Supporting Dynamic Change in B2B Coordination

Mark G. Matthews
The MITRE Corporation
McLean, VA, U.S.A.

Abstract

B2B applications require coordination of distributed services according to specific workflow policies. A major challenge with B2B applications is implementing a dynamic change to an active workflow policy. When dynamically changing a policy, active workflow instances that are in progress must be transitioned to the new policy in a dependable manner. To avoid undesirable side effects of the change, many business organizations find it necessary to abort active workflow instances prior to changing a workflow policy. This is an inefficient and ineffective change process. This paper presents the SWAP agent-based architecture that supports dynamic changes in B2B workflow policies. A major feature of SWAP is the ability to preserve safety and correctness properties while transitioning in-progress workflow instances to a new policy without the need to abort in-progress instances.

Keywords

B2B coordination, dynamic architecture, dynamic change, adaptive workflow

1. Introduction

Business-to-Business (B2B) applications involve the coordination of distributed services over the Internet according to specific business policies. Workflow management systems are gaining increasing importance in B2B applications by automating the coordination of the services according to specific business policies, called workflows. The workflow defines a specific ordered execution of the services.

The activity diagram in Figure 1 shows an example of a B2B workflow that involves three distinct business partners (customer, on-line merchant, and supplier). The specified order of execution is as follows: 1) The customer generates an order for a product and submits the order to an on-line merchant. 2) The merchant receives the order and checks the customer's credit. 3) If the credit check passes, the merchant sends the order to the applicable supplier business partner. If the credit check fails, the order is rejected and the customer is notified. 4) If the supplier determines that the ordered product is available within the inventory, the supplier ships the product and the merchant concurrently bills the customer. If the supplier determines that the ordered product is not available, the order is rejected and the customer is notified.

The need to change an implemented workflow-based coordination policy is bound to

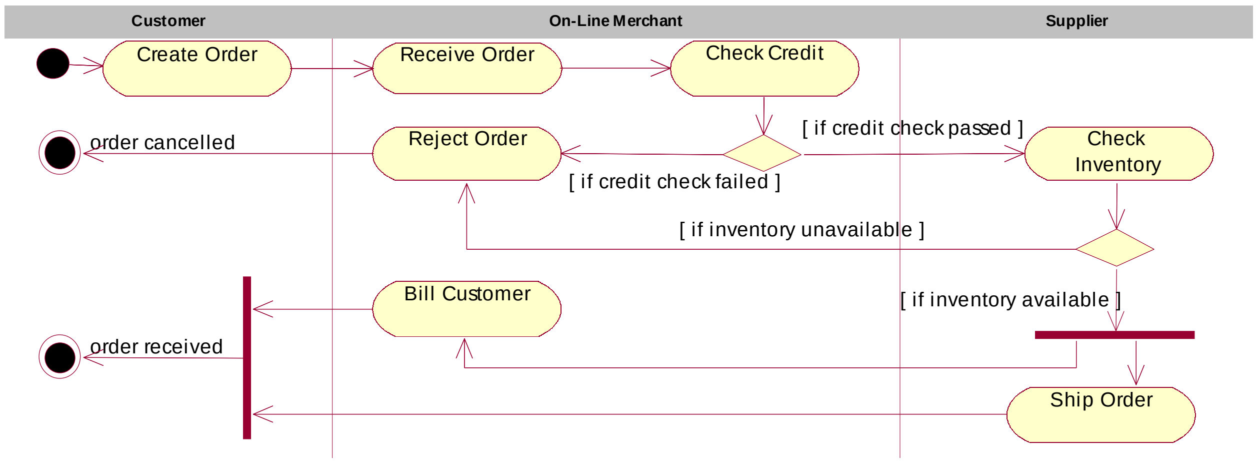

Figure 1: Concurrent Shipping and Billing Workflow Policy

arise at some point in time in order to respond to changing business requirements, changing business policies, or the introduction of new services. Performance and availability issues associated with shared network infrastructure and services may also lead to the need to change the active policy.

