

Object-Oriented Modeling Approaches to Agent-Based Workflow Services

M. Brian Blake

Department of Computer Science, Georgetown University, Washington, DC 20057, USA
blakeb@cs.georgetown.edu

Hassan Gomaa

Department of Information and Software Engineering, George Mason University
Fairfax, Virginia 22030, USA
hgomaa@gmu.edu

Abstract. With the increasing popularity of component-based services and semantic web services, the idea of specification-driven service composition is becoming a reality. With the distribution of these autonomous services, a realizable goal will be the transformation of the Internet into a *universal service repository*. In such an environment, intelligent agents can play a significant role in configuring and enacting the workflow composition of the atomic distributed services to create entirely new higher-level services. In this work, there is a large-scale agent-based architecture to support such a distributed service environment. Furthermore, we introduce an object-oriented modeling and software engineering approach towards the development, configuration, and operational control of the agents that manage processes in this cross-organizational workflow environment.

Keywords

Agent architectures, Software Process, Workflow, UML, object-oriented modeling

1. Introduction

On-line businesses are beginning to adopt a developmental paradigm where high-level component-based services and semantic web services [20] are becoming sufficiently modular and autonomous to be capable of fulfilling the requirements of other businesses. We use the term "services-based cross-organizational workflow (SCW)" to describe the workflow interaction that occurs when one business incorporates the services of another within its own processes (also described as business-to-business (B2B) [3]). This term is also associated with the idea of a third-party organization that composes the services of multiple businesses (also described as virtual enterprise [9]). Though there are many other related projects that define cross-organizational workflow, we further distinguish our work by defining an architecture that uses the autonomy of agent technologies.

In our work, the SCW environment incorporates the interoperability of general services, using the web services-oriented paradigm or using local invocation. In particular, this environment contains general services, web-based services, component-

oriented services, or invocation-based services. In Figure 1, an example is given of a SCW environment for multiple travel-related businesses. The initiating business is the travel agency company. The Travel Agency has internal services for managing customers' accounts and credit card numbers. However, the travel agency uses other third-party vendors to realize the hotel reservation and car rental reservations. The Hotel Reservation and Car Rental companies register their offerings as web services in a distributed registry, such as a UDDI registry [18]. The Travel Agency uses these registry services as a part of its internal workflow. In addition, the Travel Agency has a partnership with an on-line publishing company that publishes the finalized itineraries. In this case, the travel agency has a static connection with the partner organization and is able to access services directly over a shared network connection.

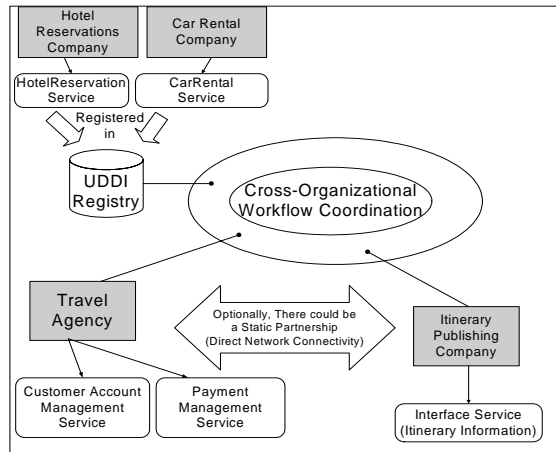


Figure 1. The SCW Environment.

In the event that the Travel Agency has automated services to collect customer requests, local developers may desire to set up a SCW environment to automatically respond to these customer requests. To support this scenario and possibly other business-oriented routines, there is the need for a framework that supports some process and methodology for workflow-oriented service specification. The workflow developers should be able to specify the process sequence of the local and distributed services. In addition, the message exchange must be specified. The specification methodology must contain support for non-functional concerns. The combination of all of these specifications should be adequate to configure the SCW framework.

Agents in this environment are characterized as having weak agency [14], which means that they have proactive and reactive characteristics as well as knowledge of their environment. In this work, we describe how an agent-based architecture can be used for the realization of this framework. In addition, we suggest the use of standard software engineering processes and languages for the specification of the aforementioned workflow processes and control. In Section 2, there is a discussion of related research in this area and, in Section 3, we detail an approach toward the realization of the SCW

environment. In Section 4, we introduce a new approach to modeling agents for this environment. Next, there is a discussion of the software developmental process for developing agent systems in this context. Finally, we discuss our experiences using these approaches.

