

Agent-Based Workflow Configuration and Management of On-line Services

M. Brian Blake
Department of Computer Science
Georgetown University
234 Reiss Science Building
Washington, DC 20057
blakeb@cs.georgetown.edu

Abstract. *With the emergence of distributed component-based technologies, information services have become extremely modular and autonomous. This autonomy has allowed organizations within one enterprise to develop component-based solutions that are independent and perform tasks relevant to their domain-specific functionality. However, enterprise-level operations typically require the workflow composition of these independent services to accomplish corporate-wide goals. In traditional development environments, enterprises have taken a “top-down” approach where the workflow was conceptualized and supporting low-level services have been created. This solution neglects the robustness and reuse associated with current distributed component-based technologies. In this work, there is a “bottom-up” approach where existing component-based services can be composed to conform to a user-specified workflow. This paper presents an agent architecture that acts as an adaptive middleware-layer between existing components. This architecture accepts workflow specifications in the Unified Modeling Language as input to this middleware-layer. Furthermore, we detail the implementation and results of this architecture that configures and manages a set of Java-based components.*

1.0 Introduction

It is common in industry that enterprise information systems support business processes or workflow. The information-oriented systems that support

workflow are called Workflow Management Systems (WFMS). The typical WFMS is static, in nature. Generally, business processes are conceptualized in advance and are “hard-coded” into these systems. With the emergence of networking and distributed computing, services are becoming autonomous and distributed across multiple sites.

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, a 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.

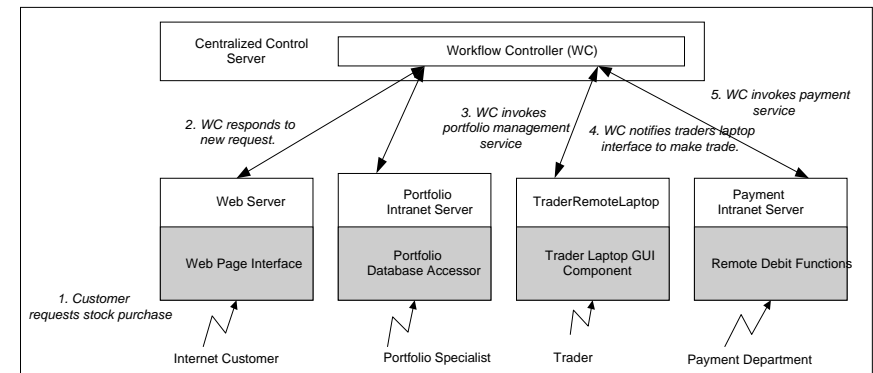


Figure 1. Distributed Component Workflow for On-line Trading.

A problem with this typical WFMS is how to change the system when the business needs changes. For example, if a new business need in the ordering domain is to handle returns, the addition of an on-line product return workflow would be required. An entirely new set of processes may need to be added. In the traditional WFMS, this would result in an effort to re-design, develop, and deploy the system. This traditional “top-down” developmental approach to workflow design and deployment would be highly ineffective considering the

cost and time required to bring about the change. More incremental and bottom-up compositional approaches are required to reduce the cost. Another major problem with the top down approach is the handling of non-functional constraints such as synchronization, error recovery, atomicity, security and etc. Design for such constraints are not done explicitly but rather folded directly into the code. Again, top-down changes to such constraints are very difficult to handle.

This paper will proceed in the following section with a background of pertinent technologies. Subsequent sections will present the approach and technical details to workflow configuration. Finally, there will be details on the prototype used to simulate the architecture for the approach.

2.0 Technical Background

There are several technologies and assumptions that are made specifically in this research that are stipulated in this section.

2.1 Workflow Terminology

The workflow language here follows workflow terminology used presently by researchers (as in Lei and Singh [33]). 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.

2.2 Agents for the Distributed Workflow Management Domain

Software agent technology has seen increased attention in professional and academic research labs with the approach of the 21st century. A software agent can be defined as an autonomous software module that acts independently and adaptively to assist an end user in performing a domain-related task [17][26][28][32][48][54][56]. Agent technology has been extensively studied in environments where dynamic information is shared across systems, where intelligence is indigenous to the software modules, and where software modules are expected to adapt to changing configurations. Common implementations of software agents are in the development of applications that follow some workflow and application frameworks that allow domain developers to concentrate on the domain specific problems. In multiple agent systems, groups of software agents communicate and negotiate shared information and responsibilities to process a collective task.

Agents have many definitions and functions. Agents in the context of this research 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) for component-based services. In this domain, there are two types of agents, configuration-layer agents and application coordination-layer agents. Configuration-layer agents 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.

2.3 Reflective Architectures

This work makes use of a reflective architecture to discover the operational semantics of existing services [16] [18][44][39]. A reflective language can be defined as a programming language that has both a base (integration) language as well as a meta-language that describes the base language. Reflective languages allow developers to determine the characteristics of compiled software. The act of determining these characteristics at run-time is called *introspection*. After introspection, the developer can invoke functions on the software without re-developing new modules. The act of dynamically invoking functions from previously deployed component is called *reflection*. Introspection and reflection are powerful tools, because they assist in the run-

time evolution of components. Developers can create software components that introspect on other components thus allowing them to evolve as the component functionality is changed.

3.0 Overview of the WARP Approach

The approach to automated compositional configuration is called Workflow Automation through Agent-Based Reflective Processes (WARP). This approach is based on the use of an agent-based middleware architecture here after called the WARP Architecture. 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 centralized database. The RMA plays a role in the workflow execution by fulfilling one or more services as defined by the workflow policy in the centralized database. The RMA registers for workflow step-level 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 overall workflow level events (i.e. workflow initiation and nonfunctional concerns). The WMA does not control the workflow execution, but in some cases it adds events to bring about non-functional changes to the execution of the entire workflow.

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 discover available services and provides service representations to the GWMA.

The GWMA accept both of these inputs and writes the workflow policy to the centralized database.

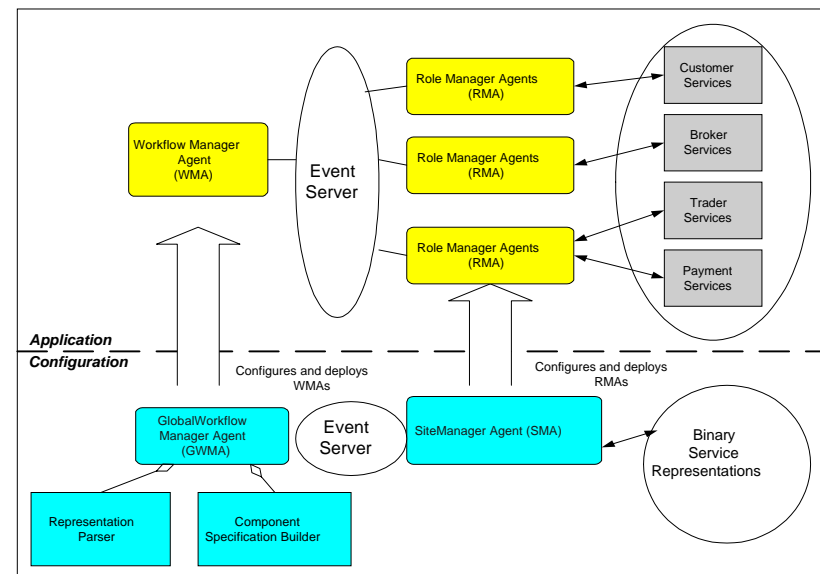


Figure 2 The WARP Architecture

The GWMA then configures and deploys WMAs to play certain aspect-oriented roles. At the completion of workflow-level configuration, the SMA configures and deploys RMAs to play each of the roles specified in the workflow database. A general view of the WARP process is shown in Figure 2.

In the context of a bottom-up compositional configuration, the WARP process is direct. The system takes workflow and service-based information and coordinates the operation of a workflow among previously deployed and new components. Any approach for compositional configuration has to be flexible enough to accept iterative programming. WARP architecture allows this iterative configuration with the input of workflow and service-based specifications. In bottom-up compositional configuration, there also must be mechanisms that can evolve as the services evolve. This evolution of concrete services drives the evolution of any support architecture. The WARP

architecture has reflective processes that allow it to discover new service characteristics as system developers integrate new components. Finally, it is important for the support architecture to follow a process that can accurately and consistently coordinate new workflow specifications. The main focus of this research is to define the components and formalize the method to achieve this semi-automated configuration. The operational view of the WARP Architecture is illustrated in Figure 3.

4.0 A Software Engineering Process for WARP Configuration

Systems configuration using the WARP architecture essentially presents a new software engineering process for development. The WARP configuration process is a semi-automated process that requires the coordination of a human (workflow designer) and multiple software agents. This configuration process is split into two main parts, specification of the component-based services and specification of the workflow processes. The configuration steps of each these main parts and the responsible roles are detailed in Table 4.1 and 4.2.

Action	Who is involved?
1. Collection of component-based services from component executables	Site Manager Agent
2. Population of the service information into the workflow-oriented database	Site Manager Agent

Table 4.1. Service Specification Steps.

Action	Who is involved?
3. Creating initial Rose Environment based on service information	Global Workflow Manager Agent
4. User-driven workflow specification	Workflow Designer (Human)
5. Generating XML from WARP Representations	Global Workflow Manager Agent
6. Populating Workflow-oriented Database	Global Workflow Manager Agent

Table 4.2. Workflow Specification Steps.

