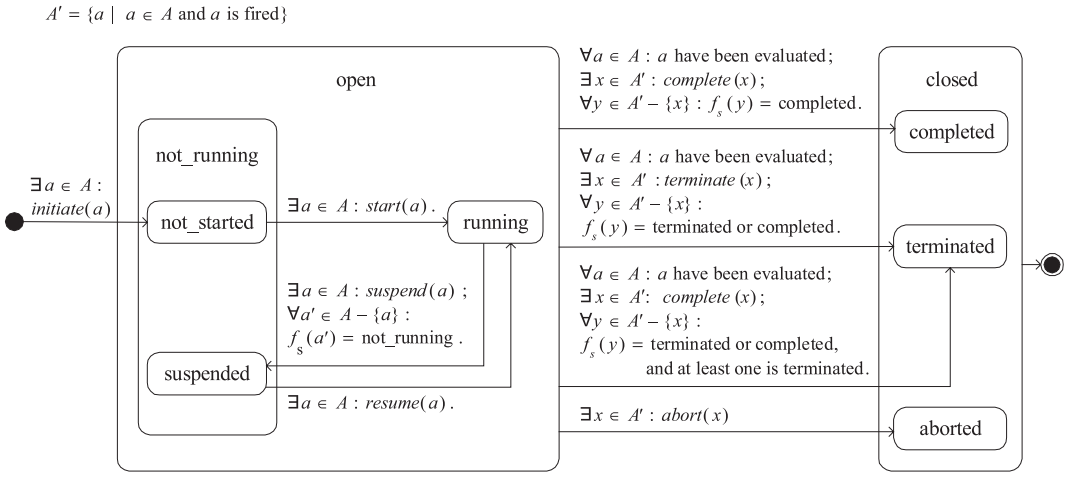


Fig. 11. State transitions of a virtual activity $va = \langle A, D \rangle$.

a process-view definition initiates a process-view and its corresponding base process. However, applying *suspend* operation on a virtual activity only suspends its member activity(s). If an external event or API call alters the state of a virtual activity/process, then the influence of state transition in base activities/processes depends on the following rule:

Rule 5 (State propagation). Given a virtual activity $va = \langle A, D \rangle$. When requesting va to be in state s, if there is a legal transition from state $f_s(a)$ to state s for any activity $a \in A$, then the state of a transfers from $f_s(a)$ to s; otherwise, $f_s(a)$ is not changed. Next, according to Rule 4, the state of va can be derived. If $f_s(va) \neq s$, then an *InvalidState* exception throws and member activities rollback to their original states. If $f_s(va) = s$, then the state transitions of member activities are committed. For a process-view *PV*, when requesting *PV* to be in state s, if its base process *BP* can transfer to s, then the state transitions of *PV* and *BP* are committed; otherwise, an *InvalidState* exception are returned to the request and *PV* and *BP* rollback to their original states.

Rule 4 indicates how to determine the execution state and state transition of a virtual activity according to the states of member activities. Rule 5 indicates how to propagate a request of state change on a virtual activity to underlying member activities, and ensures that state mapping between a virtual activity and member activities is consistent following the request. Therefore, a virtual activity can express the execution status of its member activities appropriately.

4.2.2. State mapping between an integrated process and its underlying processes

Since each virtual activity in an integrated process maps to an activity in the internal process or a partner's process-view, state mapping at the activity level is direct (i.e., one-to-one mapping). If virtual activity *va*'s underlying activities can transfer to state s when requesting that *va* to be in state s, then the transition is committed; otherwise, the transition fails. However, at the process level, when requesting an integrated process *IP* to be in state s, if the states of activities within *IP* can be transferred such that *IP* can be in state s, then the request is committed; otherwise, the transition fails.

5. Prototype system

This section details the architecture and components of our prototype system.

5.1. System architecture

The prototype is currently being implemented mainly using the Java 2 platform, Enterprise Edition

(J2EE) [13]. J2EE offers rich and uniform application programming interfaces (API) to access diverse business systems. For example, the Java Database Connectivity (JDBC) API gives a vendor-independent interface to different DBMSs. Therefore, the prototype is independent of any specific implementation of a DBMS, directory service, or messaging service. The prototype is built on BEA WebLogic (our J2EE server) [9], Microsoft SQL Server (our DBMS) [28], and Microsoft Active Directory (MSAD, our directory service) [18].

