2000

A Development Approach for Workflow-Based E-Commerce using Reusable Distributed Components

M. Brian Blake
George Mason University, mblake@gmu.edu

A Development Approach for Workflow-Based E-Commerce using Reusable Distributed Components

M. Brian Blake, Department of Information and Software Engineering
George Mason University, mblake@gmu.edu

Abstract

With the emergence of component-based technologies, there is a push toward software reuse. Reusable software components, such as Java Beans, ActiveX components, COM objects, registry services, etc., can be used to support a wide variety of distributed services. The reuse of such components has reduced the need for full lifecycle development and deployment, both in research and industry. Such "top-down" approaches are becoming impractical, while there is the increasing trend toward integrating reusable components from bottom up. Furthermore, the use of technologies like introspection and reflection has relinquished the design-time constraint of having source code in order to integrate such components. The implications of the aforementioned technologies suggest that future development processes will reside on the end-user's workstation. Consequently, end-users will produce their own systems as a product of configuring multiple 3rd party components (i.e. software "plug-and-play" (Bronsard, et al., 1997)). With this in mind, the remaining problem is how to automate this configuration to respect some user-determined policy of interaction. One such policy is the implementation of workflow. This paper describes a specification-driven approach, WARP, to configuring components according to workflow. This approach entails a process and an agent-based architecture and implementation that supports the "bottom-up" configuration and reconfiguration of reusable component-based distributed services.

Introduction

Workflow Automation through Agent-based Reflective Processes (WARP) is a new approach to the configuration of reusable components in the context of workflow. This semi-automated approach provides information to the user (workflow designer) about reflective 3rd party components through the process of introspection. A reflective language is one that has a base language and a meta-language describing that base language. Therefore in the process of introspection, a designer can learn about a component without having the actual source code. Reflection is the process of using introspected information to invoke services on the component. This gives the WARP architecture flexibility in managing a workflow of components from only their executables. After providing component information, the WARP architecture elicits workflow interaction information from the user in the form of object-oriented models. The WARP approach defines a set of object-oriented representations in the Unified Modeling Language. Upon completion of these representations, the management of the workflow is achieved through the coordination of multiple software agents.

Agents in industry and research have seen increased exposure over the past several years. Agents have many definitions and functions. WARP agents can be defined as event-based software entities that have perception of their environment. Specifically in this domain, agents act as brokers or proxies (as is middle agents (Decker et al., 1998)) for component-based services. The WARP approach uses Configuration Layer agents that interpret workflow specifications and capture them in a shared repository. The Configuration Layer agents then deploy the self-configurable Application Coordination Layer agents to manage the workflow execution.

The WARP approach is more evolutionary than revolutionary. Software has evolved to the design and development of modular components. The next logical step is to set standards for software components as in JavaBeans and ActiveX. The WARP approach is toward the creation of a framework that will facilitate the coordination of these components in the context of workflow. This paper proceeds in the next section with the workflow terminology used in this work. The on-line stock ordering process is detailed in context of this terminology. Next, there is an overview of the semi-automated WARP process and architecture. The following section describes the workflow representations that will be captured in the database-centered workflow ontology. Consequently, the operational environment is discussed in detail. Finally, there is a summary in the context of related work.

Workflow Terminology

The workflow language here follows workflow terminology used presently by researchers (Lei and Singh, 1997). In order to set the nomenclature for further discussion, the following set of definitions are adhered to throughout this paper.

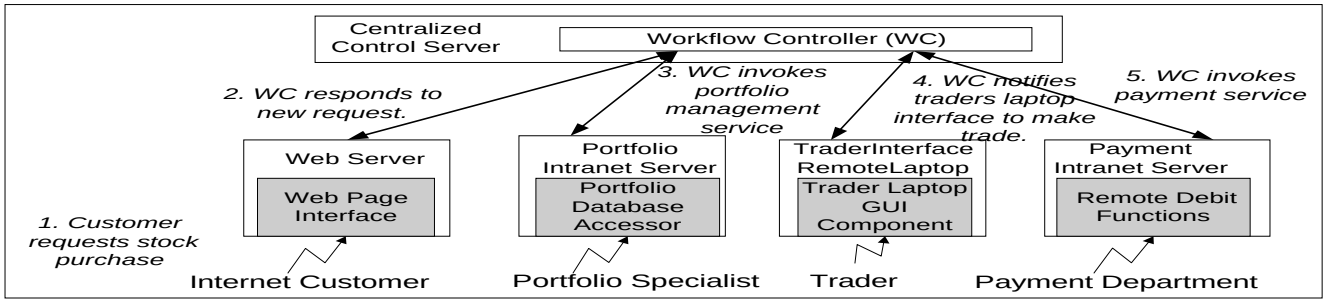
- A *task* is the atomic work item that is a part of a process.
- A task can be implemented with a *service.* (In complex cases, it may take multiple services to fulfill a one task)