When changing a workflow policy in a B2B application a major challenge is handling active workflow instances that are in progress. In practice many business organizations find it necessary to abort active workflow instances prior to changing the workflow policy. Once the new policy is implemented, the aborted instances can then be reinstantiated under the new policy. This is an inefficient and ineffective change process since it forces work to be discarded and redone under a new policy. Automated B2B coordination mechanisms are needed to support dynamic changes in workflow policies in a dependable manner.

This paper considers dynamic structural changes to workflows [5]. In the B2B workflow depicted in Figure 1 the shipping and billing activities are performed concurrently. Consider a scenario in which a change in the workflow is introduced specifying that the shipping activity is to be performed after the completion of the billing activity (Figure 2). Although the coordination may be "correct" before the change and correct for orders processed after the change, there may be problems with orders undergoing processing when the change is introduced. Orders that have undergone the shipping activity but not the billing activity in the traditional workflow (Figure 1) will not undergo the billing activity under the new workflow (Figure 2). The dynamic change in workflow policy has the undesirable side effect of failing to bill some customers. To avoid such undesirable side effects when dynamically changing a workflow policy, in-progress workflow instances are typically aborted and restarted under the new policy. This paper presents an agent-based architecture that supports dynamic change in workflow policies without the need to abort in-progress workflow instances.

Figure 3 presents a scenario in which a change in the traditional workflow is introduced specifying that the inventory and credit checking activities are to be performed concurrently.

Section 2 discusses the approach for supporting dynamic change in workflow policies. Section 3 provides an overview of the SWAP architecture. The architectural approach is illustrated through the development of mechanisms supporting dynamic changes between the workflow policies in Figures 1-3. Related work is included in Section 4 and Section 5 includes a summary and a discussion of future work.

2. Approach to Dynamic Change in B2B Workflow Policies

The major challenge involved in dynamically switching to a new workflow policy is to ensure the preservation of application specific safety and correctness properties for all jobs undergoing processing during the change. For