Fig. 12 shows the system's architecture. The prototype system uses the Java Naming and Directory Interface (JNDI) API to locate participants that registered in MSAD. The *role designer* maps role definitions onto the organizational units, groups, and users defined in MSAD. The role definitions contain job descriptions and obligations, and are consulted during base/virtual process design. A process modeler employs the *role designer* and the *virtual/base process definition tool* to specify workflow participants (internal workers and trading partners), base processes, process-views, and integrated processes. Section 3 elucidates the procedure for deriving a virtual process, including control and data flow.

During run-time, the *enactment module* consults MSAD to obtain clients' responsibilities, and then submits/receives messages to clients' work-lists or progress coordinators. In this prototype, the enactment module communicates with the clients, the interoperation module, and the invoked applications through the Java Message Service (JMS) API. JMS supports the publish/subscribe asynchronous communication mechanism in the process-view model (described in Section 5.2, *WfRequest* interface). The *interoperation module* governs the sending/receiving of messages through appropriate protocols that have been adopted by the partners, such as e-mails and XML documents, or through the adapters of partners' messaging servers. Currently, the module only adopts XML documents for communication. The client-side *work-list* presents the activities that must be performed by the client. The *progress coordinator* displays abstracted progress information of the workflow in which the client participates (progress monitoring). Moreover, a client can alter the execution state of a public process-view via the coordinator (progress control). In B2B cooperation, trading partners can monitor and control the progress of internal processes through client-side progress coordinators.

5.2. Major interfaces

Fig. 13 shows the major interfaces and their relationships, revised from the WMF specification, of the prototype system in UML (Unified Modeling Language) notation. *WfBaseProcessMgr*/*WfVirtualProcessMgr* represents a base/virtual process definition that generates and locates base/virtual process instances. *WfDataUtility* provides common utilities useable in data abstraction, as stated in Section 3.2. For example, *WfDataUtility.symbolize(19, inch, LCD panel)* returns "large size" as a symbolic descriptor of LCD panels. The prototype uses the Java binding of Open Knowledge Base Connectivity (OKBC) API [21] to access the ontology. An

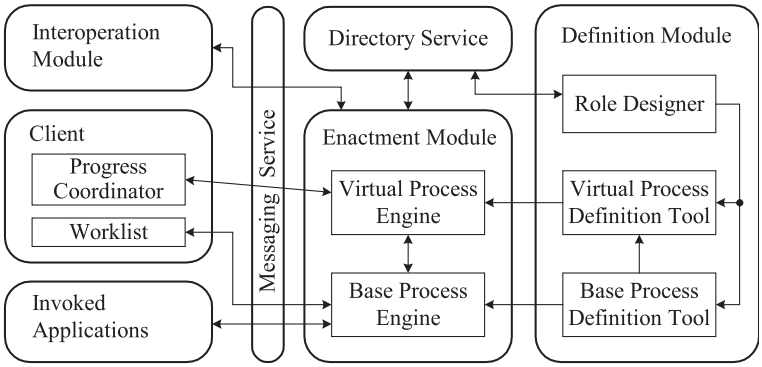


Fig. 12. Architecture of the prototype system.