- An *actor* or resource is a person or machine that performs a task by fulfilling a service.
- A *role* abstracts a set of tasks into a logical grouping of activities.
- A *process* is a customer-defined business process represented as a list of tasks.
- A *workflow model* depicts a group of processes with interdependence.
- A *workflow* (instance) is a process that is bound to particular resources that fulfill the process.

Example Workflow

A typical distributed workflow automates the on-line stock process (as in E-trade and Waterhouse Securities). The company is split into multiple divisions that handle different aspects of the stock brokering process. Taking a stepwise view of the stock purchase process, the first task occurs when a customer requests to buy shares of stock. Once the request is submitted, an portfolio management department gathers information of the trade. The trade division would purchase the stocks. Finally, a payment division would then debit the customer's account.

Figure 2.1 Typical Distributed WFMS: On-line Stock Purchase



The aforementioned process can be implemented with a workflow that spans a network of distributed servers (Figure 2.1). Each division plays a role containing components that either automate the service or push requests to human-based services. Therefore, a centralized server and controller component can coordinate the execution of the workflow. In this case, the process definition is encapsulated in a central workflow controller (WC).

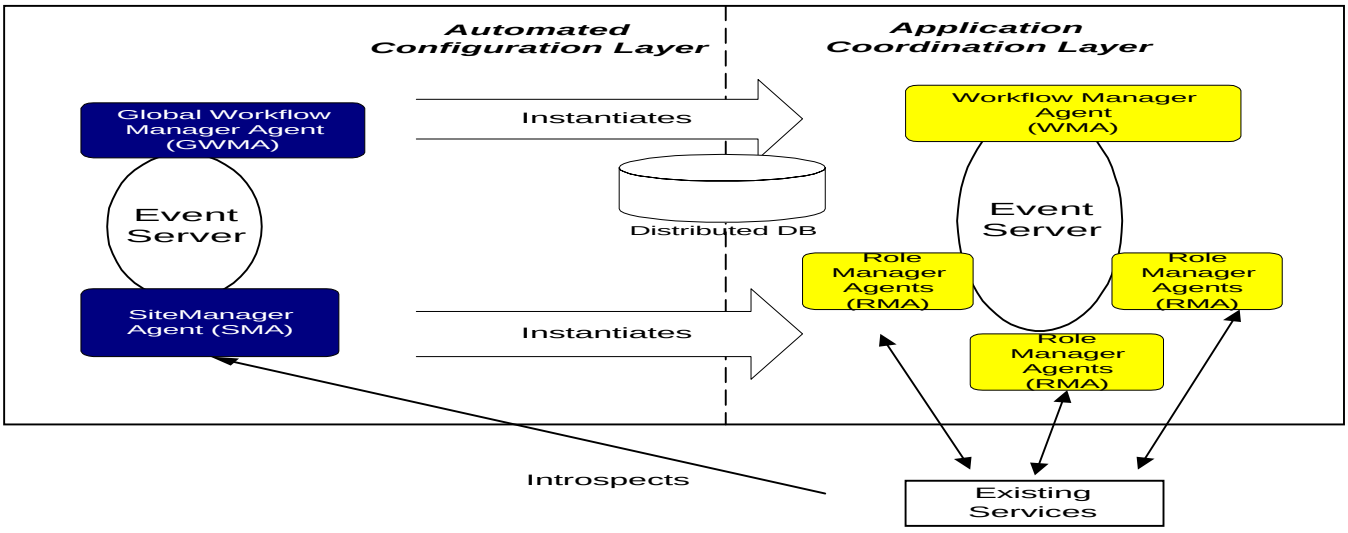
Warp Overview

This overview of WARP is discussed in two parts, the architecture and the configuration process. In this section, the WARP architecture is detailed briefly in context of the configuration process by which the architecture operates.