2. Related Research

There are other projects that also investigate the workflow composition of the services. Casati [6] in the eFlow environment uses flowchart approaches to specify the workflow composition of services. Then, both the workflow and individual services are specified in an XML format. Benatallah [2][22] conducts research that uses Unified Modeling Language (UML) [11] statecharts and agent-oriented methods for declarative peer-to-peer service composition. The research does not specifically claim to handle workflow, but the process-oriented specification is related to workflow. In addition, there is an architecture and object-oriented method for describing the process enactment. Benatallah's work is the most similar to our research and a comparison of the two approaches is given in Section 8. Previous research [12] has also described different approaches for inter-agent communication in multi-agent systems.

There are also projects that concentrate specifically on agent theories for workflow enactment of services. Helal [9] uses an agent architecture for workflow enactment with consideration of services. Chen and Griss [7] also consider the use of agents for workflow with semi-structured specification languages. Finally, Singh [17] discusses the workflow composition of services as a community of services. The major emphasis in this work is an approach to the discovery of services. In the following sections, the WARP approach will be defined and compared to the aforementioned related projects. In the Discussion section, the innovations of this approach are discussed in detail.

3. Overview of the WARP Approach

The Workflow Automation for Agent-Based Reflective Processes (WARP) is an approach for realizing the SCW environment, which builds on previous research. The WARP architecture is divided into two layers, 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, which is used as the agents' knowledge base. 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 a centralized event 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 nonfunctional 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/specifications from a workflow designer as input. The SMAs discover available services and provides service representations to the GWMA. This discovery can occur reflectively for local services or from a UDDI registry for distributed services. The GWMA accepts both of these inputs and writes the workflow policy to the centralized database. The GWMA then configures and deploys WMAs to play certain specialized 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 architecture is shown in Figure 2.

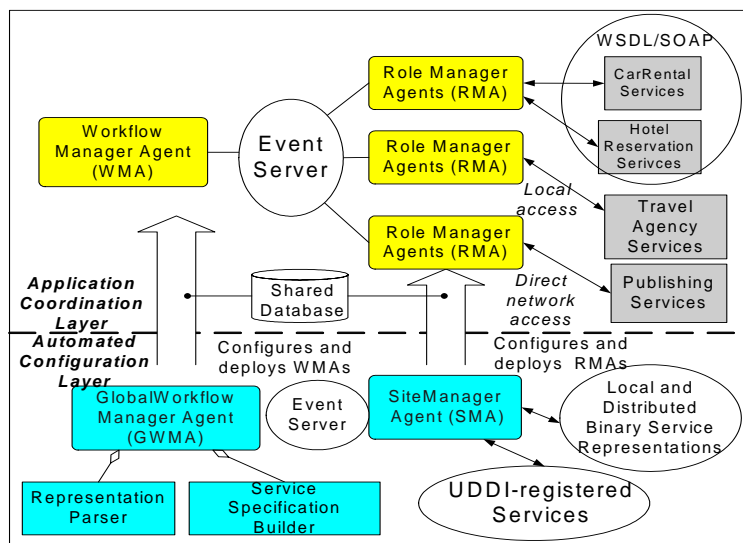


Figure 2. The WARP Architecture.

4. Agent-based SCW Software Process

Software design and configuration in the WARP environment consists of five main steps as illustrated in Figure 3. In the first step, Site Manager Agents are deployed locally or with access to distributed services that are required for the cross-organizational workflow composition. This discovery can occur on services in UDDI registry or also in local component-based service registries. These SMAs search for relevant services, and, in the second step, save the service characteristics in the service-oriented data model.

Also in the second step, with help from the Global Workflow Manager Agent, these service characteristics are captured in WARP models. In the third step, a workflow designer accesses the available services as *Service Representation Views* (to be discussed in greater detail in Section 5). The workflow designer then creates the cross-organizational process models. Once the process modeling is complete, in the fourth step, the GWMA captures the WARP models and extracts the raw information. This raw process information is stored in the process-oriented data model. Consequently, an integrated data model of both service and process data models is ready for agent access. In the final step, application-layer agents (Workflow Manager Agents and Role Manager Agents) access the integrated data model and configured themselves for workflow enactment in the SCW environment. The focus of this paper is on the modeling process; however in other work the agent self-configuration operations are presented in detail [4].