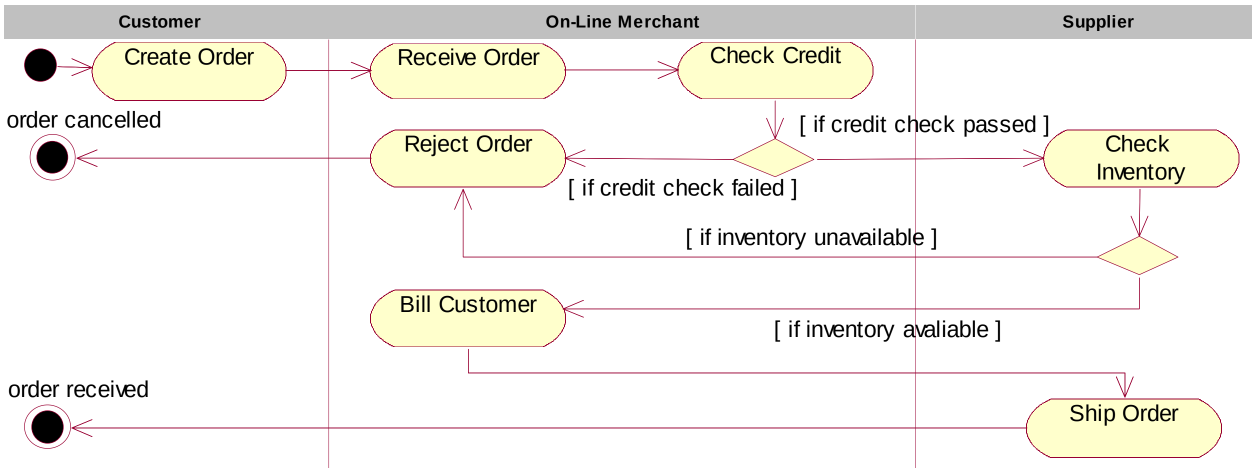

Figure 2: Sequential Shipping and Billing Workflow Policy

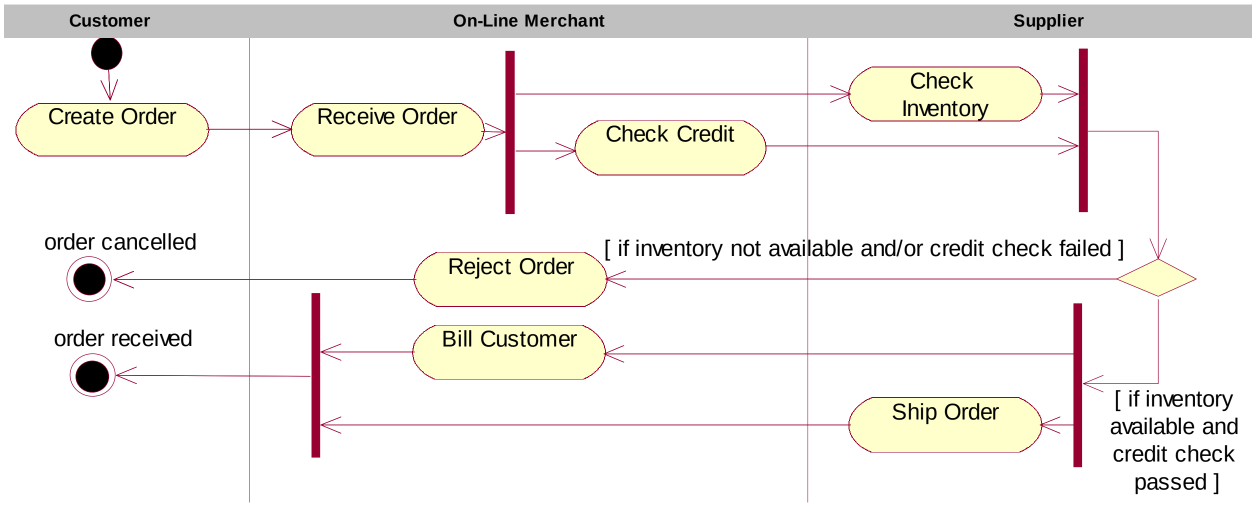

Figure 3: Concurrent Inventory and Credit Checking Workflow Policy

example, in the order processing domain discussed earlier, there are two primary properties (activity completeness and non-redundant activities) that must be preserved during a dynamic policy change.

The *activity completeness property* involves ensuring that all in-progress and future orders successfully complete all required activities according to a valid workflow policy. *Shipping and billing atomicity* is an example of an activity completeness property that involves ensuring that an order undergoes the billing activity if and only if the order undergoes the shipping activity.

The *non-redundant activities property* involves ensuring that in-process orders do not repeat completed activities as a result of transitioning between workflow policies. For example, an order shall not be shipped twice as a result of transitioning from a parallel billing and shipping policy to a serial policy.

The SWAP approach to change coordination is based on an understanding that ensuring the integrity of application specific safety and correctness properties during a dynamic workflow policy change can be accomplished by the following: a) Using knowledge of the state of in-progress workflow instances. b) Exploiting knowledge of the new policy to identify augmentations to workflow instances required to ensure key safety and correctness properties. c) Executing a sequence of actions to bring about the dynamic change in policies and their respective states.

The key idea in the SWAP approach involves defining an agent-based architecture that contains *tracking agents* to monitor workflow policy execution states and a *change coordination agent* that uses the tracked policy states to execute a set of change coordination actions to safely realize the dynamic change.

3. The SWAP Architecture

In this section we present an overview of SWAP, an agent-based architecture supporting dynamic changes in B2B workflow policies. For simplicity, we consider a case in which dynamic changes are supported between the three workflow policy choices depicted in Figures 1-3.

Figure 4 shows a SWAP architecture, modeled as a UML class diagram using stereotypes of the UML class, that supports dynamic changes between three B2B workflow policies. SWAP is a layered agent-based architecture with separate layers for application and change coordination. The architectural components in each layer, the connections between the components, and the connections between the layers are briefly described in the following subsections.