WARP Architecture and Configuration Process

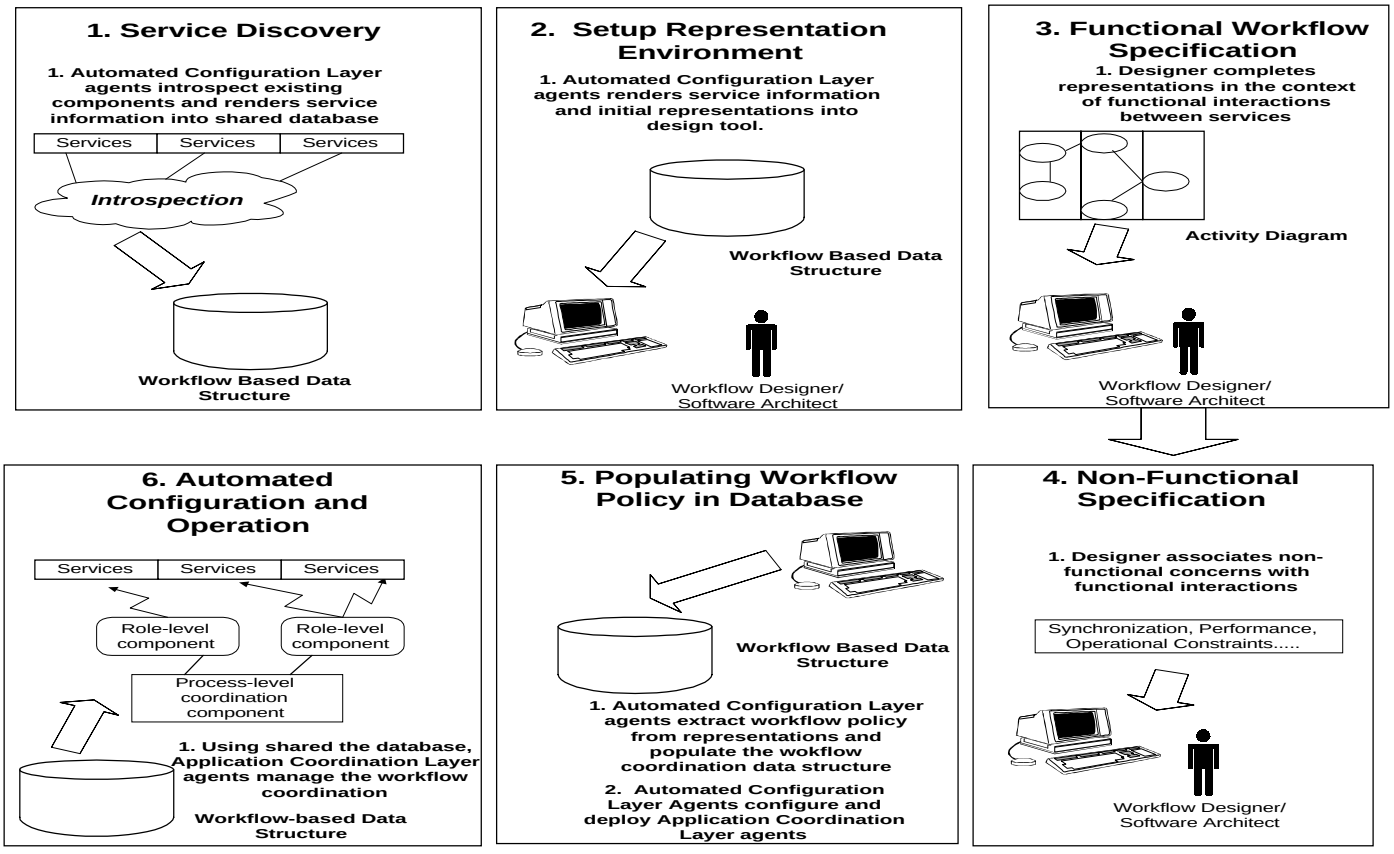
This WARP architecture consists of software agents that can be configured to control the workflow operation of distributed services. The WARP architecture is divided into two layers. These layers are the application coordination layer and the automated configuration layer. The application coordination layer is the level in which the workflow instances are instantiated and the actual workflow execution occurs. The application coordination layer consists of two agents, the Role Manager Agent (RMA) and the Workflow Manager Agent (WMA). The RMAs have knowledge of a specific workflow role. The WMA has knowledge of the workflow policy and applicable roles. When a new process is configured the workflow policy is saved in a shared distributed database. The RMA plays a role in the workflow execution by fulfilling one or multiple services as defined by the workflow policy in the shared database. The RMA registers for pertinent events in the event server based on its predefined role. When an initiation event is written into the event server, the RMA is notified. Subsequently based on its localized knowledge of services and its workflow role, the RMA invokes the correct service. The WMA has similar functionality, but instead registers for workflow level events. The WMA does not control the workflow execution, but in some cases it adds events to bring about nonfunctional changes to the execution of the entire workflow. The WARP architecture is shown in Figure 3.1. At the automated configuration layer, agents accept new process specifications and deploy application coordination layer agents with the new corresponding policy. This layer consists of the Site Manager Agents (SMA) and the Global Workflow Manager Agent (GWMA). The GWMA accepts workflow representations from a workflow designer as input. The SMAs introspect available services and provides service representations to the GWMAs by way of the shared distributed database. The GWMAs accepts both of these inputs and writes the workflow policy into the shared database. The GWMA then instantiates the WMAs to play certain workflow-level nonfunctional roles. At completion of workflow-level configuration, the SMA instantiates the RMAs to play each of the roles specified