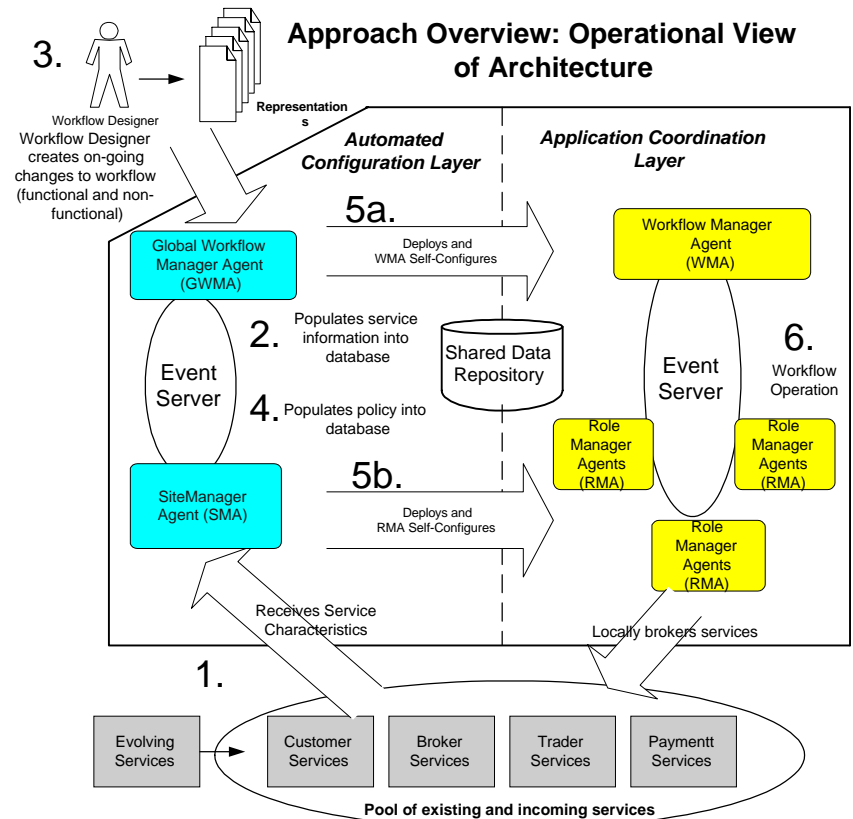


Figure 3 Overview: WARP Process

In Figure 2, step 1 shows how the SMA discovers service characteristics from the existing services. In step 2, the SMA populates this information into a workflow-based repository. In step 3 and 4, the GWMA gathers information from a Workflow Designer (human) and populates this information into the same workflow-based repository. In steps 5a and 5b, the application-layer agents are deployed. Step 6 consists of the real-time management of the workflow operations.

The aforementioned six WARP configuration steps represent an information-centric approach. Information is gathered and represented in many forms (i.e. UML Diagrams, XML Forms, Database Schemas). However, all information is eventually persisted in a database repository. The following section explains the database schema used as the workflow repository for the WARP agents. The subsequent sections explain in detail each of the six configuration steps.

4.1 WARP Workflow-oriented Database Schema

The WARP workflow-oriented database is a relational definition of the workflow terminology described in Section 2.1. The Component, Service, Return, and Parameter tables are used to capture the atomic component-based services. In this case, there is a concentration on invocation-based services, however this approach is applicable to event-based component services. The main table for the workflow is the Workflow Policy table. Each record in this table defines a single process transition. A transition can be defined by the control flow between the completion of a service or group of services as a precondition of the initiation of a subsequent service or group of services. These services are grouped in the EventGrouping table. There is also a Role table that defines a role based on a group of services. The final three functional tables are the DataFlow, Event_Table, and Workflow_Instance tables. The ExecutionAtomicity and FailureAtomicity tables are used to capture the nonfunctional concerns of exception-handling, atomicity, performance, and security. All tables represent the long-term storage for the run-time operation of the workflow. The entire database schema is shown in Figure 4.1.

4.2 Collecting Component-based Service Information

The WARP approach anticipates that on-line services will already be designed developed and deployed. The WARP architecture has Site Manager agents that gather invocation-based information from the on-line services directly from their binary code or bytecode (i.e. introspection). Invocation-based services follow a “call-and-return” style. Each service is represented by a method that takes some information as parameters and passes information back as returns. In the next section, methods will be mapped to the database schema.

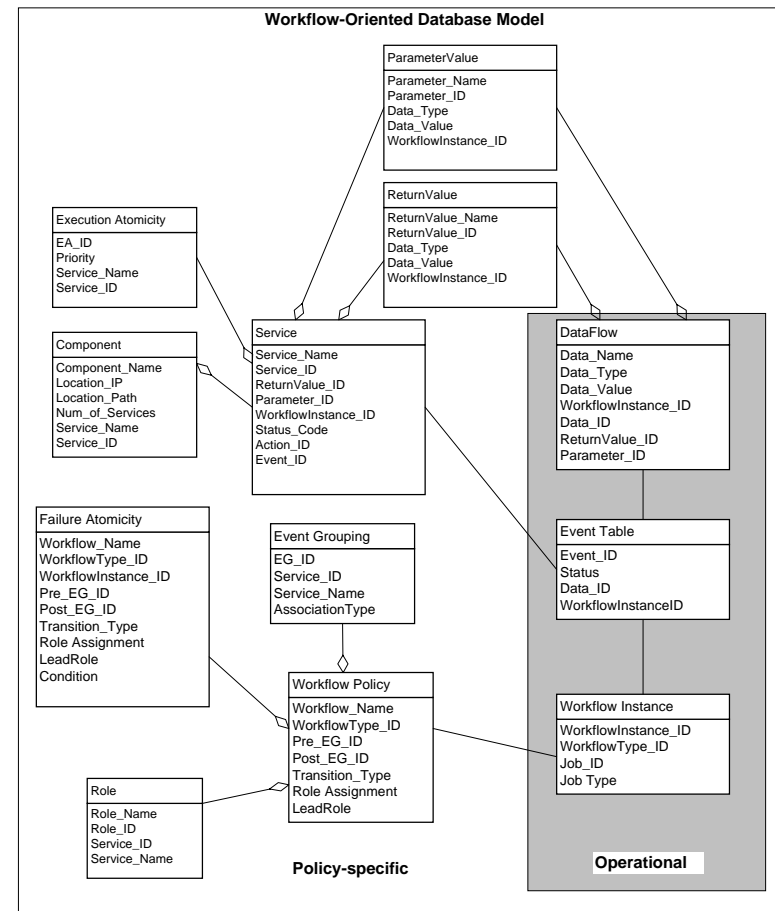


Figure 4.1 WARP Workflow-Oriented Database Schema

4.3 Populating Database with Component-based Service Information

In Figure 4.3, there is an invocation-based on-line service for a basic internet-based user interface. This is a typical method that can be called from a Java

Applet when a customer visits a particular businesses' website. The `getUserName` method would be used to gather information from the customer, in this case, about the product requested. The method would pass in the `customer_locale` parameter specifying the server location information from the internet customer. This information can be easily populated in the WARP database. Using the `getUserInfo` method, the actual method name is used as the `Service_Name` in the `Service` table. The return and parameter names (`Product_name` and `customer_locale`) are used in the `ReturnValue` and `ParameterValue` tables, respectively. The Site Manager agent is installed on the on-line business server and introspects all services and populates a centralized WARP database. The Site Manager Agent also generates unique keys to identify the components of the on-line services.

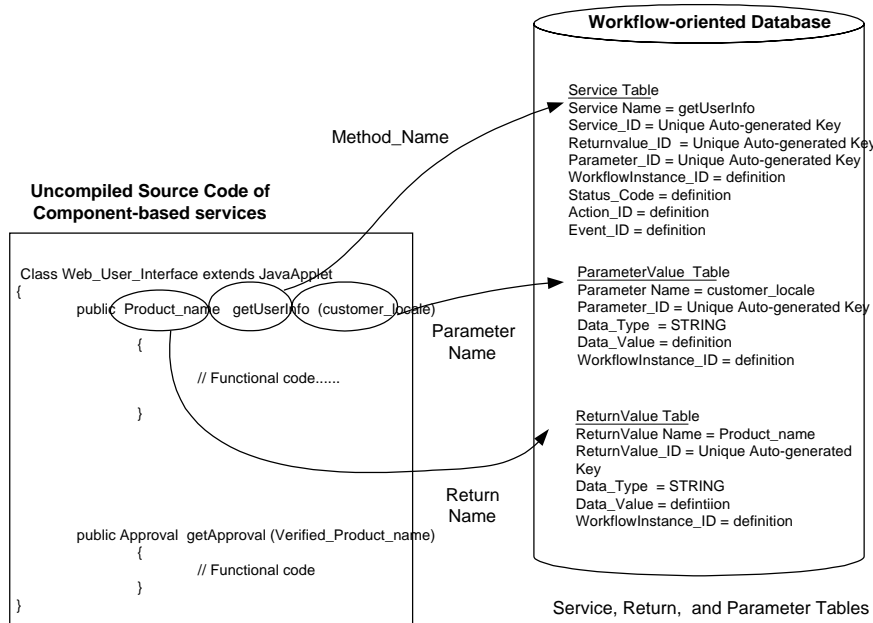


Figure 4.3 Capturing Service Information in the WARP Database

4.4 Creating the Initial Rose Environment

After the service-based information is populated into the WARP database, the Global Workflow Manager retrieves this information and sets up the initial UML environment in Rational Rose modeling tool (using the Rational Extensibility Interface (REI)). This environment will be enhanced with workflow-oriented information from a human workflow designer. The initial environment consists of class diagram illustrations of all of the services that were discovered on the on-line server. This initial view is defined in WARP semantics the *Service Representation View*. In Figure 4.4, there is a Service Representation View for the various enterprise services surrounding electronic commerce support for an on-line stock purchasing business. Each class in this view can represent a different division or server within the enterprise. In this case, internal services are separated from web (interface) services. The `getUserInfo` services used earlier is grouped as a web interface service.

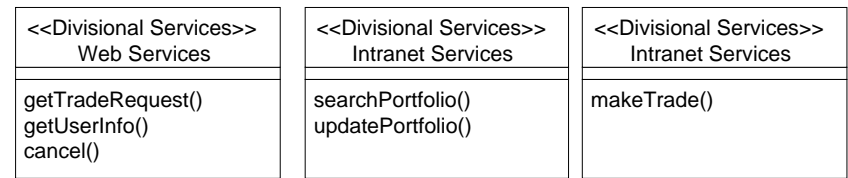


Figure 4.4 Service Representation View

4.5 User Driven Workflow Specification

After the initial environment is created, the next major step is accomplished by a human-in-the-loop. At this point, a workflow designer must configure the atomic on-line services for workflow operation. There are four major configuration issues that the workflow designer has to satisfy as described below.

1. Assign pertinent workflow roles
2. Group roles into the workflow policy
3. Describe the control and message flow during a workflow
4. Describe additional control and message flow for exception-handling cases and other nonfunctional concerns.