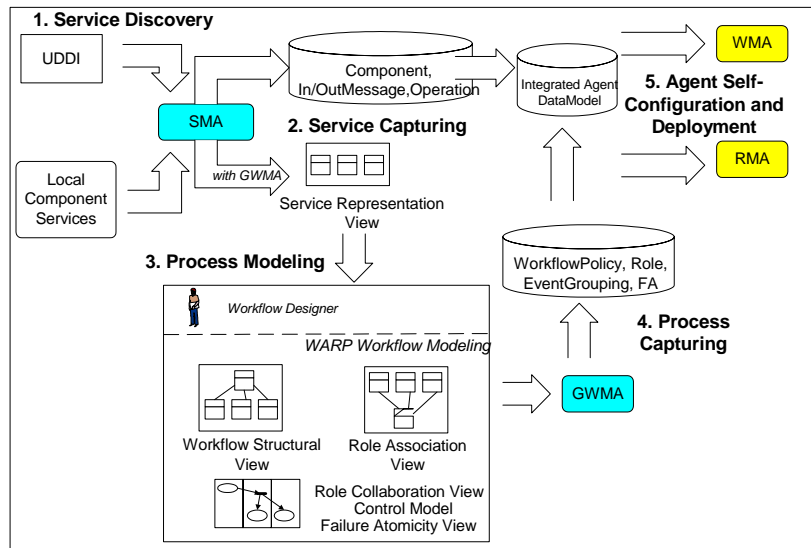


Figure 3. An Agent-Integrated Software Development Process for the SCW Environment.

In the following section, each of the five steps in the WARP development process is described in terms of the supporting object-oriented modeling approach. This is followed in Section 6.0 by a concrete example of the object-oriented modeling approach based on the travel agency example discussed in Section 1.

5. Object-Oriented Modeling for the SCW Environment

The WARP developmental process is an approach that combines human-based and agent-based modeling to gather information that can be used by agents to manage the SCW operation. In defining the steps for the WARP development process, each of the underlying models are defined and discussed in this section.

5.1 Service Discovery and Service Capturing

Similar to the work of Benatallah [2] and Singh [17] in the area of service discovery, we adopt the ideas of elementary services and service communities. Elementary services, in the context of the WARP approach, are atomic component-based services, invocation-based services, and/or web services. In WARP, agents characterize these services and store them in an agent-accessible repository for later composition. A service community is the workflow composition of elementary services. In WARP, this service community is a virtual community because services are distributed. Accessible to the agents is a data repository that maintains the information that defines the workflow-oriented composition specifications that make up this virtual community, as described further in Section 5.2. Site Manager Agents (SMA) are responsible for the automated capturing of service characteristics in the WARP environment. The SMAs have two basic functions for gathering services, registry access and reflection. In registry access (UDDI), SMAs use the access methods provided by the registry. These access methods are relatively straightforward, such as the `find_service` and `get_serviceDetail` methods in the UDDI specification [18].

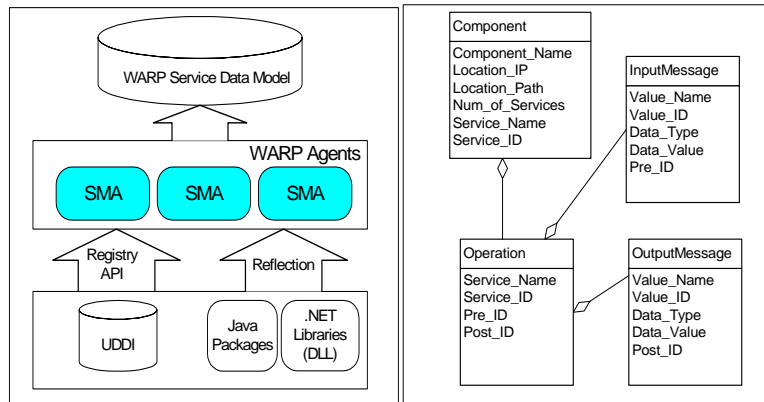


Figure 4a. SMA Population Process **Figure 4b.** WARP Service Data Model

The other function of the SMA is to gather service characteristics directly from the binary representations of the services. The SMA makes use of reflective architectures to discover the operational semantics of certain services. Reflection can be defined as a programming approach that has both a base language as well as a meta-language that describes the base language. Two such reflective approaches for component-service development are the JavaBean API and .NET. Such reflective approaches allow developers to determine the characteristics of previously deployed software. The act of determining these characteristics at run-time is called introspection. The SMAs are able to reflectively gather operations, pre and post conditions, directly from the previously compiled services, without the requirement of having initial source code or specifications, as described in previous work [4]. An overview of the automated population procedure

is illustrated in Figure 4a, in which the SMAs operate on an object-oriented data model of service information, which is gathered by the aforementioned functions. This data model (Figure 4b) consists of four entities, Component, Operation, InputMessage, and OutputMessage. The Component entity defines the components that are available for composition. A component is an independent software artifact. Multiple components may have the same name, so a unique identification (Service_ID) is associated with each. The component entity describes the component-oriented characteristics such as location and network path.