Figure 3.1 WARP Architecture



in the shared workflow-based database. As a summary to the components/processes of the Automated Configuration Layer, the WARP process can be split into 6 steps. The WARP configuration process is essentially a workflow process where the agents and the workflow designer are the workflow roles. These configuration steps are illustrated in Figure 3.2.

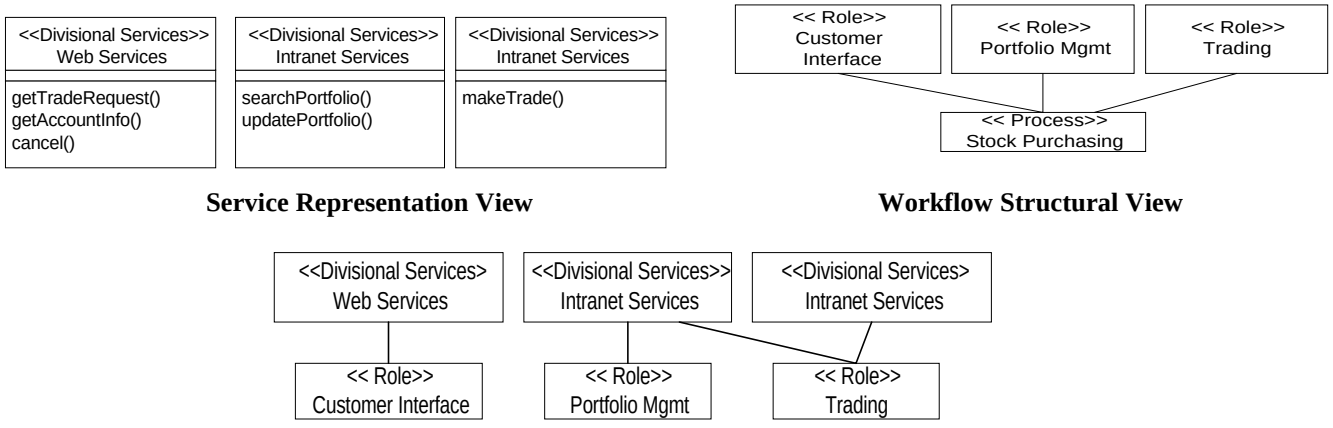
Figure 3.2 WARP Configuration Process



WARP Representations

There is a great deal of investigation in the use of the UML for business process modeling. This paper presents a systematic approach to this modeling that is unique. Our approach separates functional and non-functional concerns, explicitly. This separation of concerns is in the same spirit as research in aspect-oriented programming (AOP) (Kiczales et al., 1997). This separation allows the functional and non-functional configuration to occur independently. The representations are separated into three views, structural, functional, and non-functional views. The structural views show specific information for the component-based services (Service Representation View), definition of the roles (Role Aggregation View), and the composition of the workflow or process (Workflow Structural View). The structural views are represented in UML class diagrams as in Figure 4.1.