3.1 Application Coordination Layer

The application coordination layer consists of agent representatives of the application specific service components required to perform the workflow tasks specified by the policies described above. The client agent performs activities related to the customer business entity. The credit, billing, receiving, and rejecting agents perform activities related to the on-line merchant business entity. The billing and shipping agents perform activities related to the supplier business entity. The architecture includes a workflow scheduling agent for each supported workflow policy. These agents implement the individual workflow policies that govern the coordination between the customer, merchant, and supplier service agents.

The architecture includes an application coordination agent that accepts orders from the client agent and delegates the ordering job to the active workflow scheduling agent. The application coordination agent also includes several data structures for maintaining various job queues and the status of individual orders. There is a distinct job queue for each activity (shipping, billing, inventory, credit, rejecting, and receiving) within the workflow to indicate the active orders that are being processed by the merchant and supplier agents at any point in time.

The application coordination layer agents interact via the application coordination event-channel.

3.2 Change Coordination Layer

The change coordination layer consists of a tracking agent, a change coordination agent, and a shared coordination data space.

3.2.1 Tracking Agent

The job of the tracking agent is to maintain enough knowledge of the state of in-progress workflow instances to support a dynamic workflow policy change at any point in time. The tracking agent interacts with the application coordination agent via the change coordination event channel. Each time there is a qualitative change in one of the application coordination layer state variables (i.e., key job queues), the application coordination agent propagates the change to the tracking agent. The tracking agent maintains an abstract state vector that abstractly captures the processing state of the application coordination layer agents. Each time the abstract state vector is updated, the vector is written to