Assign Workflow Roles

As described in Section 2.1, a workflow is comprised of roles. In an enterprise setting, these roles can be realized by the component-based services distributed across the corporate information systems. The workflow designer must take his/her domain knowledge and group workflow roles to the underlying services that will be used to implement the specific role-based tasks. WARP semantics define this view as the *Role Aggregation View*. The Role Aggregation View, like the Service Representation View, is captured as a UML class diagram. In Figure 4.5.1, the three roles, Customer Interface, Portfolio Management, and Trading are creating and are associated with previously defined services. This case shows a simple association where all services in one division are accessible to a role. However, in complex cases, the associations can be further specified to show exactly which of the low-level methods are accessible by a specific role. In this view, the Trading role can use services from multiple intranet locations.

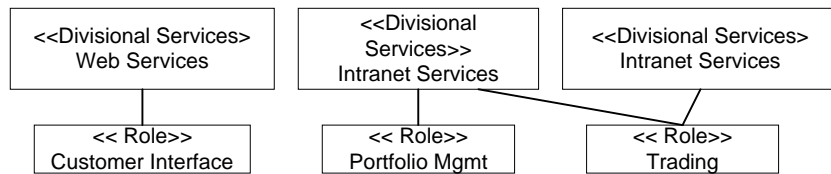


Figure 4.5.1. Role Aggregation View

Group Roles in Workflow Policy

The next logical step is for the workflow designer to define individual workflow policies. Before the actual control flow and message flow is defined, the WARP semantics require that the workflow designer specify all roles that can possibly participate in a particular workflow schema. This information is captured in the *Workflow Structural View*. This view again is captured using the UML class diagram. Each individual schema is associated with all roles that can possibly play a part in the workflow execution. In Figure 4.5.2, a stock purchasing schema is detailed where all of the aforementioned roles may play a part.

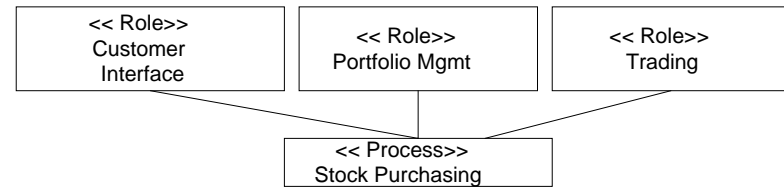


Figure 4.5.2 Workflow Structural View

Specifying Control and Message Flows

As the workflow executes, services are invoked either sequentially or asynchronously as per the workflow policy. The *Control Flow View* defines how various steps in the workflow are executed. The *Role Collaboration View* shows the specific messages (primitive data or objects) that are passed between individual services. These views are important to describing the interaction among the autonomous components. Both views are depicted using UML activity diagrams.

The *Control Flow View*, represented in a UML activity diagram, captures the sequence by which services are invoked for a particular workflow as defined in the Workflow Structural View. Since tasks can happen both concurrently and sequentially in the event-based component domain, the Control Flow View must have notations that represent both execution sequences. In addition, this view must support the joining of tasks. Joining occurs when multiple service must complete before the beginning of the subsequent service. In 4.5.3, the stock purchasing process is modeled in the Control Flow View. In this process, the customer requests a stock trade from a web service. The next service verifies the customer's portfolio information. The next step shows concurrent tasks, portfolio update and trading services. Another step could be added to show the join condition after the concurrent tasks. Both concurrent task and join conditions can be represented the stereotype <<FORK>> and <<JOIN>>, respectively. The <<FORK>> stereotype is shown in Figure 4.5.3.

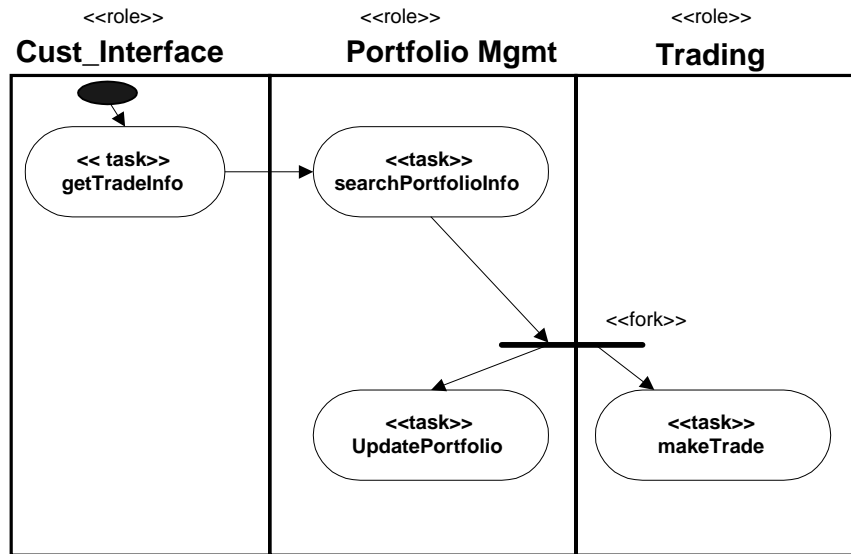


Figure 4.5.3 Control Flow View

The *Role Collaboration View* is used to show the messages that are transferred between roles. The WARP approach considers distributed components that were not particularly designed to interact. With this in mind, components may not fit together seamlessly. In some cases, one component-based service may offer more information than what is needed in the subsequent service. In other cases, the combination of information from multiple services may be the pre-conditions of a subsequent service. The purpose of the Role Collaboration View is to make this mapping. Considering a “call-return” style component service, a method might require parameters as a precondition, and the return from the invocation may be the post condition. In configuring a set of invocation-based services, the Role Collaboration View shows the mapping of the return(s) of the first service to the parameter(s) of the subsequent service. For invocation-based services, the Role Collaboration View can be modeled in a UML activity diagram. Dotted line arrows and objects in the activity diagram show the message exchange between roles. The transition is modeled with more UML classes. In the future, this transition can be modeled with XML schemas.

In this case, the return of the customer’s `getTradeInfo` service is the stock name, customer identification number, and account information. Also, this diagram shows how data can be shared across multiple roles. The same information from the customer service can be used in the portfolio management’s update portfolio service and search portfolio information service. This is useful with concurrent services. In addition, by defining names on certain message objects, the WARP architecture can dynamically specify message objects in the software.

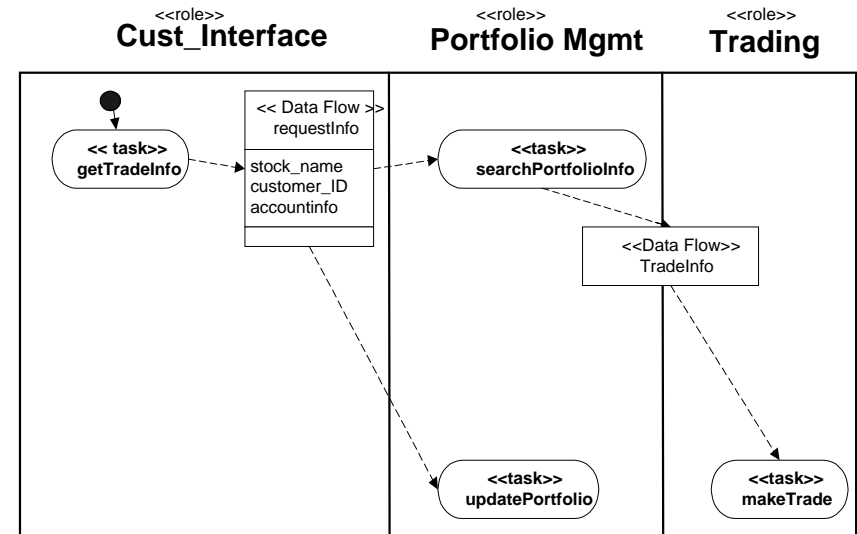


Figure 4.5.4 Role Collaboration View

Handling Exceptions Cases

The workflow designer may also model common nonfunctional concerns. The first concern is assurance of failure atomicity or recovery, when some domain-related problem occurs. Kamath [29] writes “... *failure atomicity requirements of a workflow govern how and what changes made by the steps of a workflow are made persistent depending on the success or failure of a workflow. Traditional transactions use serializability as the correctness criterion and hence failure atomicity corresponds to the 'all-or*

-nothing' property, i.e.; if a transaction commits, all the changes made by the transaction are applied to the database and if a transaction aborts, none of the changes will be applied to the database. This criterion applies irrespective of whether a system failure or a logical failure occurs. However this notion is not suitable for workflows. In fact, we need to be able to not always have "all-or-nothing" failure-handling functionality."

The WARP approach consists of base-level component-based services. These services can either abort or rollback, in the event of errors. For workflow, most times a complete roll-back of all component-based services is not necessary. The Failure Atomicity Diagrams in WARP are specifications for control in these failure scenarios.

A normal concern in the stock purchasing workflow is when a customer submits an incorrect user id. Failure handling is modeled using additional control flow and role collaboration views. These views are exactly the same as the regular functional views, the only difference is that the error cases are defaulted in the class diagrams as in Figure 4.5.5.

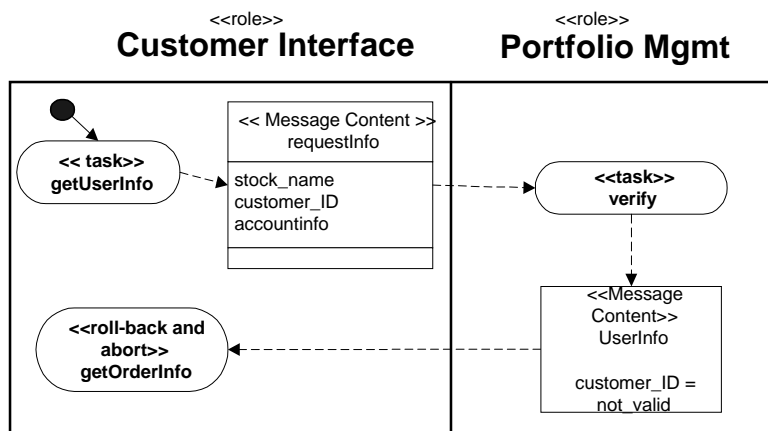


Figure 4.5.6 Failure Atomicity View

The new control specification would include system-level activities that represent the internal workflow-based error-handling knowledge. Workflow

designers would annotate activities with stereotypes like <<abort>>, <<roll-back>>, <<roll-back and abort>>, <<re-execute>>, <<roll-back and re-execute>>, <<initiation>>, etc. In Figure 4.5.6, an additional Role Collaboration View shows that stock purchasing process has error-handling concerns. The customer identification value is defaulted to "not-valid" in the Role Collaboration Model. In some cases, there would also have to be an additional Control Flow View. However, in this case the Control Flow View would be obvious so it is not shown here. There are also other nonfunctional cases that can be handled in much the same manner, but not discussed further here.