Figure 4.1. On-line Stock Purchasing Process: Structural Views



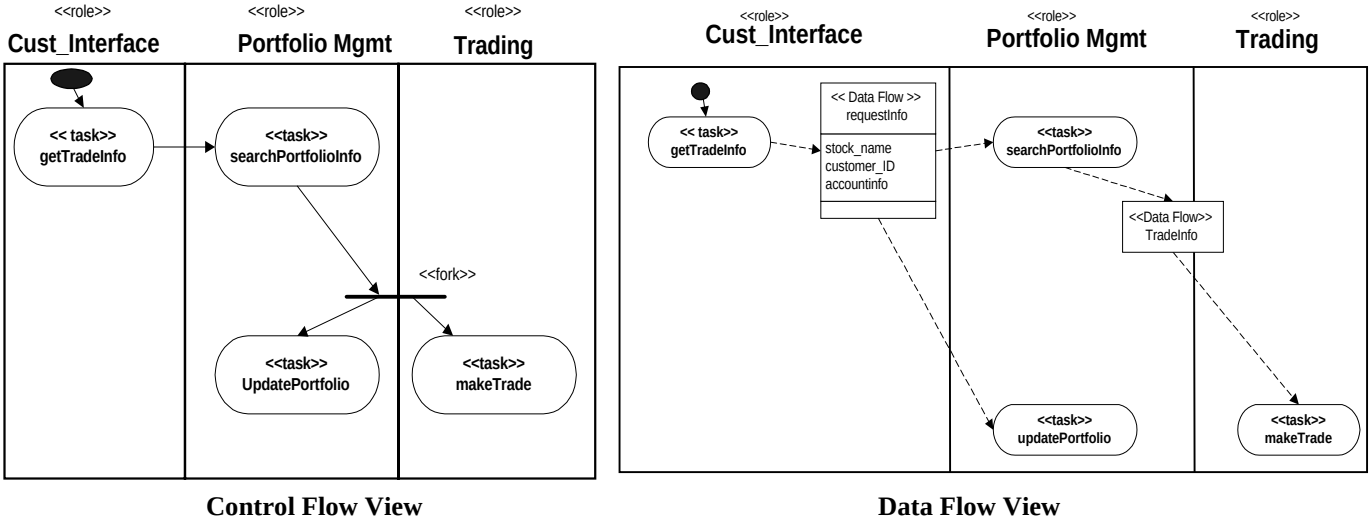
Service Representation View

Workflow Structural View

Role Aggregation View

The Service Representation View shows the actual component-based services available for reconfiguration. The Role Aggregation View and Workflow Structural View use association relationships to show the composition of the role and process, respectively. The functional views show the data and control flow of the workflow. The use of activity diagrams is becoming the industry de facto for showing workflow representations. Our representations use separate activity diagrams to show control and data flow, the Control Flow View and the Data Flow View. Figure 4.2 shows the control and data flow views that detail the on-line stock purchasing process illustrated in Figure 2.1.

Figure 4.2 On-line Stock Purchase Process: Functional Views



Control Flow View

Data Flow View

Typical nonfunctional concerns are those of synchronization, concurrency, atomicity, etc. These concerns are generally imposed on the system or the process as a whole. We have done research into representing these concerns into one generic view. However, formalization of the nonfunctional view is forthcoming.

