

Coordinating Interorganizational Workflows Based on Process-Views

Minxin Shen and Duen-Ren Liu

Institute of Information Management, National Chiao Tung University, Taiwan

{shen, dliu}@iim.nctu.edu.tw

Abstract. In multi-enterprise cooperation, an enterprise must monitor the progress of private processes as well as those of the partners to streamline interorganizational workflows. In this work, a process-view model, which extends beyond the conventional activity-based process model, is applied 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 according to diverse commercial relationships, and establish an integrated process that is comprised of private processes as well as the process-views that these partners provide. Participatory enterprises can obtain appropriate progress information from their own integrated processes, allowing them to collaborate more effectively. Furthermore, interorganizational workflows are coordinated through virtual states of process-views. This work develops a regulated approach to map the states between private processes and process-views. The proposed approach enhances prevalent activity-based process models to be adapted in open and collaborative environments.

1 Introduction

During cross-enterprise collaboration, each participatory enterprise needs to conceal its private processes to preserve autonomy. However, successful collaboration among multiple enterprises requires information sharing. Therefore, modelers should provide various external interfaces of a private process that not only conceals sensitive information but also reveals that which is essential to cooperation. Outside partners can monitor and control the progress of a private process through these external interfaces. To automate interorganizational workflows, modelers must also incorporate the process information as provided via partners' external interfaces into internal processes. Furthermore, an ideal process model should describe internal processes and external interfaces uniformly to increase comprehensibility.

Our previous study [9] proposed a process-view model that enhances the capability of process abstraction in conventional activity-based process models [4]. A process-view, i.e., a virtual process, is abstracted from an actual process. According to distinct organizational role's 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. Therefore, this work extends the process-view model to endeavor these issues.

A process-view abstracts critical commercial secrets and is an external interface of an internal process. An enterprise can design process-views, which are unique to each partner. Process-views of participatory enterprises comprise a collaboration workflow. Furthermore, the *virtual states* of a process-view present progress status of an internal process. An enterprise can monitor and control the progress of partners through the virtual states of their process-views. The proposed approach provides a modeling tool to describe interorganizational workflows as well as an interoperation mechanism to coordinate autonomous, heterogeneous and distributed workflow management systems (WfMSs).

The remainder of this paper is organized as follows. Section 2 presents the process-view model and its applications within inter-enterprise cooperation. Section 3 summarizes the procedure of defining an ordering-preserved process-view presented in [9]. Next, Section 4 presents the coordination of interorganizational workflows through the virtual states of process-views and then Section 5 discusses some properties of process-view based approach and related work. Conclusions are finally made in Section 6.

2 Process-View Based Coordination Model

A process that may have multiple process-views is referred to herein as a *base process*. A process-view is an abstracted process derived from a base process to provide abstracted process information. Based on the process-view definition tool, a modeler can define various process-views to achieve different levels of information concealment.

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 $dep(x, y, C)$ indicates that x is completed and C is true is one precondition of whether activity y can start.
2. BA is a set of activities. An activity is a 4-tuple $\langle AID, SPLIT_flag, JOIN_flag, SC \rangle$, where (a) AID is a unique activity identifier within a process. (b) $SPLIT_flag/JJOIN_flag$ may be “NULL”, “AND”, or “XOR”. NULL indicates this activity has only one outgoing/incoming dependency (Sequence). AND/XOR indicates the AND/XOR JOIN/SPLIT ordering structures defined by WfMC [15]. (c) SC is the *starting condition* of this activity. 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 all incoming dependencies' conditions.
3. $\forall x, y \in BA$, (a) if $\exists dep(x, y, C)$, then x and y are adjacent; (b) the *path* from x to y is denoted by $x \rightarrow y$. (c) x is said to have a higher *order* than y if $\exists x \rightarrow y$, i.e., x proceeds before y , and their *ordering relation* is denoted by $x > y$ or $y < x$. If $\nexists x \rightarrow y$ and $y \rightarrow x$, i.e., x and y proceed independently, their ordering relation is denoted by $x \infty y$.