4.6 Generating XML and Populating the Database with Workflow Policies

The final two steps of the WARP configuration process, generating XML from the WARP UML-based models and populating the WARP database, can be combined. Once the user has specified the workflow schemas in the UML models, the corresponding Rational Rose model files can be read by the Global Workflow Agent and populated into the WARP database.

The transformation from Rose model files to the database schema is a stepwise process. In the first step, specific information from the models are parsed and used to generate XML files. The reason for using XML files between the UML models and the database is for scalability. By making a standard XML interface to the database, this architecture has the flexibility to receive workflow policy information from any source that can generate XML. This will allow the addition of other agents in the future. In the second step XML files are parsed and the information is used to populate the database directly. WARP has XML file formats for roles, workflow policy, and failure atomicity. The mapping from the Role Association View to XML to the database schema is illustrated in Figure 4.5.7. The name of the Role class in the Role Association View maps directly to the Role Name fields in both the XML file and the database schema. This is also consistent for the Service Name. If this was an unspecified association, then, by default, all service names from the Web Services class would be captured in XML then into the database schema. The Service ID fields are matched from the identification keys generated in the earlier population processes where the base-level services are inserted into the database.

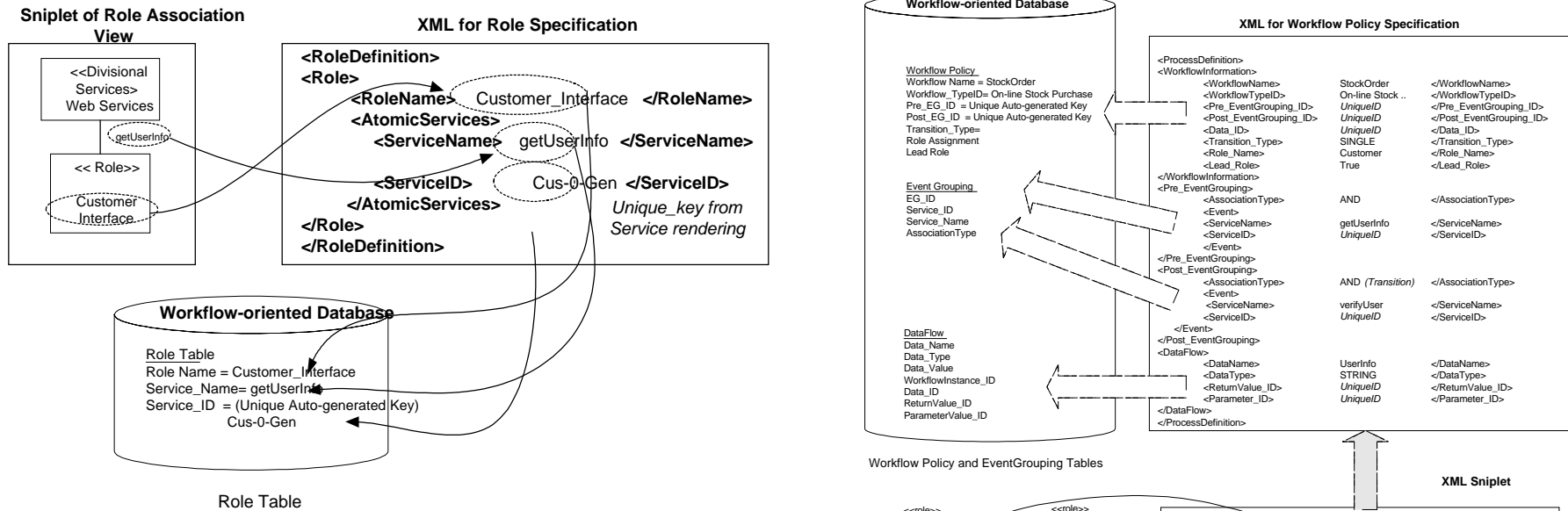


Figure 4.6.1 Mapping Roles to the WARP Database

Probably, the most significant mapping is the process that populates the Workflow Policy and EventGrouping tables. The Control Flow View and Role Collaboration View are combined into one XML File that is subsequently populated into the Workflow Policy and Event Grouping tables. This process is illustrated in Figure 4.6.2.

When mapping to the Workflow Policy table, the Control Flow View gives the sequence in which the services will be invoked. The Role Collaboration View gives information about which return value will be mapped to which parameter value. As previously stated this is important when there are multiple returns and multiple parameters. The process in Figure 4.6.2 shows a small subset of the mapping that occurs in the stock purchasing workflow. This illustrations shows the feasibility of this mapping, but there is also other mapping that can not be shown in the scope of this paper.

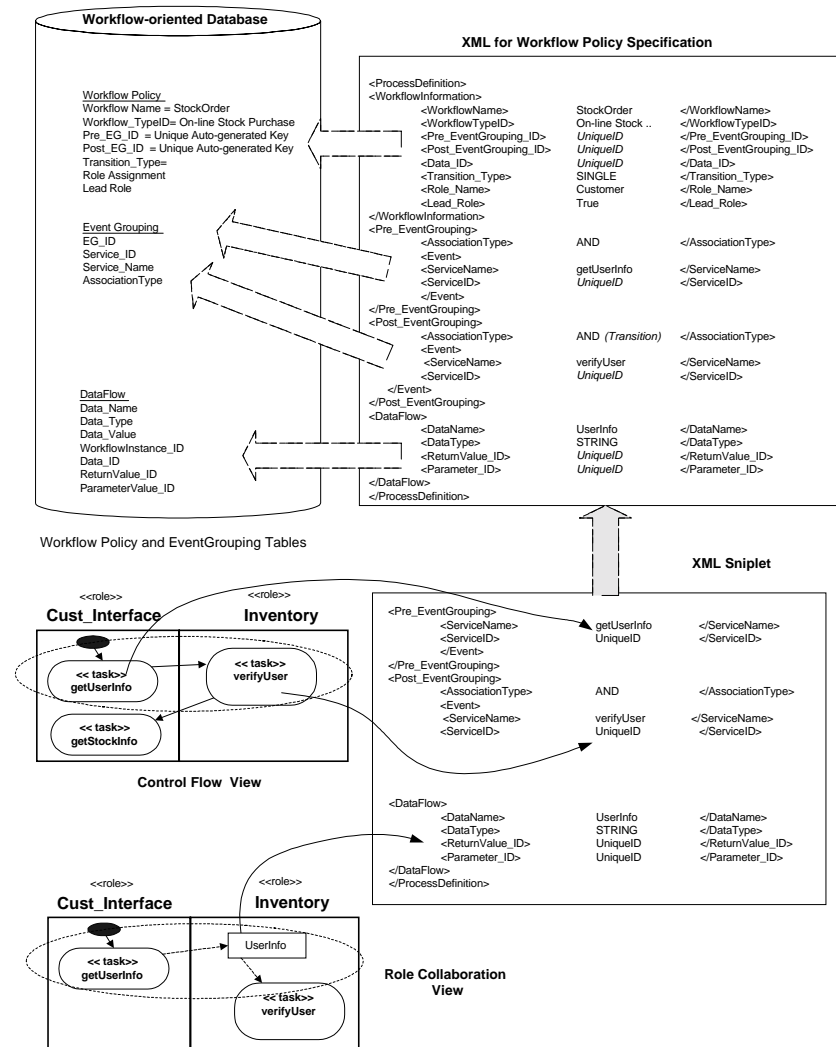


Figure 4.6.2 Mapping the Workflow Policy to the WARP Database

The WARP configuration process is a comprehensive approach to including workflow operational information to pre-existing distributed component-based services. There are many other standards for representing workflow, but we choose the industry standard object-oriented representations of the UML. Taking a bottom-up approach, the use software engineering-based notations, should lend ease to the workflow designer. In this domain, workflow designers would have to work closely with the software architects and designers of the services to create these evolving workflow policies.

5.0 Workflow Management and Execution using WARP

The WARP automated operational process is tightly integrated into the WARP shared database as in Figure 4.1. Using the database for policy information helps in the dynamic reconfiguration of the workflow of distributed components. WARP application layer agents (Role Manager Agent (RMA) and Workflow Manager Agent (WMA)) self-configure themselves based on the policy in the shared database. Each time a new workflow instance is initiated, these agents decide on the next move based on the policy in the workflow-oriented database. The following sections detail the operational self-configuration/initialization of the agents and the operational processes (normal or nonfunctional (error-handling)) of the workflow.

5.1 Agent Self-Configuration/ Initialization

The RMA and WMA are initiated by the SMA and GMWA respectively after the workflow policy information is populated in the shared database. The RMA and the WMA have different responsibilities in the operation of the workflow. The RMA acts as a broker for individual roles, and the WMA acts a broker for the workflow process. Both of these agents must configure for these responsibilities for both normal operation and nonfunctional or error-handling cases. At this point, all workflow information is populated in the workflow-oriented database. Both WMAs and RMAs query specific information in self-configuration. Based on the queried information, both agents can instantiate event listeners to notify them when events have been multicast. These multicast events are the basis for the workflow operations. The following section looks at the self-configuration by agent.

Role Manager Agent Self-Configuration

The RMA self-configures based on the role that is supplied to it. During workflow operation, the RMA merely invokes an action after being notified of a pertinent event. The RMA does not differentiate between normal operation and nonfunctional operation. In both cases, the RMA executes services based on some preceding event. This event can be either a previous service completion event or some error-based event. The main difference is that for regular operation these rules are extracted from the Workflow Policy table and for error-handling cases, this information is extracted from the Failure Atomicity table. Consequently, the RMAs can be classified as reflex agents[ref], as they encapsulate Event/Action rules. The general self-configuration steps of the RMA are captured in Table 5.

RMA Self-configuration	
1.	Query the policy table to find a list of rules that will be performed by the predefined role
2.	Build TaskObjects that encapsulate the E[C]A rules
3.	Register for each relevant service completion event as defined in the TaskObjects.