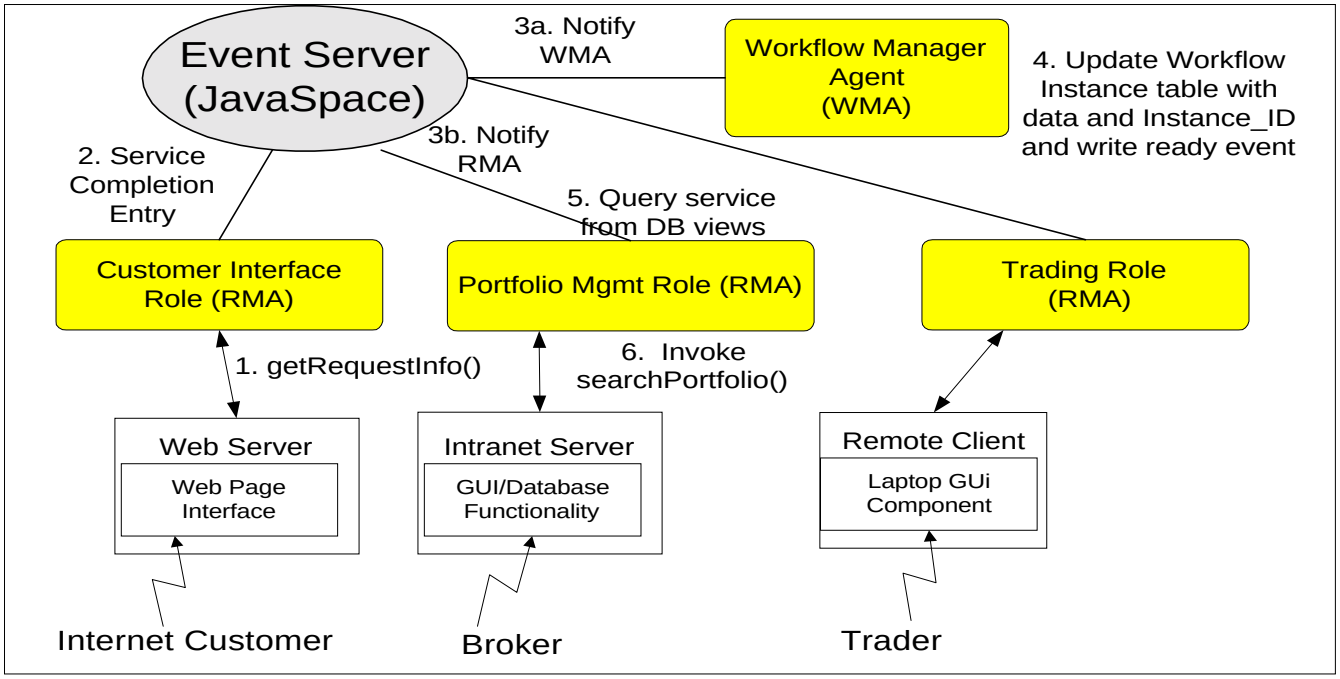
Workflow-Oriented Database

The WARP architecture relies heavily on a shared database to determine the operations of the workflow. The representations in the previous section are stored in a shared repository. These models are translated into a shared relational database structure. Agents access this relational database to control the configuration, reconfiguration, and operation of the workflow. Considering the space for this paper, no further database detailed is covered in this paper (Blake, 2000).

Operational Details

To consider the operational environment, again let's use the on-line stock-purchasing domain. A configured system would have a RMA for each of the roles. Let's consider the three aforementioned roles are the Customer Interface Role, the Portfolio Management Role, and Trading Role. There is a RMA for each role and there is one WMA that helps in the coordination of the entire workflow. Based on the Control Flow View, each role would place notify commands in the space for service completion prior to their affiliated services. For example, the Portfolio Management Role would want a notification on the completion event of a getTradeRequest service. Suppose a customer invokes the getTradeRequest service. The RMA for the Customer Interface Role would insert the pertinent data for this service completion in the database and publish the service completion in the event server. The RMA for the Portfolio Management Role would be notified of this completion. First it would check to see if this service is pertinent to any of its workflow instances. If the answer is yes, the RMA for the Portfolio Management Role would wait for the ready event to be written to the server by the WMA. The WMA would have also been notified of the getTradeRequest service completion. The WMA would update the Workflow instance table with the new workflow process' instance ID. Finally, the WMA would submit a ready event to the event server upon completion. The RMA would invoke the proper service for this step in the workflow policy based on the action specified in its role-based view (searchPortfolio service). Subsequently the output data would be inserted in the database and the service completion would be written to event service. This process sequence is shown in Figure 5.1 for case of the stock purchase process excluding any error-handling.