In this research, an individual component may contain multiple elementary services. Thus the Component entity is an aggregation of the Operation entity, which specifies the independent elementary services. Each service consists of a number of Input Messages and Output Messages as defined by the respective entities. In these entities, the Input/Output Messages are further defined by their data type. The Input/Output Messages contain a *Value_Name* field that stores the data. Since message information is heterogeneous across different types of services, to normalize the data we create this agent-adapted field using ontological approaches. Input/Output Messages of multiple services are transformed into independent ontologies, and then a consensus ontology is used to help integrate these messages. This consensus ontology is decomposed into the *Value_Names*. Thus agents incorporate the integrated knowledge. This approach is presented in other work [21]. Finally, the unique identification numbers are used to distinguish components, services, and input/output messages that may be named similarly.

5.2 Process Modeling

Considering the fact that, in previous steps, elementary services have been discovered, captured, and stored by the SMAs, the next step in the WARP approach is where humans intervene to model the workflow composition of these services. A service community can be defined as a repository of these compositions. In the specification of this service community, the WARP approach incorporates industry-standard modeling approaches, in particular UML. In this section, a subset of the workflow modeling semantics is described in detail.

The workflow language here follows workflow terminology used commonly by researchers. 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 one task)
- An *actor* or resource is a person /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.

The WARP approach separates the semantic modeling of the workflow from the specification of services that implement the model. The *task*, *role*, *process*, and *workflow model* terms are used for workflow process specification. However, the terms, *service*, *actor*, and *workflow instance*, specify implementation-oriented information. This is not a new approach, but one that is necessary to ensure that the services and the workflow modeling can evolve separately.

In the WARP approach, workflow processes are specified using UML activity diagrams and class diagrams. We introduce a new approach to modeling workflow in which workflow processes are specified using multiple views, which use different UML diagrams. The advantage of this approach is that it promotes the *separation of concerns* in workflow specification, thereby allowing greater and more effective specificity. In addition, multiple agents can be deployed to implement the workflow process with respect to the individual concerns. Currently, this approach has been investigated using the basic *workflow patterns* as defined by van der Aalst [19]. These patterns include *normal sequence*, *parallel split*, *synchronization*, *exclusive choice*, and *simple merge*. The semantics of these patterns is similar to that provided in recent semantic web services-oriented specifications such as DAML-S, OWL-S and industry web services standards such as WSFL, BPEL4WS, and BPML [16].

There are three major concerns that are specified in WARP-based models: structural, dynamic, and nonfunctional concerns. The structural concerns deals with specification of workflow roles and how those roles are associated with specific workflow processes. From an implementation perspective, the description of underlying services must be considered in addition to their binding to workflow roles. The dynamic concerns incorporate control flow and message flow for normal workflow operation. Nonfunctional concerns consider such things as performance, atomicity, and error-handling. Nonfunctional concerns tend to be peripheral concerns important to the operation of the workflow. We adopt two groupings as specified by Kamath [15], *failure atomicity* and *execution atomicity*. In the failure atomicity grouping, models must define sequence actions that must take place when errors occur. The execution atomicity grouping includes specification of services and groups of services that are incompatible in the same workflow instance or across instances.

Using the WARP approach, the aforementioned three workflow concerns can be modeled using multiple models and views in UML, as summarized in Figure 5. The two central models are the Control Model and Role Collaboration Model. These models are described using control-based and information-based activity diagrams. One distinguishing feature of the WARP approach is the separation of control flow and message flow into two different activity diagrams. The other WARP views are the *Service Representation View*, the *Role Association View*, the *Workflow Structural View*, the *Failure Atomicity Views*, and *Execution Atomicity Views*. Each of these views is described in greater detail in the following sections.