Table 5.1 RMA High-Level Self-Configuration Steps

The first step in the RMA self-configuration is to capture the Event, Condition, and Action rules (E[C]A) from the Workflow Policy table. These rules can be similarly captured from the Failure Atomicity table for error-handling cases. The query that is executed to capture the E[C]A rules is shown in Table 5.2, in this case for the Customer Interface role. In this query, the predicate event, action, type of transition, and lead role are collected from the workflow policy table based on the Customer Interface role. The transition type indicates what constraints are on the rule. This transition type will be further defined in the operation of the workflow later in this Chapter. The lead role field indicates if this particular transition is the first one in the workflow definition. Again, this will be used later in WMA configuration and operationally.

SQL Query to Uncover E[C]A Rules for Normal Operation
<pre>SELECT pre_eg_id, post_eg_id, transition_type, lead_role FROM workflow_policy WHERE role_assignment like 'Customer Interface';</pre>

Table 5.2 Query for Capturing the E[C]A rules

In the second step, the service name for the action is queried. The E[C]A rules and the actual service name for the action are used to populate the TaskObjects. These TaskObjects hold the instructions that the RMA will comply to. The query that returns the service name from the action is detailed in Table 5.3. In addition the definition of the TaskObject is shown in Table 5.4.

SQL Query to Obtain Low-level Services and Status Information for Service for Rater Reflection
<pre>SELECT service_name FROM event_grouping WHERE eg_id like 'SPECIFIC EG_ID';</pre>

Table 5.3 Query for Action

Code for TaskObject which Encapsulates the E[C]A
<pre>public class TaskObject implements Entry { public String Instance_ID; public String EG_ID; public String Action_ID; public String Transition_Type; public String Condition; public String workflowType_id ; public Service preService; }</pre>

Table 5.3. Task Object Definition

The RMAs and WMAs use the underlying Java event model. The RMAs have RMA_Listener objects that become active when a RMA takes a role. The TaskObject class defines a workflow step, thus is used to configure the RMA_Listener. The RMA_Listener implements the Java ActionListener interface. The RMA_Listener brokers events from a JavaSpace server [Ref]. In configuration, the RMA_Listener first configures the JavaSpace server in order to receive notifications for all the predicate events. A successful notification returns a key that is used to identify the proper TaskObject containing the pertinent action to complete. Later we will discuss the actionPerformed method which is invoked during operation the workflow executes. A pictorial view of this configuration is illustrated in Figure 5.1. This pictorial view also shows how service-based information is acquired by the agent to make reflective connections to the component-based services. This process runs independent of the workflow-oriented self-configuration. Also, this figure shows that the same process for normal operation can be used to gather information for error-handling cases if the Workflow Policy table is substituted with the Failure Atomicity Table.

5.2 Workflow Manager Agent Self-Configuration

The WMA has a far more complicated job. Consequently the WMA has 2 separate listeners. The WMA handles all nonfunctional concerns described earlier. These listeners are configured for the following functionalities as detailed in Table 5.4

Functionality or Concern	Listener Name
Making Workflow Instance Ids and starting a workflow	WMA_WKInstanceListener
Failure Atomicity	WMA_ErrorListener

Table 5.4 Types of WMA_Listeners

WMA_WKInstanceListener

The WMA_WKInstanceListener listens for the first event that initiates a workflow. This first event can be some system event, like a timer, or can be the completion of some service. When this event is captured this listener creates a

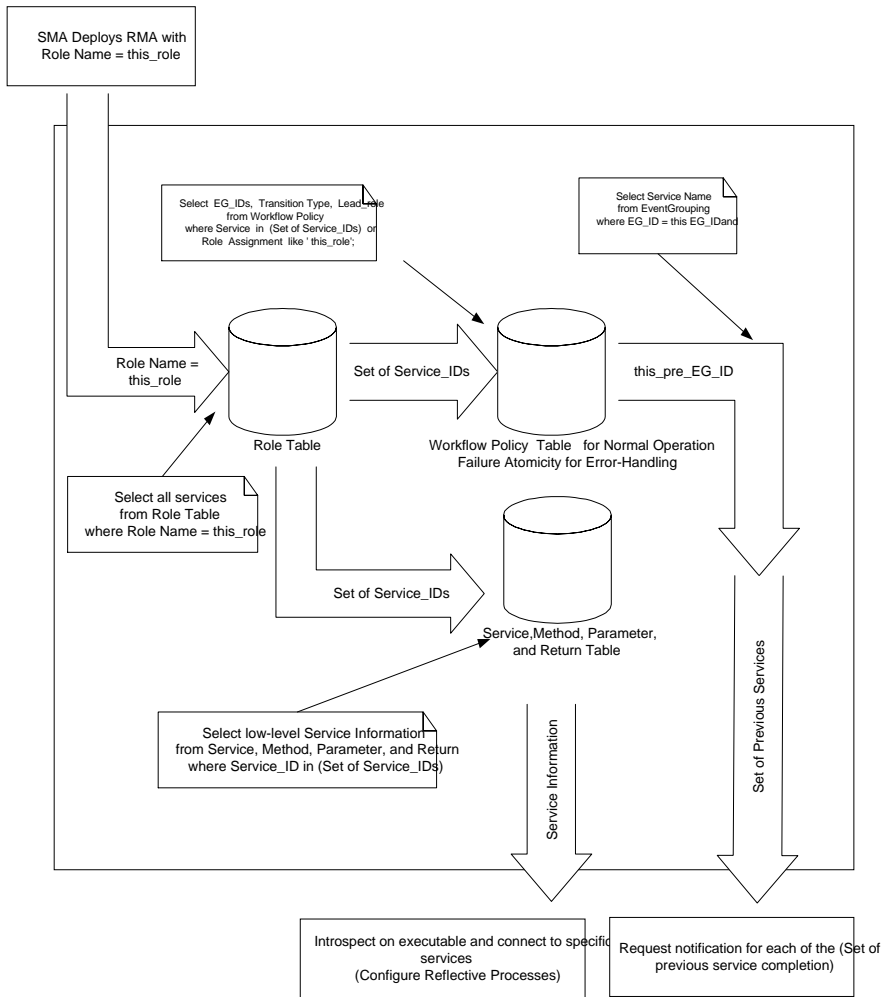


Figure 5.1 RMA Self-Configuration

unique workflow instance ID and fires an initiation event to start the first service. This initiation event is received by the first RMA in the workflow. The steps for this listener are detailed in Table 5.5.

WMA_WKInstanceListener Self-configuration
1. Query for the first event (service completion/job request) in the prescribed workflow process
2. Instantiate listener for the those events

Table 5.5 WMA_WKInstanceListener

The query to capture the first event of the Stock Purchasing process is detailed in Table 5.6. This query simply returns all Event Grouping Ids that are the first step in a specific process (i.e. where Lead Role equates to true).

SQL Query to Uncover first event in Workflow Process
SELECT pre_eg_id FROM workflow_policy WHERE workflow_name like 'Stock Purchasing' AND lead_role like 'True';

Table 5.6 Query for First Event in Workflow Process

WMA_ErrorListener

The WMA_ErrorListener listens for events with error conditions. When an error condition is captured, the WMA starts an error-handling workflow that fires events that roll-back, abort, re-execute, etc. based on the nonfunctional modeling. This functionality is identical to the functionality of the WMA_WKInstanceListener. The only differences are that this listener also listens for both specific conditions on the events and the service name. It queries the Failure Atomicity table, which has a Condition field.

SQL Query to Uncover first event in Error-Handling Process
SELECT pre_eg_id, condition FROM failure_atomicity WHERE workflow_name like 'Stock Purchasing' AND lead_role like 'True';

Table 5.7 Query for Error-Handling Configuration

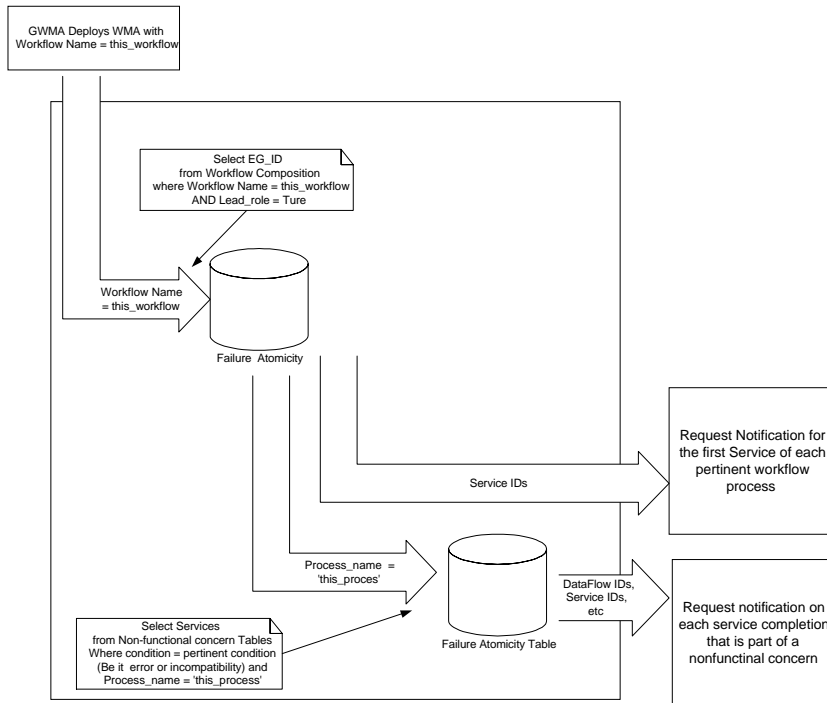


Figure 5.2 WMA Self-Configuration

The pictorial view of the overall WMA configuration is displayed in Figure 5.2. This figure shows how notifications are configured for the anticipated services. The implementation for this functionality is the same as the self-configuration/initialization for the RMA except for the change in queries. Operational Processes

At this point, the agents have self-configured and initialized themselves. All agents have registered to receive specific events pertinent to their interaction. A summary of what type of events the individual agents are listening for is detailed in Table 5.8. At this point, the architecture waits for some job initiation or the

completion of some service to start a workflow instance. This section details the various modes of operation that occur in situations when these events or service completions are captured.