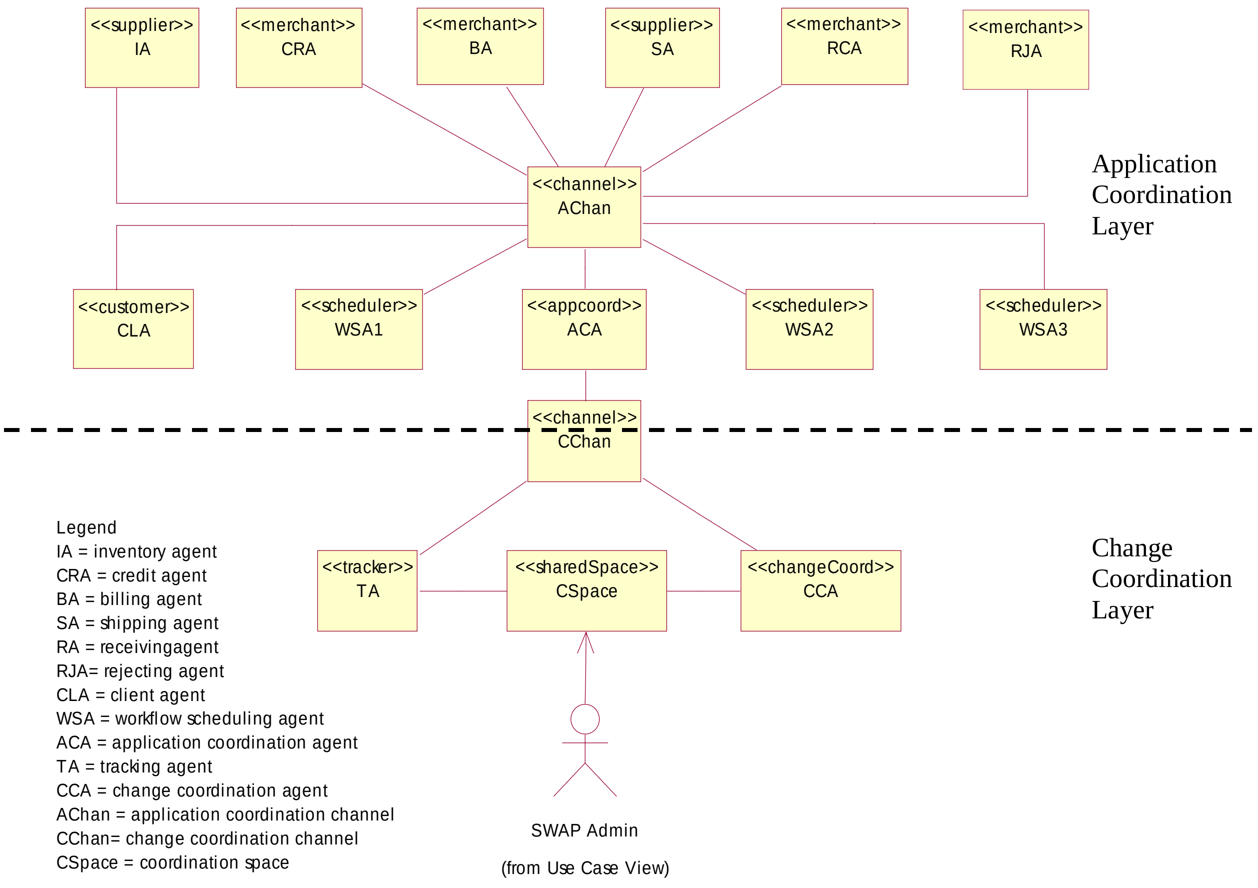

Figure 4: The SWAP Architecture for Dynamic Change in B2B Coordination

the coordination space.

The approach to developing the tracking agent consists of the following steps: 1) Identify the key state variables of the workflow scheduling agents in the application coordination layer. The set of key state variables is referred to as the state vector. This step requires an analysis of the behavioral specifications of the workflow scheduling agents. 2) Identify qualitative abstractions of the state variables. Qualitative abstractions are used to reduce the complexity of the tracking and change coordination agents. 3) Develop a tracking agent behavioral specification consisting of a state model with qualitative abstractions based on the state models of the workflow scheduling agents. Choosing the right abstractions is critical to the approach for formal correctness analysis [6].

For the example considered in this paper, the following abstract state vector adequately tracks the application layer processing state:

$$sv_a = \{AIC, ACC, AS, AB, CE, SE\}$$

In the above equation, AIC, ACC, AS, and AB abstractly capture the inventory check, credit check, shipping, and billing job queues. The credit exceptions (CE) and shipping exceptions (SE) variables capture exceptions identified when switching from a concurrent policy to a sequential policy. CE captures those jobs that have passed a credit check but not an inventory check. Once the inventory check has been passed, these orders will skip the credit check state specified by the sequential policy. Likewise, SE captures those jobs that have completed the shipping activity but not the billing.

An analysis of the above state variables shows that it is not important for the tracking agent to know the individual job ids in each queue, or even the number of jobs in each queue. It is only important for the tracking agent to know if there are any (1 or more) jobs in a state variable queue. Hence, the qualitative abstractions of empty (E) and nonempty (N) are used for all state variables maintained by the tracking agent. This translates to a requirement for the application coordination agent to notify the tracking agent each time a state variable changes from an empty to a nonempty abstract value or vice versa.

3.2.2 Change Coordination Agent

The change coordination agent accepts a policy switching decision from the SWAP workflow administrator via the shared coordination space. The change coordination agent, based on current tracking information maintained in the coordination space by the tracking agent, then executes the set of configuration and control actions required to bring about the dynamic switching between workflow scheduling agents. The change coordination agent interacts with the application coordination agent via the coordination channel.