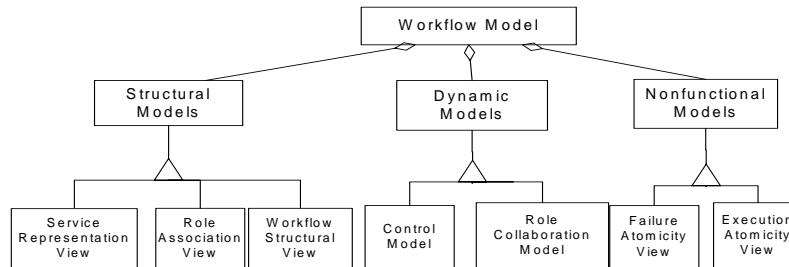


Figure 5. Overview of WARP Models.

5.2.1 Service Representation View

Since services are captured autonomously by the SMAs, there must be some representation such that the available services can be presented to the human workflow modeler. In the WARP approach, visual representations are presented to the human designer prior to modeling. The Service Representation View (Figure 6) allows the GWMA to represent the services available to be modeled in context of the organization housing them. This view is represented in a UML class diagram, in which the class name represents a unique identifier of the organization and class operations are used to depict the independent services available within that organization. Service composition can occur independently of organizational boundaries; however workflow modelers typically need knowledge of available organizations. In addition, this organizational name is later associated with the routing to the services.

5.2.2 Role Association View

The Role Association View (Figure 6) allows a modeler to define workflow roles based on the distributed services that the particular role will encapsulate. This view is also illustrated in a class diagram. This view builds on the Service Representation View such that additional classes are added to represent each of the roles. These classes are then associated to the service-based classes. An unnamed association assumes that all services from that organization are used or available to that role. However, by listing specific service names as the association, a subset of services from the organization can be explicitly designated.

5.2.3 Workflow Structural View

The Workflow Structural View (Figure 6) allows a modeler to specify which roles will be enacted in a particular workflow process. The workflow structural view only provides *a composition of the workflow* as the actual sequence of services is defined in the dynamic models discussed in Section 5.2.4. A class diagram is used that incorporates the workflow roles specified in the Role Association View. The three views provided by the structural models, as shown in Figure 6, complement each other.

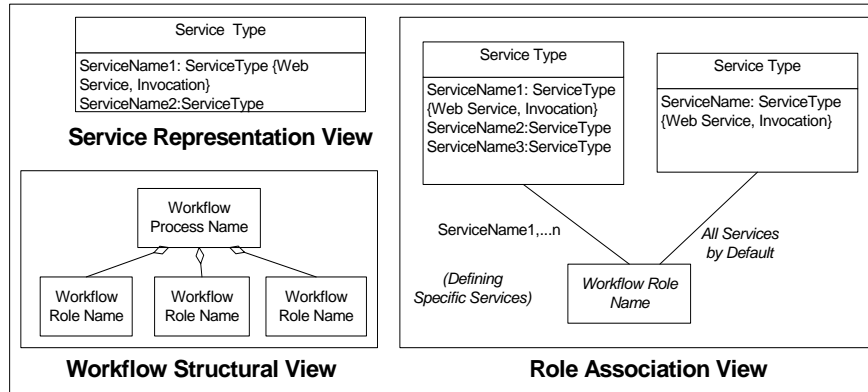


Figure 6. WARP Object-Oriented Structural Models.

5.2.4 Dynamic and Nonfunctional Models

The semantics for the Control Model and the Role Collaboration Model follow closely to related work using activity diagrams for workflow [8]. Each role is illustrated with a new UML swim lane, as illustrated in Figure 7. Each time a role executes a specific service an activity state (oval) is placed in the swim lane. The major difference that distinguishes the WARP approach is the use of the Control Model as an activity diagram that describes the sequence of actions, in addition to the Role Collaboration Model that describes the exchange of messages. In the Control Model, standard fork and join notations are used and the transitions are illustrated with solid arrows. In the Role Collaboration Model, the dotted arrow notation is used between messages. Class notations are used between the services to illustrate the individual messages. The class name represents a message as defined by the modeler. The attributes of this class are a list of the Value_Names, which act as the set or subset of postcondition messages of one service that serve as the set of precondition messages to the subsequent service.

The workflow designer may also model common nonfunctional concerns. The first concern is the assurance of failure atomicity or recovery, when some domain-related problem occurs. The Failure Atomicity Views consists of a Control Model and the Role Collaboration Model. The major difference is that default values are stipulated with the Value_Name attributes. In addition, agents can parse Failure Atomicity Views that mix control flow and message flow in one diagram. This feature was allowed because the Failure Atomicity Views tend to consist mostly of control flows after the initial exception is realized.