Agent	Awaiting
RMA	- Completion of a predicate service
WMA	- Initiation Service completions or Job Initiation Events - Services with an Error Condition

Table 5.8 Registered Services after Initialization

5.3 Workflow Operation

The standard operation of the WARP agents occurs when a job initiation event or service completion is captured and the workflow process executes normally without error. This operational view is illustrated in Figure 5.3. To summarize this standard operational flow, initially a job event or service completion initiates the workflow instance. The WMA captures this event and builds a unique ID based on the correct workflow policy as defined in its TaskObject. The WMA broadcasts an event that initiates the workflow. From this point on, the RMAs coordinate to complete the workflow. The RMAs have reflex engines that await certain service completions and perform the pertinent actions. The implementation of the RMAs for this standard operation is encapsulated in the Role Manager Agent component.

Operation with the And Condition

In some cases, one service is dependent on the completion of multiple predicate services or events. This AND condition is another case that is handled in the operation of the WARP agents. The big difference is that before the RMA invokes the reflective component, it first checks to see if the other service had been completed in the Event Table or the JavaSpace. If so, then it invokes the service normally, otherwise it exits. If it exits, then once the other AND event is captured the action will automatically be executed through the same inquiry process.

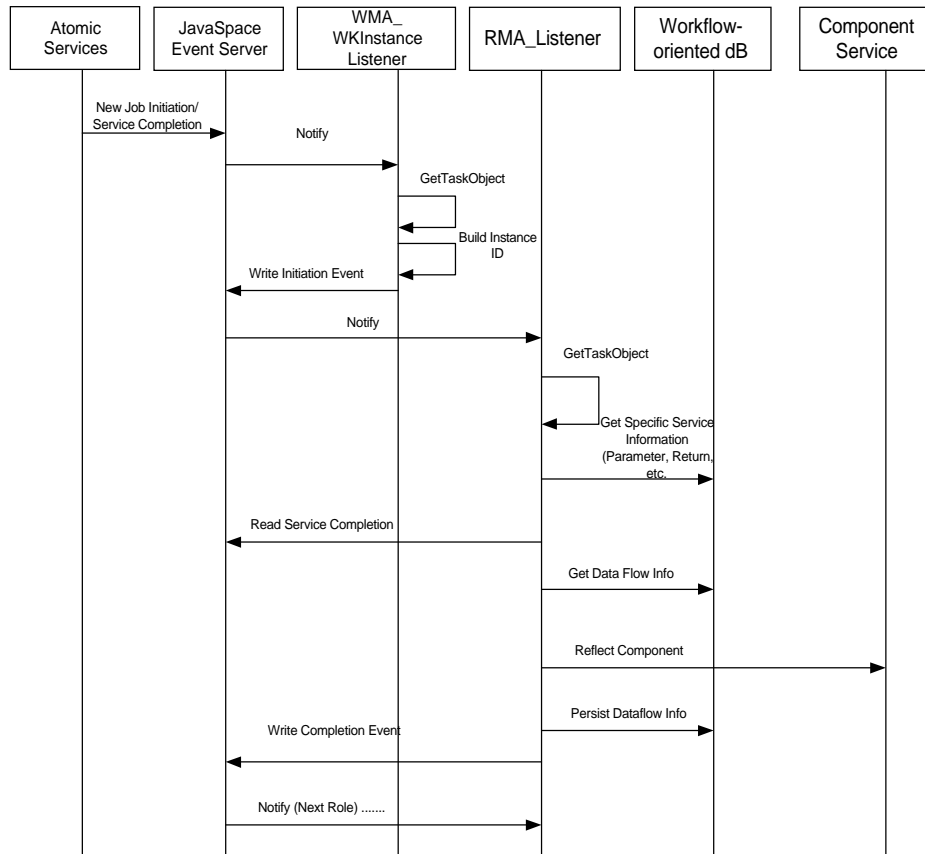


Figure 5.3 Standard Operation Flow

Operation with the ExclusiveOR Condition

The ExclusiveOR condition occurs when multiple services have the same predicate event. In these user-specified cases, WARP agents must ensure that only one RMA can fulfill the service. This is a relatively simple condition to

ensure. This condition can be fulfilled by specifying how the agent communicates to the JavaSpace event server. This operation is the same as the standard operation but in these cases the RMA must use the *take* command to take the service completion from the space as opposed to using the *read* command (i.e. the read command only gets a copy of the event). By taking the service completion event from the space, no other RMA can take that same event.

Operation for the Error-Handling Case

The error-handling case occurs when a system or logical error occurs during the workflow operation. Typically when there is an error, the service event is populated with an unfavorable condition. The WMA_ErrorListener is initialized to listen for service completions and conditions. By setting the condition to some error code, the WMA will be notified for error cases. Incorporated in the workflow policy are specific workflow processes that handle various cases of errors as defined by the workflow designer. The operation of the error-handling case is the same as that of the standard operation once the error event is captured. The WMA just initiates the beginning of a workflow that is specifically for handling this case. The RMAs also are initialized to handle these cases.

6.0 The WARP Prototype Design, Implementation, and Performance

This implementation of the WARP architecture has been developed to dynamically coordinate a workflow of distributed JavaBeans, Java Applets and/or Java Servlets. Typically Java Beans are packaged for event-based communication while the remote Java Applets have a call-return style of communication. Java Servlets are tightly integrated with the web server and receive HTTP-based commands as events that initiate their functions. The prototype of the WARP architecture discussed here seamlessly manages a specified workflow of distributed components, be it JavaBeans, Java Applets, or Java Servlets. The following sections describe this implementation of the WARP architecture.

6.1 Software Design

This prototype builds WARP agents using pure Java as the programming language. All agents are JavaBean/Applets, meaning they have both event-based and call-return style of communication. Using JavaBean/Applets give the agents graphical user interface and Internet functionality. This allows the agents to be accessible from Internet. The agents communicate through a JavaSpace event server. The workflow-oriented database is implemented with an Oracle database. The WARP Representations are captured using Rational Rose 98i developer tool. Using the Rose Extensibility Interface (REI), agents can access the representations via the model (.mdl) file created by Rational Rose. The agent architecture was developed from a strict object-oriented design. The software design of the prototype is illustrated in Figure 6.1.

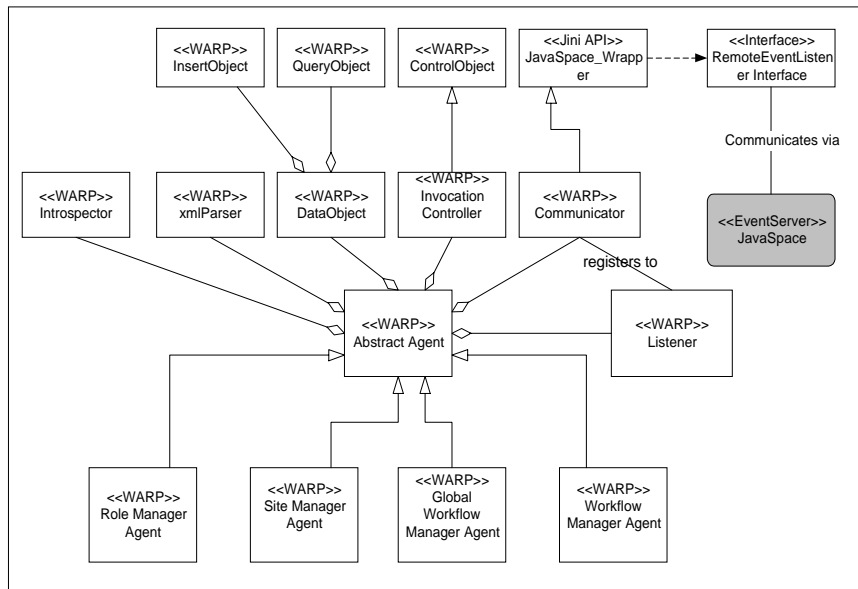


Figure 6.1 Object-oriented Design of WARP Prototype

The design of this prototype follows the WARP architectural design explicitly. The policy execution engine is incorporated in the Invocation Controller component. Each type of WARP agent implicitly has all of the functions but may use them for different purposes as stated in the previous section.

The functionality of the Abstract Agent is the most important functionality in the design. Each agent can transmit events over the event server, register for events, and receive events. The functions are delegated to the Communicator class and the Listener classes. The Communicator class is used to send and receive events, while the Listener class is used to register for events. The Listener class is activated when a notification returns based on an event registration. The implementation is approximately 20,000 lines of program code.

6.2 Performance Considerations and Results

An important consideration to the WARP approach is the impact of the overhead caused by this real-time configuration. In order for automated configuration that is continually reconfigured, the agents must check the workflow-oriented database for each action stemming from an event. More specifically, the agent must ensure the correct service to be invoked. Since the workflow policy is captured in a database, the policy queries cause a great deal of overhead. The relevant performance concern must consider what is the magnitude of overhead in WARP versus systems that do not support automated configuration. Another concern is how overhead in the WARP approach relates to the overhead in other approaches. This section looks at the overhead imposed by the WARP system specifically and also in context of other systems.

Overhead Associated with WARP Approach

The evaluation of overhead in this section is based on a system that uses all of the same technology but has all hard-coded interactions. This means that the system is static and only performs a certain type of process. This type of system has no reconfiguration functionality and all nonfunctional concerns are bundled into the implementation. Conceptually, this system should have the fastest performance and in the case of Java-based services, this fact has been proven. A variation from this base-level system is allowing a memory-based table where the components can operate with minimal workflow characteristics.

The WARP approach can run in several modes. One mode initially configures all agents whereas decision support for workflow interactions are stored in memory. Another mode is storing only events and actions but allowing the agent to query the database for services as the workflow executes. The final option does not store any information allowing the agents to access policy and services at operation time. The actual percentages of overhead incurred in the Java-based prototype is listed in Table 6.1.

Number of Workflow Steps	Java Components with Memory-based workflow look-up tables	WARP Configured Components that check Database only at configuration (Option 1)	WARP Configured Components that check services at each step (Option 2)	WARP Configured component s that check policy and service each step (Option 3)
3	2%	9%	13%	23%
5	2%	10.7%	14%	27%
50	2%	12%	15%	37%

Table 6.1 Percent of Overhead Imposed on Workflow Operation Versus Hard-coded Interactions

The optimal operation allows the role-level agents to check subsequent services at run-time (Option 2). This allows services to change at run-time seamlessly. Configuration at run-time for every instance is not practical (Option 3), because most workflows do not change for every instance. Option 1 does not give any features above that of a static workflow, but as seen in Table 6.1 has more overhead than the base-level case. For a typical workflow, the WARP approach has about 15% overhead. In the next section, the WARP approach is compared to other systems in context of overhead and functionality.