Scenario-based analysis is used in the approach to developing the change coordination agent. The approach consists of the following steps: 1) Identify a set of workload scenarios that covers the state space of the tracking agent specification. The workload scenarios

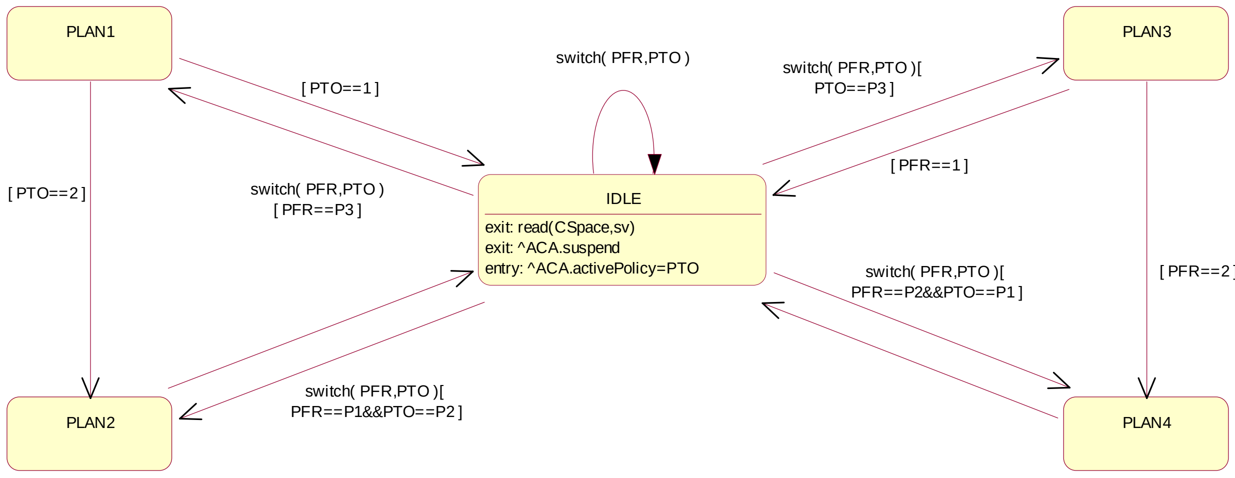
PLAN1
PLAN3
switch(PFR,PTO)
[PTO==1]
switch(PFR,PTO)[PTO==P3]
[PFR==1]
[PTO==2]
switch(PFR,PTO) [PFR==P3]
IDLE
exit: read(CSpace,sv)
exit: ^ACA.suspend
entry: ^ACA.activePolicy=PTO
switch(PFR,PTO)[PFR==P2&&PTO==P1]
[PFR==2]
switch(PFR,PTO)[PFR==P1&&PTO==P2]
PLAN2
PLAN4

Figure 5: Change Coordination Agent Specification

correspond to jobs in different stages of processing. 2) Generate an event trace for each workflow policy against each workload scenario. An event trace captures the changes in the job-queue state variables as jobs undergo execution based on the workflow policy. 3) Systematically analyze the event traces for pairwise switching of policies such that certain integrity constraints (for example customers get billed if and only if the product gets shipped) are preserved. This involves identifying the workload constraints of the active policy in a given state (specified by the abstract state vector) in the trace and the matching workload constraints of the new policy to be activated in a specific state. 4) Generate the change plans. This involves combining the results obtained from the set of pairwise analyses performed in step 3 and resolving any discrepancies between analyses. The configuration actions (change plan) for activating the new policy then consist of setting up the workload state variables of the workflow scheduler agents according to the valid transitions obtained from the above analysis. 5) Model the change coordination agent as a decision tree captured by a finite state machine model. The decision conditions are captured by the transitions and the actions (change plans) are captured by the activity states.

The change coordination agent specification for the example presented in this paper is depicted in Figure 5. This specification shows the actions performed upon receipt of a change decision from the SWAP administrator. For example, upon receipt of a decision to dynamically switch from policy 3 to policy 2, the change coordination agent performs the following activities. 1) Read the abstract state vector from the coordination space. 2) Send a command to the application coordination agent to suspend operations. 3) Execute plan 1 which manipulates the job queues in the application coordination agent to serialize the inventory and credit checking activities. Change plan details have not been included due to space limitations. 4) Execute plan 2 which manipulates the job queues in the application coordination agent to serialize the billing and shipping activities. 5) Send a command to the application coordination agent to set the activity workflow policy to policy 2.