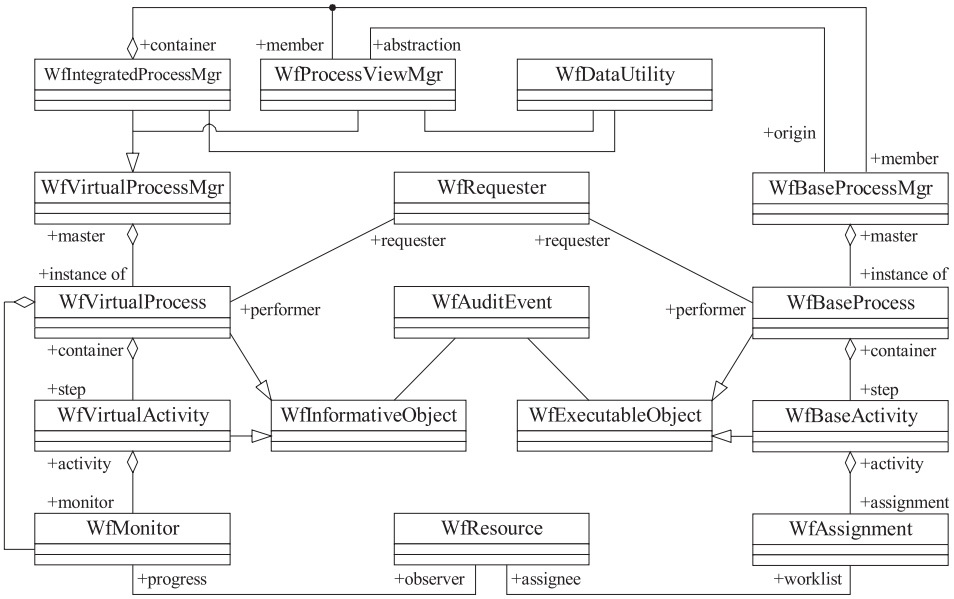


Fig. 13. Class diagram of process-view model.

experimental ontology is constructed using Ontolingua [23].

WfExecutableObject is an abstract base class that defines common attributes, states, and operations for WfBaseProcess and WfBaseActivity. *WfBaseProcess* signifies a base process instance and *WfBaseActivity* stands for a base activity in a base process. However, a virtual process/activity is not inherited from WfExecutionObject since it is not actually performed but is used to present abstracted progress status. *WfInformativeObject* defines common attributes, states, and operations for *WfVirtualProcess* and *WfVirtualActivity*. Moreover, *WfAssignment* associates base activities with necessary resources that are represented by *WfResource*. Similarly, *WfMonitor* associates virtual activities/processes with participants (trading partners) that are represented by WfResource. Finally, *WfRequest* is introduced to model the publish/subscribe communication pattern [6] in distributed systems. A requester can subscribe to (receive) the events published by a performer, and performer-produced events are sent to all registered requesters. Therefore, when the state of an activity/process is changed, a status-change event, as modeled by *WfAuditEvent*, is created to inform registered requesters. The asynchronous message-based approach is suitable for loosely coupled B2B interoperation over the Internet.

6. Discussion

Integration of B2B workflows has attracted considerable interest. This section reviews related work and standards, and discusses the strengths of the process-view model.

6.1. Modeling B2B interactions

During cross-enterprise collaboration, each participatory enterprise does not expose its internal processes to preserve autonomy. However, successful collaboration among multiple enterprises requires the sharing of information. Therefore, modelers should provide various external interfaces of an internal process that not only conceal sensitive information but also reveal that which is essential to cooperation. External partners can monitor and control the progress of an internal process via these external interfaces. Modelers must also incorporate the process information provided via partners' external interfaces to internal processes in order to automate B2B work-

flows. Furthermore, an ideal process model should uniformly describe internal processes and external interfaces to increase the comprehensibility of the model. The following subsection reviews related work and elucidates the decision support of the proposed approach to B2B cooperation.

6.1.1. Conventional approaches

WfMC identifies four B2B interoperation models—chained service, nested sub-process, peer-to-peer, and the parallel-synchronized model [29]. Currently, most workflow related standards, e.g., Wf-XML [31] and WMF [22], only support chained service and nested sub-process models.

The nested sub-process model, also known as the service activity model, employs an activity as an external interface to abstract an entire process that is implemented by another enterprise. Its notion is based on service information hiding [2], that is, a consumer does not need to know the internal process structure of a provider. Therefore, the activity can be viewed as a business service performed by an external partner. Various investigations regarding B2B workflow management are based on the service activity model, for example, Refs. [20,26]. A consumer can monitor service progress via service activity states. However, a service consumer only knows whether the invoked process (enacted by the service provider) has started or is completed. Such poor progress data increase the uncertainty in the decision space and cause inefficient coordination among the cooperating enterprises.

The typical service activity model prohibits external partners from seeing the structure of internal processes, such that a service consumer is notified only at the end of the service provider's process. Kuechler et al. [15] proposed the monitored-nested model, in which the sequence and states of activities within a provider's process are observable by service consumers. A service consumer may choose interesting states to be monitored. The consumer is automatically notified when the monitored states are reached. Therefore, cooperating enterprises can obtain more progress information about one another to improve coordination efficiency. However, trading partners can see the structures of internal processes since the monitored-nested model does not impose process abstraction. In contrast, the process-view model only shares abstracted processes that conceal sensitive process information. Partners can obtain abstracted progress data. Moreover, the process-view model enables an enterprise to provide various views of an internal process, each suited to one of the diverse partners. Exposed progress data is thus customized to satisfy partners' needs.