Differences of Overhead between WARP and Other Systems

The WARP approach handles a considerable amount of functionality that is not handled in other systems but at some cost. The list of systems and functional capabilities are listed in Table 18. The closest system in respect to

functionality, is the system developed by Shrivistava at the University of Newcastle. This system still does not have distributed control or the separation of nonfunctional concerns. Thus, this system lacks the flexibility and reusability of the WARP approach. Though the percent of overhead is lower than the WARP approach, the reduction of overhead is not a considerable improvement for the lack of major capabilities. The other systems (WIDE and Singh) were not able to provide overhead

Workflow System	Heterogeneous Components	Distributed Control	Automatic Configuration	Handles Configurable NF Concerns	Average Overhead
WIDE	N	N	Y	N	N/Avail
Newcastle	Y (CORBA compliant)	N	Y	N	13% (CORBA)
Singh	N	N	N	N	N/Avail
WARP	Y (All JAVA)	Y	Y	Y	15%

Table 6.2 Survey of Workflow Systems

The WARP approach is considerable enhancement over the aforementioned approaches with relatively low overhead. The WARP approach considers the important configurable architecture issues (reusability, distributed control, nonfunctional concerns, and universal adaptability).

Scaleability of the WARP Implementation

The scope of the workflow can increase in two directions. One direction is an increase in the number of steps in the workflow. In Table 6.1, three workflow processes of varying lengths are shown grouped by steps. There are 3, 4, and 20 role agents for the 3, 5, and 50 steps, respectively. As the number of steps and roles increase, there is a relatively low increase in overhead. The reason for this is because the role agents have their own processes. In most workflow domains, there are typically less than 100 steps, especially at the high-level (i.e. component level). However, subprocesses under the high-level component invocations can have much higher numbers of steps. The WARP approach considers the high-level component invocations, therefore this implementation is scaleable for the increase of workflow steps and roles. Even

if the number of steps increase higher than normal levels, this architecture can support it by reducing the services that a particular role can invoke and distributing the agents across more machines or machines with higher processing speeds.

Also important is the number of instances that can be processed at the same time. In understanding the scalability for an increase in workflow instances, a 10 step workflow process was used. Table 6.3 and Table 6.4 show various overhead associated with this 10-step process based on different increments of concurrent instances. These results were taken by setting up print statements at the beginning and conclusion of each component instance and at the beginning and ending of the workflow instance.

Workflow Instance (10 steps/ 6 Roles) Mode (Option 2)	Completion Time/ Overhead (10 Instance)	Completion Time/ Overhead (20 Instances)	Completion Time/ Overhead (50 Instances)	Completion Time/ Overhead (100 Instances)
Overhead for Reflective Components without WARP Architecture vs. One instance (8.2 sec)	9.56/ 16.6%	14.2/ 73.2%	16.01/97.5%	56.7/ 700%

Table 6.3 Regular Component Overhead Associated with an Increase in Workflow Instances

The first conclusion, based on the results in Table 6.3, is that overhead imposed by the Java-based reflective processes on multiple component-based processes is excessively high. Table 6.4 shows the completion time for sequenced component invocations that were invoked through reflection. Therefore, the overhead of the reflective processes are combined in these percentages.

Workflow Instance (10 steps/ 6 Roles) Mode (Option 2)	Completion Time/ Overhead over values in Table 22 (10 Instance)	Completion Time/ Overhead (20 Instances)	Completion Time/ Overhead (50 Instances)	Completion Time/ Overhead (100 Instances)
Overhead for Reflective Components with WARP architecture vs. Regular Component in Table 22	10.87/ 13.7%	16.27/ 14.6%	18.51/ 15.6%	65.32/ 15.2%

Table 6.4 Additional Overhead Caused by the WARP Architecture as Workflow Instances Increase

The difference in processing time for many instances as opposed to processing time of one instance (8.2 seconds) is great. There are two main reasons for this large overhead. One reason is associated with the increase in component-based services. For threaded registry-based services, new processes are started for each instance. As the instances increase, a large number of system resources are used in starting and invoking the large number of component services. This overhead is to be expected in any workflow system and is independent of the WARP architecture. However, the overhead cost imposed by the reflection processes are extremely high especially for large number of processes. This WARP implementation uses Java-based reflective functionality that tends to fluctuate depending on the operating system. These test runs were performed on two Windows NT-based machines. Without being able to separate invocations from the reflection processes, it is not possible to completely assess to what extent the reflective processes imposed additional overhead.

Another conclusion was made from the results in Table 6.4 shows that the overhead imposed specifically by the WARP architecture does not increase significantly with the increase of instances. This table shows the amount of additional overhead caused by the WARP architecture on top of the overhead contributed in Table 6.4 by the Java-based reflective processes. This additional

overhead is at an average of 15% with the varying number of instances. This is expected because new instances do not cause new role agents to be constructed. The role agents have relatively simple instructions that do not significantly increase processing time. As explained in section 4, the agents follow Event[Condition]/Action rules or reflex rules. The agents must execute the resulting action or “reflex action” when a certain event and condition is received. New workflow instances reuse the same configured reflex rules, therefore system memory is not significantly increased.

In conclusion, the WARP architecture is scalable for both the increase in workflow process steps and for the increase in workflow instances. Future work would include exploring other reflective architectures and their impact on overhead in the system. In doing this, additional results can be obtained for the effects in the use of reflection. However, in-depth study in improving the performance in this architecture is out of the main scope of this paper.

7.0 Related Work

The culmination of all the relevant work has led to new area called “software agent-oriented workflow”. Software agents have been researched over the last decade for systems where there are autonomous entities that need coordination [54][28][56][32]. The automated workflow configuration domain has proven to be a relevant area. In the past few years, the term, software agent-oriented workflow, has been devised as an area where agents are be used to fulfill the workflow roles. Since workflow roles and tasks tend to evolve as the workflow schema/business need changes, these roles can be managed by autonomous agents. This direction has led to the several software agent-oriented workflow approaches and systems [12][13][36][46][52][55][58]. Also several workshops and conference tracks have been dedicated to agent-oriented workflow, most recently the Track on Software Agent-oriented Workflow at the 2000 International Conference on Artificial Intelligence, Las Vegas, NV June 2000.

The WorkWeb [55] system is a software agent-oriented workflow that manages and controls office resources. Office resources are processes that occur in an operational corporation on a daily basis. These process consists of a workflow for complete expense reporting, human resource allocations, appraisal processes, etc. WorkWeb is an agent-based architecture that autonomously manages each workflow process. The system relies on the communication of all of the workflow instances to assure the optimality of the system. Starting a new

workflow in the WorkWeb system is a process of automated collaboration, that distributes instance responsibilities across the application.

The WARP approach differs from the WorkWeb System in several ways. The WorkWeb system handles the coordination among various threads in the workflow operation. The WFMS in the WorkWeb system is static and the services are fixed. WARP uses a bottom-up compositional approach that coordinates a changing set of services. These services can have different means of interaction be it invocation-based or event-based. The WARP approach encapsulates the coordination of multiple threads in the workflow operation within the operation of the WARP agent middleware architecture. Moreover, the WARP approach considers a larger set of services that are dynamic in nature.

Singh [52] allows workflow designers to specify design time rules about workflow interactions. These rules are incorporated into agents that manage the individual workflow roles. Using specialized algorithms the agents can help the workflow perform more accurately and efficiently. This approach is more of a design time approach. It is not relevant to real-time business systems. The WARP approach is more toward automated configuration than workflow design.

There is also a collaborative effort (The CoABS Grid) [15] by DARPA and numerous commercial and academic labs. The CoABS Grid is oriented toward the dynamic coordination of autonomous services, at this time mostly agent-based services. Similar to this effort is work done by Subramanian[3] at the University of Maryland. His work also entails connecting component-based services using agent technology.

Both the work at the University of Maryland and that of the CoABS Grid are similar to WARP’s underlying processes, but neither approach explicitly models workflow-based transactions. Both of these works are toward the connection of heterogeneous services.

Meng [36] devises a mobile agent based workflow where the rule-based workflow control is encapsulated into a mobile agent that sequentially executes the role-based services at each workstation. This is a promising distributed agent solution. This research is still early in development. At this point unlike the WARP approach, this research does not incorporate nonfunctional concerns and business to business communication has not been specified.

Chen and Griss [13] define “dynamic agents” for workflow configuration. These agents consist of a fixed part (services) and a dynamic part (workflow specification). This approach uses XML messaging to assure a standard data format. This is also a Java-based approach with a prototype toward the e-

commerce environment. As aforementioned agent-oriented workflows, this approach only considers functional concerns. In this work, agents are programmed via XML messages. This does not provide the same ease of reasoning as the model-based representations in the WARP approach.

8.0 Summary

This first major contribution of this work is a reusable agent-based architecture that supports the automated configuration of workflow-based components. This architecture, unlike most systems, supports a compositional approach to workflow development where existing reflective services can be configured for workflow [9] [11].

Also in this work, representations in the UML have been specified to support this automated configuration. These representations support both the functional and nonfunctional nature of the workflow. This separation of workflow-based functional and nonfunctional concerns is a contribution to the current field of aspect-oriented programming. This work extends the current research on the separation of concerns into the workflow area. Moreover, this work is toward the integration of concerns at the component-level as opposed to the current research that integrates the concerns at the code-level [6][8]. This approach is necessary for dynamically changing architectures where application domains must evolve without changing source code. This work also presents a systematic method of combining the UML-based workflow representations to configure an agent-based architecture and relational database for the management of workflow.

Finally, there is a reusable operational process for combining the functional and nonfunctional specifications. This operational weaving extends the flexibility of the architecture with respect to the addition of services and the change of workflow policies.

The WARP approach has particular applicability to the e-commerce domain by providing a standard method and process for distributed workflow specification and automated configuration. This approach presents the design of mechanisms that can support the current changing on-line market environments [7][10].