After execution of the sequence of actions identified above, the application coordination agent will forward all new orders to workflow scheduling agent 2. In-progress workflow instances will continue execution under workflow policy 2. Those in-progress instances identified as credit or shipping exceptions will skip those activities under workflow policy 2.

4. Related Work

The SWAP approach described in this paper builds on existing research on workflow management systems and software architectures to define a domain specific architecture for dynamic change in workflow-based B2B coordination.

There has been a significant amount of work conducted in the area of workflow models and workflow management systems. Workflow studies have been conducted by researchers in the area of organizational design, office information systems and software engineering [11,12,13]. Research in automation of workflow has resulted in several systems [4] that exploit existing distributed computing technologies. In most of the systems, adapting to dynamic changes in a dependable manner is not well-addressed [7]. More recent work [5, 6, 7] has started to address such dynamic change concerns. SWAP develops an integrated architecture for change management and develops a systematic method for the design and analysis of the change coordination mechanisms in the architecture to ensure safety properties during dynamic switching between policies.

The SWAP work on change coordination is related to the area of dynamic software architectures [1, 8] and on using architectural specifications to plan and analyze changes in the run-time system [9,10]. The change coordination layer in SWAP exploits domain specific knowledge to identify the necessary components and their role in the change process.

5. Summary and Future Work

This paper addresses the problem of dynamic change in the context of a B2B coordination of services. This paper presents an overview of the SWAP architecture, an agent-based architecture that supports dynamic changes in B2B workflow policies. An instance of the SWAP architecture supporting dynamic changes between three workflow policy choices is presented. The design and analysis of the change coordination mechanisms within the architecture is discussed.

The above approach has also been applied to other domains [2,3]. Future work is focused on generalizing the architecture and methods based on results obtained in applying SWAP to multiple task domains.

References

[1] R. J. Allen, R. Douence, D. Garlan. Specifying and Analyzing Dynamic Software Architectures, Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE '98), March 1998.

[2] P. Bose and M. G. Matthews, "Coordination of View Maintenance Policy Adaptation Decisions: A Negotiation-Based Reasoning Approach", Proceedings of the International Workshop on Self-Adaptive Software, Oxford, England, April 2000.

[3] P. Bose and M. G. Matthews, "NAVCo: Negotiation-based Adaptive View Coordination", in Proceedings of the Automated Software Engineering Conference, 1999.

[4] A. Dodac et. al., "Workflow Management Systems and Interoperability", NATO ASI Series, Springer Verlag, Berlin 1998.

[5] C. Ellis, K. Keddara, G. Rozenberg, "Dynamic Change within Workflow Systems", Proceeding of the ACM Conference on Organizational Computing Systems, 1995, pp 10-21.

[6] M. Kamath and K. Ramamritham, "Correctness Issues in Workflow Management", Distributed Systems Engineering Journal, Special Issue on Workflow Management, Volume 3, Number 4, December 1996.

[7] M. Klein, C. Dellacros, and A. Bernstein, eds., "Workshop Towards Adaptive Workflow Systems" CSCW-98 Workshop Proceedings, ACM Press, 1998.

[8] J. Magee, J. Kramer. Dynamic Structure in Software Architectures. Fourth SIGSOFT Symposium on the Foundations of Software Engineering, San Francisco, October 1996.

[9] P. Oreizy, N. Medvidovic, R. N. Taylor, "Architecture-based Runtime Evolution", ICSE 1998.

[10] P. Oreizy et. al., "An Architecture Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, 1999.

[11] L. Osterweil, “Automated Support for the Enactment of Rigorously Described Software Processes,” Proceedings of the Third International Process Programming Workshop, Computer Society Press, 1988, pp 122-125.

[12] A. Seth, editor, Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems.

[13] L. A. Suchman, "Office Procedure as Practical Action: Models of Work and System Design", ACM Transactions on Office Information Systems, vol. 1, no. 4, October 1983, pp 320-328.