Figure 5.1 Operational Environment.



Summary

This work describes a process and agent-based architecture that supports the configuration of distributed components in the context of workflow. This approach is implemented with Java-based agents and uses a JavaSpace event server. The design tool for capturing workflow representations is the Rational Rose Developer Enterprise 98 edition using the Rose Extensibility Interface (REI). This initial architecture has the ability to configure JavaBean components based on a user-defined workflow captured in Rose.

Related Work

The WorkWeb system (Tarumi, 1997) handles the coordination among various threads in a predefined workflow. The workflow in the WorkWeb system is static and the services are fixed. The WARP approach encapsulates the coordination of multiple threads in the workflow within the operation of the WARP agent architecture. Moreover, the WARP approach considers a larger set of services that are dynamic in nature.

There is other work in dynamic reconfiguration and run-time evolution (Shrivastava and Wheater, 1998). This work details a system for evolving workflow systems based on workflow scripts and task models. Using a CORBA-based support architecture, they accept workflow scripts as specifications for reconfiguration. This approach relies on a static set of workflow components. Shrivastav devises a system that takes a "top-down" approach. The system uses a set of CORBA-based services that can be dynamic in nature. Shrivastav devises a workflow script that specifies the functional nature of the workflow where nonfunctional constraints are bundled in. The WARP approach separates functional and non-functional specifications. This separation allows for a greater ease of reuse for nonfunctional considerations. The WARP approach also distributes the workflow policy among the underlying services thus decentralizing the control. This maintains the autonomy among services for the support of the addition of new services and the removal of deprecated services.

Future Work

At this point in our work, we are developing the WARP architecture and evaluating it on multiple e-commerce domains. Future work in the short term is toward the formalization of nonfunctional concerns. This formalization will be leveraged on the operation of the architecture. In addition, we are implementing the WARP architecture to respect other domains in supply chain management.

Acknowledgements

I would like to recognize Dr. Prasanta Bose of the Department of Information and Software Engineering for his motivation and direction in the development of this approach. His sound advice was instrumental in the development of the architecture and the agent-based collaboration.

References

Blake, M.B. and Bose, P. "An Agent-based Approach to Alleviating Packaging Mismatch", *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS2000),* Barcelona, Spain June 2000

Booch, G., Rumbaugh, J., Jacobsen, I., "The Unified Modeling Language User Guide", Addison Wesley, Reading MA, 1998

Blake, M.B., "WARP: An Agent-Based Process and Architecture for Workflow-Oriented Distributed Component Configuration", *Proceedings at the 2000 International Conference on Artificial Intelligence (IC'AI2000),* Las Vegas, NV, June 2000

Bronsard, F, et al., "Toward Software Plug-and-Play" *Proceedings of the 1997 Symposium on software reusability*, 1997 Pages 19-29

Decker, K., Sycara, K., and Williamson, M., "Middle-Agents for the Internet" ,*In the Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan,, August 1997*

Kamath, M. and Ramamrithan, K., "Correctness Issues in Workflow Management". Distributed Systems Engineering Journal-Special Issue on Workflow Systems, 3(4): 213-221, December 1996.

Kiczales, G., Lamping, J., Mandhekar, A., Maeda, C., Lopes, C., Loingtier, J. and Irwin, J., "Aspect-Oriented Programming." Xerox Parc Technical Report spl-97-008, 1997.

Lei, K. and Singh, M.. A Comparison of Workflow Metamodels, *Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling*, Los Angeles, November 1997.

Shrivastava, S. and Wheater, S., "Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications" The 4th International Conference on Configurable Distributed Systems (CDS'98), Annapolis, Maryland, USA, May 4-6 1998.

Tarumi, H., "WorkWeb System -- Distributed Multi-Workflow Control with a Multi-Agent System", *IPSJ International Symposium on Information Systems and Technologies for Network Society*, World Scientific, Sep 1997