2.1 Virtual Process: A Process-View

A process-view is generated from either base processes or other process-views and is considered a *virtual 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 *path* from va_i to va_j is denoted by $va_i \rightarrow va_j$; 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 virtual dependency is used to connect two virtual activities in a process-view. Figure 1 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.

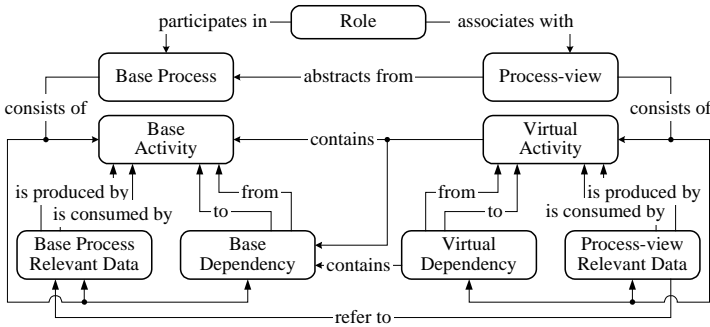


Fig. 1. Process-view model

2.2 Process-View Based Coordination

Figure 2 illustrates the cooperation scenario and system components, 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 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.

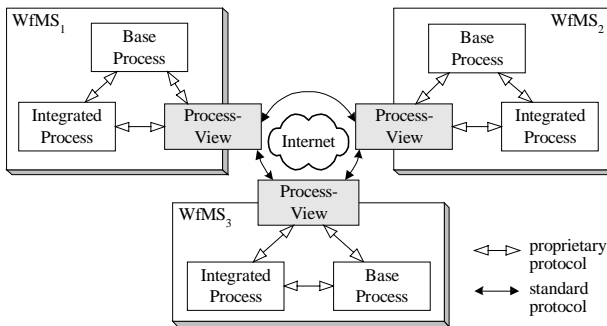


Fig. 2. System architecture and interaction scenario

A process-view is an external view (or interface) of the private 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 private processes and partners’ process-views. Notably, the integrated process is also a virtual process. Each of its virtual activity/dependency is either a base one of a private base process or a virtual one of partners’ process-views.

As a base activity/process, a virtual activity/process is associated with a set of states to present 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 a private 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*. Figure 3 illustrates the three phases for the cooperation between enterprise A and B.

Base process phase is the traditional 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, designing a process-view 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, or a personalized view of an interorganizational workflow through consolidating private base process and partners’ process-views.

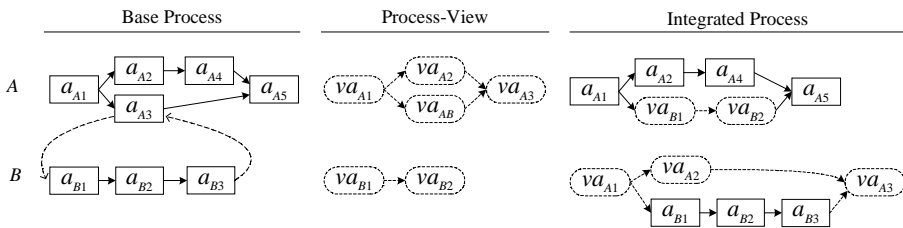


Fig. 3. Three phases of designing interorganizational workflows

3 Ordering-Preserved Process-View

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 ordering-preserved approach to derive a process-view from a base process has been presented in [9] and is summarized in this section. The ordering-preserved

approach ensures that the original execution order in a base process is preserved. A *legal* virtual activity in an ordering-preserved process-view must follow three rules:

Rule 1 (Membership). A virtual activity's member may be a base activity or a previously defined virtual activity.

Rule 2 (Atomicity). A virtual activity, an atomic unit of processing, is completed if and only if each activity contained by it either has been completed or is never executed. A virtual activity is started if and only if one activity contained by it is started. In addition, if an ordering relation, \Re (i.e., $>$, $<$ or ∞), between two virtual activities is found in a process-view, then an implied ordering relation \Re exists between these virtual activities' respective members.