The conventional service activity model only supports WfMC-specified activity states [29], e.g., initialized, running, suspended, completed, and aborted. To semantically reveal the progress status of a service provider's process, Georgakopoulos et al. [8] proposed the Collaboration Management Infrastructure (CMI) which enables modelers to define application-specific states that extend from standard activity ones. Although only generic states and operations are applied in the process-view model, the adopted hierarchy of states facilitates further extension regarding specific application domains as in CMI approach.

Casati and Discenza [3] introduced two types of event nodes to explicitly define the tasks that exchange data with partners. Send nodes notify events to partners' processes, while request nodes demand events from partners' processes. Events that are transmitted among enterprises are managed by an event service. The authors aim at embedding the interaction mechanisms into conventional activity-based process models. However, different partners may receive identical progress data since event nodes are embedded in an internal process. Moreover, an internal process becomes more complicated as the number of event nodes increases. In contrast, each process-view is unique to a partner, such that different partners obtain different progress data. When cooperation has ended, an enterprise needs only to delete related process-views that will not affect internal processes. Separating interoperation parts from internal processes keeps internal process definitions stable and concise.

There are some similarities between the process-view model and [4]. Although event nodes are not explicitly specified in the process-view model, each virtual activity may receive and send events to communicate with partners. A process-view functions as an event adaptor between internal and partners' workflows. That is, a process-view forwards the state-change events, as activated by an internal process, to partners (see Rule 4, state abstraction), and dispatches the state-change events, as requested by

partners, to an internal process (see Rule 5, state propagation). Moreover, our implementation of the process-view model also adopts the publish-subscribe event model to exchange data among cooperating workflows.

6.1.2. Process-view approach

The advantages of the process-view model are outlined below.

• **Supporting decision making for process modelers.** Process modelers must create a trade-off between autonomy and cooperation when designing B2B workflows. Restated, they must determine how much information is to be exposed in order to facilitate coordination. Conventional approaches, as discussed above, are restricted by the original granularity of process definitions that is not intended for partners. Therefore, determining which parts of internal processes should be shared with partners is extremely difficult. Namely, the exposed parts may be too detailed or too rough to enable cooperation. The process-view model enables a modeler flexibly and systematically to generate various levels of abstraction (granularity) of an internal process. A process-view can be viewed as a compromise between privacy and publicity.

Based on various commercial relationships, an enterprise can use the process-view model to provide partners with appropriate abstracted processes. Therefore, participatory enterprises can obtain appropriate and real-time progress information regarding trading partners. In such a collaborative environment, enterprises are better able to make correct decisions in rapidly changing markets.

• **Uniform representation of interoperation interfaces and internal processes.** This work proposes process-views as public interfaces for external partners. The interfaces are represented as (virtual) processes that are represented in the same way as internal processes. Moreover, the mechanism for controlling and monitoring the public interfaces, as provided by partners, and internal processes are identical; that is, using states and state transition functions. Such uniform representation and usage allows the process-view model to be easily understood and adopted by WfMS administrators and users.

• **Providing integrated progress data of internal and partners' processes.** Integrating internal processes and partner-provided interfaces is straightforward since they are identically represented. An integrated process creates a customized view of the supply chain and presents progress data of both internal and partners' processes. Providing integrated progress data facilitates decision-making and helps workflow administrators to analyze and monitor the performance of B2B workflows.

6.2. Process abstractions for B2B workflows

The "views" described in Koetsier et al. [14] (contract view) and Chiu et al. [5] (workflow view) provided partial visibility of a process to support interorganizational workflows. That is, their view includes some selected activities of a process. Although those studies do not use the term "abstraction", their ideas are similar to partial abstraction as stated in Section 3.3. Moreover, they do not focus on systematic means to generate process abstractions and to manage the state mappings between virtual and base processes.