References

- [1] Abiteboul, S., "On Views and XML," *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp 1-9, 1999
- [2] Allen, R. J., Douence, R. and Garlan D., "Specifying and Analyzing Dynamic Software Architectures," *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE98)*, March 1998.
- [3] Arisha, K. Subrahmanian, V.S. et. Al, "IMPACT: Interactive Maryland Platform for Agents Collaborating Together," *IEEE Intelligent Systems Magazine*, Vol 14, Nr 2., pp 64-72 1999
- [4] Bacon J., Moody, K., Bates J., Hayton R., Ma C., McNeil A., Seidel O., Spiteri M. *Generic Support for Distributed Applications*. IEEE Computer 33(3): 68-76 (2000).
- [5] Batory, D. and O'Malley, S. "The Design and Implementation of Hierarchical Software Systems with Reusable Components," *ACM Transactions on Software Engineering and Methodology*, 1992.
- [6] Blake, B. and Goma, H. "A Knowledge-Based Approach to Support Workflow Automation in Component-Based Services" (under review).
- [7] Blake, M.B. "KOJAC: Implementing KQML with Jini to Support Agent-Based Communications in Emarkets," *AAAI-2000 Workshop on Knowledge-based Electronic Markets (KBEM) at the 17th National Conference on Artificial Intelligence*, Austin, TX, August 2000
- [8] 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
- [9] Blake, M.B. and Bose, P. "An Approach to Automated Workflow Configuration through the Composition of Distributed Invocation-Based Services," *5th International Conference on Reliable Software Technologies*, Berlin, Germany June 2000 (presentation)
- [10] Blake, M.B., "A Development Approach for Workflow-Based E-Commerce using Reusable Distributed Components," *2000 Americas Conference on Information Systems (AMCIS2000) (Track on Workflow Technology and E-Commerce Applications)*, Long Beach, CA, August 2000
- [11] 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) (Session on Software Agent-Oriented Workflow)*, Las Vegas,

- NV, June 2000Booch, G., Rumbaugh, J., Jacobsen, I., "The Unified Modeling Language User Guide," Addison Wesley, Reading MA, 1998
- [12] Borghoff, U.M.,et al. "Reflexive Agents for Adaptive Workflows," *Proceedings of the Second International Conference and Exhibition on Practical Application of Intelligent Agents and Multi-Agents (PAAM'97)* (London, UK April 1997) pp.405-420
- [13] Chen, Q., Hsu, M., Duyal, U., and Griss, M. "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation," *Proceedings of the 4th International Conference of Autonomous Agents (Agents 2000)* Barcelona, Spain, June 2000.
- [14] Cingil I., Dogac A., Tatbul, N.and Arpinar, S. "An Adaptable Workflow System Architecture on the Internet for Electronic Commerce Applications," *International Symposium on Distributed Objects and Applications*, Edinburgh, United Kingdom 1999.
- [15] The CoABS Grid, Control of Agent-Based Systems (CoABS Project)
<http://coabs.globalinfotek.com/>
- [16] Cointe, P. "Reflective Languages and Metalevel Architectures," *ACM Computing Surv.* 28, Article 151, 1996.
- [17] 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.
- [18] Demers, F. and Malenfant, J. "Reflection in Logic, Functional and Object-oriented Programming: a Short Comparative Study," *Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI Montreal*, August 1995.
- [19] Dogac, A. et al. eds., "Workflow Management Systems and Interoperability," NATO ASI Series, Springer Verlag, Berlin, Germany, 1998.
- [20] Ellis,C.,et al. "Dynamic Change Within Workflow Systems," *In Proceeding of COOCS'95*, Aug 1995
- [21] Fradet, P., and Südholt, M. "Towards a Generic Framework for AOP," *Proceedings of the International Workshop on Aspect-Oriented Programming at ECOOP-98*, LNCS 1543, 1998.
- [22] Garlan D. and Shaw, M., "Software Architectures: Perspectives on an Emerging Discipline," Addison Wesley Publishers, 1996.
- [23] Garlan, D., Allen, R. and Ockerbloom, J. "Architectural Mismatch or Why it's hard to build systems out of existing parts," *Proceedings of the 17th International Conference on Software Engineering (ICSE 1995)*, Seattle, WA, April 1995.
- [24] Glushko, R. et al, "An XML Framework for Agent-based E-commerce," *Communications of the ACM* 42, 3 pp 106-107 March 1999
- [25] Gomaa H and Kerschberg, L. "An Evolutionary Domain Life Cycle Model for Domain Modeling and Target System Generation," *In Proceedings of the Workshop on Domain Modeling for Software Engineering, International Conference on Software Engineering*, Austin, TX 1991.
- [26] Graham, J. and Decker, K., 1999. "Towards a Distributed, Environment-Centered Agent Framework," *In Proceedings of the 1999 Intl. Workshop on Agent Theories, Architectures, and Languages [ATAL-99]*, Orlando, FL 1999.
- [27] Jacobsen I., Booch, G., and Rumbaugh, J." The Unified Software Development Process, " Addison Wesley 1999.
- [28] Sycara, K. and Wooldridge, M. "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems*, 1(1), p.7-38, 1998.
- [29] 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.
- [30] Klein, M., Dellacros, C., and Bernstein, A. eds., "Workshop Towards Adaptive Workflow Systems," *Proceedings of the CSCW-98 Workshop*, ACM Press, Seattle Washington, November 1998.
- [31] Kramer, J., Magee, J., and Finkelstein, A., "A Constructive Approach to the Design of Distributed Systems", *Proceedings of the 10th International Conference on Distributed Computing Systems*, Paris, May 1990.
- [32] Kumar, S. and Cohen, P. "Towards a Fault-Tolerant Multi-Agent System Architecture," *Proceedings of the 4th International Conference of Autonomous Agents (Agents 2000)* Barcelona, Spain, June 2000.
- [33] Lei, Y. and Singh, M.P., "A Comparison of Workflow Metamodels", *Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling at ER'97*, Los Angeles, CA, November 1997

- [34] Martin P. Robillard and Gail C. Murphy. "An Exploration of a Lightweight Means of Concern Separation," ECOOP 2000 Workshop on Aspects, Dimensions, and Concerns (ADC2000) June 2000.
- [35] Medvidovic, N. and Taylor, R., "A Framework for Classifying and Comparing Architectural Description Languages," ACM SIGSOFT Software Engineering Notes, Volume 22, Number 6, November 1997.
- [36] Meng, J. and Helal, S. "An Ad-Hoc Workflow System Architecture Based on Mobile Agents and Rule-Based Processing," *Proceedings of the 2000 International Conference of Artificial Intelligence (IC'AI2000)* Las Vegas, NV June 2000.
- [37] Murphy, G., Walker, R., and Baniassad, E. "Evaluating Emerging Software Development Technologies: Lessons Learned from Assessing Aspect-oriented Programming," IEEE Transactions on Software Engineering, 25(4):438-455, July/August 1999.
- [38] Oreizy, P., Medvidovic, N., Taylor, R.N., "Architecture-based Runtime Evolution," Proceedings of the International Conference of Software Engineering (ICSE'98), 1998
- [39] Parlavantzis, N., Coulson G., Clarke, M., Blair, G. "Towards a Reflective Component Based Middleware Architecture" ECOOP'2000 Workshop on Reflection and Metalevel Architectures, Sophia Antipolis, June 2000.
- [40] PeopleSoft, <http://www.peoplesoft.com>
- [41] Perry, D. and Wolf, A.L. "Foundations for the Study of Software Architecture," ACM SIGSOFT Software Engineering Notes, 17:4, October 1992.
- [42] Ranno, F., Shrivastava, S. and Wheeler, S., "A Language for Specifying the Composition of Reliable Distributed Applications," to be published.
- [43] Rational Rose and Rational Extensibility Interface (REI), www.rational.com
- [44] Redmond, B. and Cahill, V. "Iguana/J: Towards a Dynamic and Efficient Reflective Architecture for Java," ECOOP'2000 Workshop on Reflection and Metalevel Architectures, 2000
- [45] Reichert, M., "Supporting Dynamic Changes in Workflow Without Losing Control," *Journal of Intelligent Information Systems*, 1998.
- [46] Rouiller, A., Silva R., and Meira, L. "An Agent-Based Foundation for Adaptive Multi-Company WFMS," *Proceedings of the 2000 International Conference of Artificial Intelligence (IC'AI2000)*, Las Vegas, NV June 2000.
- [47] Rumbaugh, J., Jacobsen, I., Booch, G., "The Unified Modeling Language Reference Manual," Addison Wesley 1999.
- [48] Russell, S and Norvig, P., *Artificial Intelligence, A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey 1995.
- [49] Shaw, M. "Architectural issues in software reuse: It's not just the functionality, it's the packaging," *In Symposium on Software Reusability*, 1995.
- [50] Sheth, A., Worah D., Kochut K., Miller J., Zheng K., Palaniswami D., and Das, S. "The METEOR Workflow Management System and its use in Prototyping Significant Healthcare Applications," *Journal of Intelligence Information Management Systems*, pp. 185-215, 1997.
- [51] Shrivastava, S. and Wheeler, 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.
- [52] Singh, M. and Huhns, M. "Multiagent Systems for Workflow," *International Journal of Intelligent Systems in Accounting, Finance and Management*, Volume 8, pages 105-117 1999.
- [53] Sun Microsystems Inc., Java Beans Specification, Distributed Event Model Specification, Java Space Specification, and the Java Language Specification. <http://java.sun.com>.
- [54] Sycara, K., Decker, K., Anandeeep, P., Williamson, M. and Zeng, D., "Distributed Intelligent Agents," The Robotics Institute, Carnegie Mellon University 1995.
- [55] Tarumi, H., et al "WorkWeb System -- Distributed Multi-Workflow Control with a Multi-Agent System", *IPSI International Symposium on Information Systems and Technologies for Network Society*, World Scientific, Sep 1997.
- [56] Treese, Win, "Putting it Together: What's all the noise about XML?," *Networker* 2, 5 (Dec 1998) 27-29
- [57] University of Michigan, Department of Electrical Engineering and Computer Science, "Survey of Agent-based Architectures," University of Michigan, <http://ai.eecs.umich.edu/cogarch0/index.html>
- [58] Walshe, D. et al "An Interoperable Architecture for Agent-Oriented Workflow Management," *Proceedings of the 2000 International Conference of Artificial Intelligence (IC'AI2000)* Las Vegas, NV June 2000.
- [59] Weiss, A. "XML gets down to Business," *Networker* 3,3 pp 36-37, September 1999