Rule 3 (Ordering preservation). The implied ordering relations between two virtual activities' respective members must conform to the ordering relations in the base process.

Based on the above rules, virtual activities and dependencies in an ordering-preserved process-view are formally defined as follows:

Definition 3 (Virtual Activity). For a base process $BP = \langle BA, BD \rangle$, a virtual activity va is a 6-tuple $\langle VAID, A, D, SPLIT_flag, JOIN_flag, SC \rangle$, where

1. $VAID$ is a unique virtual activity identifier within a process-view.
2. A is a nonempty set, and its members follow three rules:

Its members may be base activities that are members of BA or other previously defined virtual activities that are derived from BP .

The fact that va is completed implies that each member of A is either completed or never executed during run time; the fact that va is started implies that one member of A is started.

$\forall x \in BA, x \notin A$, the ordering relations between x and all members (base activities) of A are identical in BP , i.e., $\forall y, z \in BA, y, z \in A$, if $x \Re y$ exists in BP , then $x \Re z$ also exists in BP .

3. D is a nonempty set, and its members are dependencies whose succeeding activity and preceding activity are contained by A .
4. $SPLIT_flag/JJOIN_flag$ may be "NULL" or "MIX". NULL suggests that va has only one outgoing/incoming virtual dependency (Sequence) while MIX indicates that va has more than one outgoing/incoming virtual dependency.
5. SC is the *starting condition* of va .

The $SPLIT_flag$ and $JJOIN_flag$ cannot simply be described as AND or XOR since va is an abstraction of a set of base activities that may associate 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. The abbreviated notation $va = \langle A, D \rangle$ is used for brevity.

Definition 4 (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 $vdep(va_i, va_j, VC_{ij}) = \{ dep(a_x, a_y, C_{xy}) \mid dep(a_x, a_y, C_{xy}) \in BD, a_x \in A_i, a_y \in A_j \}$, where the virtual condition VC_{ij} is a Boolean combination of C_{xy} .

The procedure of defining an ordering-preserved process-view is summarized as follows: A process modeler must initially select essential activities. The process-view definition tool then generates a legal minimum virtual activity that encapsulates these essential activities automatically. 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 ordering fields (*JOIN/SPLIT_flag*) and starting condition (*SC*) of each virtual activity automatically. [9] presents the algorithm that implements the process-view definition tool.

4 Coordinating Inter-enterprise Processes through Virtual States

In this section, the mechanism that coordinates inter-enterprise processes through activity/process states is described. During run time, cooperative partners monitor and control the progress of inter-enterprise processes through the execution states (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

The state of a process or activity instance represents the execution status of the instance at a specific point. A state transition diagram depicts the possible run-time behavior of a process/activity instance. Currently, both WMF and Wf-XML support the generic states as shown in Figure 4, in which *WfExecutionObject* is a generalization of a process or activity instance [11]. Furthermore, the hierarchical structure of states imposes superstate/substate relationships between them.

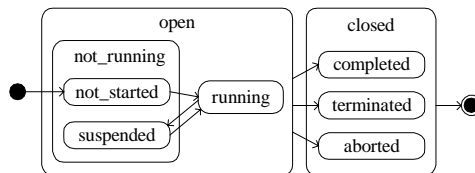


Fig. 4. States of a *WfExecutionObject* [11]

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

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*. 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 employed 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.

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.

Active state. Atomicity rule states that a virtual activity is *active*, i.e., in *open* state, if at least one member activity is active. *Active degree* (or grade) of active states is introduced to extend the atomicity rule for state mapping. Active states 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. The atomicity rule is extended as follows: if two or more member activities of a virtual activity va are active and states of member activities compose a state set Q , then the (virtual) state of va equals the most active state in Q .