In contrast, a process-view is derived from bottom-up aggregation of activities to provide various levels of aggregate abstraction of a process. B2B workflows are coordinated using virtual states of process-views. The generic states/operations defined in existing standards were adopted to utilize those standards as a backbone in order to integrate heterogeneous and distributed systems in multi-enterprise cooperation. This work has developed a uniform approach to manage state mappings between base processes/activities and virtual processes/activities. Moreover, the adopted hierarchy of states can easily include the notion of application-specific states (proposed by CMI [8]) to express semantic progress information.

6.3. Composing virtual supply chains

WISE [16] proposed a framework for composing a virtual business process by using the process interfaces as provided by cooperating enterprises. Additionally, CrossFlow [10] proposed a framework, which is based on a service contract, to manage WfMS cooperation between service providers and consumers. Those projects focus on providing brokering architectures to exchange business processes as business services. This work contributes a systematic

approach from which external interfaces can be derived. The process-view model can be extended to support the trading architectures that WISE and CrossFlow proposed.

Some standardization approaches, e.g., RosettaNet [25], have defined a set of common processes, also known as Partner Interface Processes (PIP), adopted by trading communities, to enable supply chain integration. For example, RosettaNet specified a standardized process for querying order status. Thus, participatory enterprises can be coordinated through common PIP. However, cooperating enterprises must identify the relationships between internal processes and the PIP. In such circumstances, the process-view model acts as a wrapper program that maps existing workflows to the PIP.

Conventional workflow-based approaches may use a single global view to support the supply chain, as in [12,20]. However, in our approach, each participatory enterprise can establish its own integrated view of a supply chain (personalized supply chain). The proposed approach is more realistic since companies often demand different supply chain information.

7. Conclusion

This work has presented a process-view-based coordination model for B2B workflow management. A process-view is an abstracted process that can be viewed as an external interface of an internal process. Representation and usage of process-views are compatible with conventional activity-based process models. This work also offers a systematic approach for creating an interoperation interface, that is, a process-view, which shares essential progress data to support cooperation but conceals internal process structures to preserve autonomy. In addition to monitoring and controlling the progress of partners' workflows via their process-views, an enterprise can obtain personalized supply chain data from its integrated process. Moreover, enterprises interact through virtual states of process-views that conform to interoperation standards. Therefore, distributed, heterogeneous and autonomous WfMSs can be integrated in an open environment. The proposed approach mitigates the shortcomings of inter-enterprise workflow collaboration.

Our future work will address three themes. First, an enterprise must verify the correctness of B2B workflows before enforcing cooperation. An integrated process provides a proving ground for B2B workflow integration. Future enhancement will support tools for verifying integrated process definitions. Second, handling the four WfMC interoperation models by using process-views will be investigated. Finally, exceptions (for example, suspension of production due to an earthquake) always occur in commercial markets, so that B2B workflows do not always operate as expected. Some studies have proposed advanced workflow models to manage exceptions in WfMSs, including Refs. [4,11]. The process-view model should incorporate exception-handling mechanisms to establish a robust collaboration environment.

Acknowledgements

This research was supported by the National Science Council of the Republic of China under the grant NSC 91-2416-H-009-008.

References

[1] J.C.M. Baeten, W.P. Weijland, Process Algebra, Cambridge Univ. Press, Cambridge, UK, 1990.
[2] A.P. Barros, A.H.M. ter Hofstede, Towards the construction of workflow—suitable conceptual modeling techniques, Information Systems Journal 8 (4) (1998) 313–337.
[4] F. Casati, A. Discenza, Modeling and managing interactions among business processes, Journal of Systems Integration 10 (2) (2001) 145–168.
[3] F. Casati, S. Ceri, S. Paraboschi, G. Pozzi, Specification and implementation of exceptions in workflow management systems, ACM Transactions on Database Systems 24 (3) (1999) 405–451.
[5] D.K.W. Chiu, K. Karlapalem, Q. Li, Views for inter-organization workflow in an E-commerce environment, Proceedings of the 9th IFIP Working Conference on Database Semantics (DS-9), Hong Kong, China, 2001.
[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, Boston, MA, USA, 1995.
[7] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management—from process modeling to workflow automation infrastructure, Distributed and Parallel Databases 3 (2) (1995) 119–153.
[8] D. Georgakopoulos, H. Schuster, A. Cichocki, D. Baker, Managing process and service fusion in virtual enterprises, Information Systems 24 (6) (1999) 429–456.

[9] P. Gomez, P. Zadrozny, Professional J2EE Programming with BEA WebLogic Server, Wrox Press, Birmingham, UK, 2000.
[10] P. Grefen, K. Aberer, Y. Hoffner, H. Ludwig, CrossFlow: cross-organizational workflow management in dynamic virtual enterprises, Computer Systems Science and Engineering 15 (5) (2000) 277–290.
[11] C. Hagen, G. Alonso, Exception handling in workflow management systems, IEEE Transactions on Software Engineering 26 (10) (2000) 943–958.
[12] K. Hiramatsu, K. Okada, Y. Matsushita, H. Hayami, Inter-workflow system: coordination of each workflow system among multiple organizations, Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98), 1998.
[13] N. Kassem, Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition, Addison-Wesley, Boston, MA, USA, 2000.
[14] M. Koetsier, P. Grefen, J. Vonk, Contracts for cross-organizational workflow management, Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies, London, UK, 2000.
[15] W. Kuechler, V.K. Vaishnavi, D. Kuechler, Supporting optimization of business-to-business E-Commerce relationships, Decision Support Systems 31 (3) (2001) 363–377.
[16] A. Lazcano, G. Alonso, H. Schuldt, C. Schuler, The WISE approach to electronic commerce, Computer Systems Science and Engineering 15 (5) (2000) 345–357.
[17] D.-R. Liu, M. Shen, Workflow modeling for virtual processes: an order-preserving process-view approach, Information Systems 28 (6) (2003) 505–532.
[18] A.G. Lowe-Norris, Windows 2000 Active Directory, O'Reilly, Cambridge, CA, USA, 2000.
[19] R. Milner, A calculus of communication systems, Lecture Notes in Computer Science, vol. 92, Springer-Verlag, Berlin, Germany, 1980.
[20] M.Z. Muehlen, F. Klien, AFRICA: workflow interoperability based on XML-messages, Proceedings of the 1st International Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00), Stockholm, Sweden, 2000.
[21] OKBC, http://www.ksl.stanford.edu/software/OKBC.
[22] Object Management Group, Workflow Management Facility, Document number formal/00-05-02, April 2000, http://www.omg.org.
[23] Ontolingua, http://www.ksl.stanford.edu/software/ontolingua.
[24] L. Pomello, G. Rozenberg, C. Simone, A survey of equivalence notions of net based systems, in: G. Rozenberg (Ed.), Advances in Petri Nets 1992, Lecture Notes in Computer Science, vol. 609, Springer-Verlag, Berlin, Germany, 1992, pp. 410–472.
[25] RosettaNet, http://www.rosettanet.org.
[26] K. Schulz, Z. Milosevic, Architecting cross-organizational B2B interactions, Proceedings of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Los Alamitos, CA, USA, 2000.
[27] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, Journal of the ACM 43 (3) (1996) 555–600.
[28] R. Vieira, Professional SQL Server 2000 Programming, Wrox Press, Birmingham, UK, 2000.
[29] Workflow Management Coalition, The Workflow Reference Model, Technical report WfMC TC-1003, Jan. 19, 1995, http://www.wfmc.org.
[30] Workflow Management Coalition, Interface 1: Process Definition Interchange Process Model, Technical report WfMC TC-1016-P, Nov. 12, 1998, http://www.wfmc.org.
[31] Workflow Management Coalition, Interoperability Wf-XML Binding, Technical report WfMC TC-1023, May 1, 2000, http://www.wfmc.org.

Duen-Ren Liu received the BS and MS degrees in Computer Science and Information Engineering from the National Taiwan University, Taiwan, in 1985 and 1987, respectively, and the PhD degree in Computer Science from the University of Minnesota, in 1995. He is currently an associate professor of the Institute of Information Management, National Chiao Tung University, Taiwan. His research interests include database systems, information systems, electronic commerce, workflow systems, and Internet applications. Dr. Liu is an associate member of the IEEE, and a member of the ACM.

Minxin Shen is a PhD student at Institute of Information Management, National Chiao Tung University, Taiwan. He received his Bachelor's degree, with a double major in Business Administration and Computer Science, from Feng Chia University, Taiwan in 1998. His current research interests include workflow management systems, computer supported cooperative work, Web services, and electronic commerce.