Inactive state. Atomicity rule also states that a virtual activity is *inactive*, i.e., in *closed* state, if all members are either inactive or never initialized. According to the definition of the *closed* state and its substates in [11, 14], an execution object, `WfExecutionObject`, is stopped in *completed* state if all execution objects contained within it are *completed*. Second, an execution object is stopped in *terminated* state if all execution objects contained within it are either *completed* or *terminated*, and at least one is *terminated*. Finally, an execution object is stopped in *aborted* state if at least one execution object contained within it is *aborted*. Therefore, based on these definitions, the state of an inactive virtual activity 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 $va = \langle A, D \rangle$. If $\exists a \in A, f_s(a) = open$ or its substate, then let the state set $Q = \{f_s(a), \forall a \in A\}$, $f_s(va)$ equals the most active state in Q . If $\forall a \in A, f_s(a) = closed$ or its substate, and $\exists a \in A, f_s(a) = aborted$, then $f_s(va) = aborted$. If $\forall a \in A, f_s(a) = either terminated or completed$, and $\exists a \in A, f_s(a) = terminated$, then $f_s(va) = terminated$. If $\forall a \in A, f_s(a) = completed$, then $f_s(va) = completed$.

Figure 5 depicts the state transitions of a virtual activity that are triggered by the state transitions of its member activities.

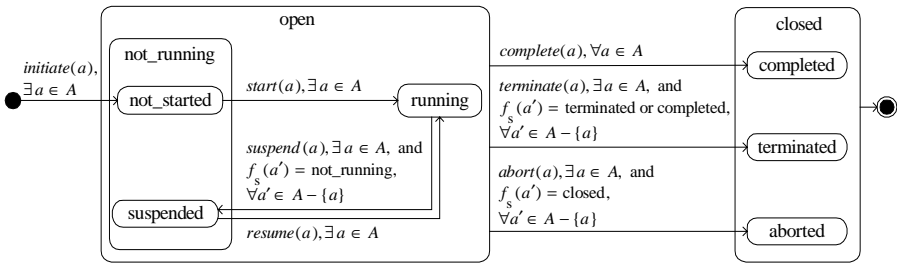


Fig. 5. State transitions of a virtual activity $va = \langle A, D \rangle$

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 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). For a virtual activity $va = \langle A, D \rangle$, when requesting va to be in state s , $\forall a \in A$, if a transition from state $f_s(a)$ to state s is valid, 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 [11] 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.

State Mapping between an Integrated Process and Its Underlying Processes

Since each virtual activity in an integrated process only maps to an activity in the private process or a partner’s process-view, state mapping at the activity level is direct. If virtual activity a ’s underlying activity can transfer to state s when requesting that a to be in state s , then the transition is committed; otherwise, the transition fails. Similarly, if the activities within an integrated process IP can transfer to state s when requesting IP to be in state s , then the request is committed; otherwise, the transition fails.

5 Discussion

Several investigations defined and implemented *interaction points* among cooperating enterprises. Casati and Discenza [2] introduced event nodes in a process to define the tasks that exchange information with partners. Lindert and Deiters [8] discussed the data that should be described in interaction points from various aspects. Van der Aalst [13] applied a message sequence chart to identify interactions among enterprises. In these approaches, the external interface is the set of interaction points. Above

investigations ensure that only interaction points are public and the structure of internal processes remains private.

A widely used modeling method uses an activity as an external interface to abstract a whole process enacted by another enterprise. The notion is based on service information hiding [1], i.e., a consumer does not need to know the internal process structure of a provider. Therefore, the activity can be viewed as a business service that an external partner enacts. This approach resembles the traditional nested sub-process pattern in which a child sub-process implements an activity within a parent process. Various investigations are based on the paradigm of *service activity* such as [5, 10, 12]. Via service activity states, a consumer can monitor service progress. Most approaches only support Workflow Management Coalition (WfMC) specified activity states [15] to comply with interoperation standards such as Wf-XML [14] and Workflow Management Facility (WMF) [11]. To reveal a more semantic status of the service provider's process, CMI [5] enables modelers to define application-specific states that extend from standard activity ones.

This work focuses mainly on supporting collaborative workflow modeling and interoperation. Conventional approaches are restricted by original granularity of process definitions that is not intended for outside partners. Therefore, determining which parts of private processes should be revealed to partners is extremely difficult. Process-view model enables a modeler to generate various levels of abstraction (granularity) of a private process flexibly and systematically. A process-view can be considered a compromised solution between privacy and publicity.

Meanwhile, in parallel with the publication of our work, Chiu et al. proposed a workflow view that provides partial visibility of a process to support interorganizational workflow in an e-commerce environment [3]. A workflow view contains selected partial activities of a process. In contrast, a process-view is derived from bottom-up aggregation of activities to provide various levels of aggregated abstraction of a process. Interorganizational workflows are coordinated through virtual states of process-views. The generic states/operations that was defined in standards were adopted to use the existing standards as a backbone to integrate heterogeneous and distributed systems in multi-enterprise cooperation. This work has developed a regulated approach to manage the states mapping between base processes/activities and virtual processes/activities. Although only generic states and operations are discussed herein, the adopted hierarchical structure of states facilitates further extension regarding specific application domains, e.g., the CMI approach [5].

WISE [7] proposed a framework to compose a virtual business process through the process interfaces of several enterprises. In addition, CrossFlow [6] proposed a framework, which is based on service contract, to manage WfMS cooperation between service providers and consumers. These projects focus on providing broking architectures to exchange business processes as business services. Our contribution is 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.

6 Conclusion

A process-view model to conduct interorganizational workflow management was presented herein. Notably, a process-view is an abstracted process that can be viewed

as an external interface of a private process. The proposed approach not only preserves the privacy of an internal process structure, but also achieves progress monitoring and control. Moreover, enterprises interact through the 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 alleviates the shortcomings of inter-enterprise workflow collaboration.

Acknowledgements. The work was supported in part by National Science Council of the Republic of China under the grant NSC 89-2416-H-009-041.

References

1. A. P. Barros and A. H. M. ter Hofstede, "Towards the Construction of Workflow - Suitable Conceptual Modelling Techniques", *Information Systems Journal*, 8(4), pp. 313-337, 1998.
2. F. Casati and A. Disenza, "Modeling and Managing Interactions among Business Processes", *Journal of Systems Integration*, 10(2), pp. 145-168, 2001.
3. D. K. W. Chiu, K. Karlapalem, and 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, April 24-28, 2001.
4. D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management - from Process Modeling to Workflow Automation Infrastructure", *Distributed and Parallel Databases*, 3(2), pp. 119-153, 1995.
5. D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker, "Managing Process and Service Fusion in Virtual Enterprises", *Information Systems*, 24(6), pp. 429-456, 1999.
6. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig, "CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises", *Computer Systems Science & Engineering*, 15(5), pp. 277-290, 2000.
7. A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler, "The WISE Approach to Electronic Commerce", *Computer Systems Science & Engineering*, 15(5), pp. 345-357, 2000.
8. F. Lindert and W. Deiters, "Modeling Inter-Organizational Processes with Process Model Fragments", *Proceedings of GI workshop Informatik'99*, Paderborn, Germany, Oct. 6, 1999.
9. D.-R. Liu and M. Shen, "Modeling Workflows with a Process-View Approach", *Proceedings of the 7th International Conference on Database Systems for Advanced Applications (DASFAA'01)*, pp. 260-267, Hong Kong, China, April 18-22, 2001.
10. M. z. Muehlen and F. Klien, "AFRICA: Workflow Interoperability Based on XML-Messages", *Proceedings of CAiSE'00 workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00)*, Stockholm, Sweden, June 5, 2000.
11. Object Management Group, "Workflow Management Facility", Document number formal/00-05-02, April 2000.
12. K. Schulz and Z. Milosevic, "Architecting Cross-Organizational B2B Interactions", *Proceedings of the 4th International Enterprise Distributed Object Computing Conference (EDOC 2000)*, pp. 92-101, Los Alamitos, CA, USA, 2000.
13. W. M. P. van der Aalst, "Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow", *Information Systems*, 24(8), pp. 639-671, 1999.
14. Workflow Management Coalition, "Interoperability Wf-XML Binding", Technical report WfMC TC-1023, May 1, 2000.
15. Workflow Management Coalition, "The Workflow Reference Model", Technical report WfMC TC-1003, Jan. 19, 1995.