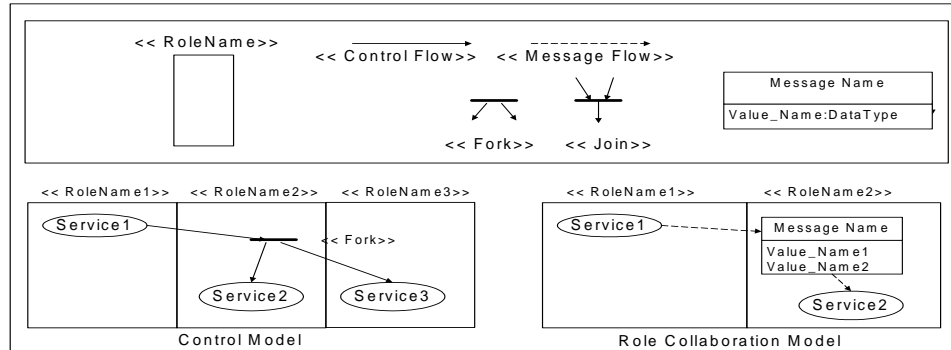


Figure 7. WARP Dynamic Models.

Agents in the WARP architecture monitor messages for the existence of these exception-driven values. The existence of these values serves as a trigger for the execution of the exception-handling-specific workflows (specified in the Failure Atomicity Views). Consequently, for each possible exception, there is a corresponding set of Failure Atomicity Views.

In addition, UML stereotypes are used to specify actions that must be taken to correct services that have been executed in the process of correcting the workflow state. Several common actions represented as stereotypes are <<abort>>, <<roll-back>>, <<roll-back and abort>>, <<re-execute>>, <<roll-back and re-execute>>, <<initiation>>. To illustrate this modeling approach in a concrete example, an incorrectly formatted customer identification scenario is shown in Figure 8. In this scenario, a customer interface web service named *getUserInfo* is executed. When a customer enters an invalid customer_ID the following service (*verify*) populates that field as *not_valid*. This view shows that by using the <<roll-back and re-execute>> stereotype the initial service will be reset and executed again to correct the actions.

5.3 Capturing the Process Models

In the WARP architecture, GWMA's operate on information extracted from the WARP models and views as represented in the data model in Figure 9. The main table for the process specification is the *Workflow Policy* table. Each record in this table defines a single process transition. A transition can be defined as the control flow between the completion of a service or group of services and 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 *FailureAtomicity* table is used to capture the nonfunctional concerns of exception-handling, atomicity, performance, and security. All tables represent the long-term storage in the run-time operation of the workflow.

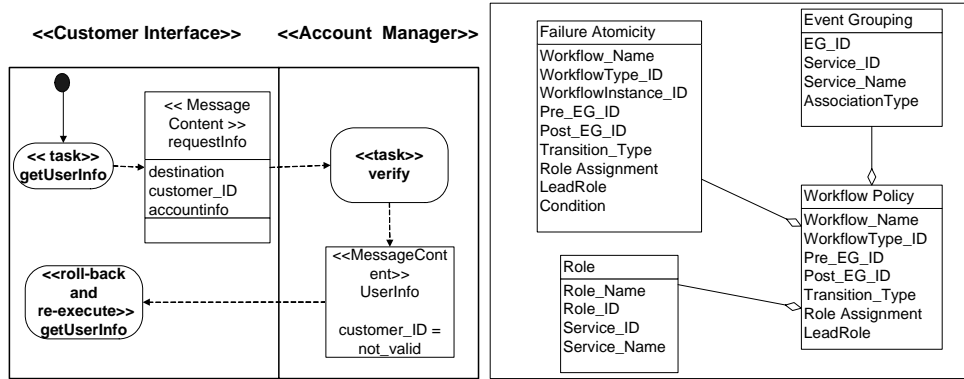


Figure 8. Failure Atomicity View Example. **Figure 9.** WARP Process-Oriented Data Model.

5.4 Agent Self-Configuration and Deployment

The final step in the WARP process is the configuration and deployment of the WARP application layer agents. Workflow Manager Agents assume workflow process responsibilities from the Workflow Policy entity in the process-oriented data repository. Through a sequence of queries, independent Role Manager Agents can be assigned roles as defined in the Role entity. Both WMAs and RMAs can automatically self-configure themselves by collecting information from the process repository via a sequence of general data queries. Since WMAs and RMAs are event-based, they register for events, such as service completion events, which serve as the stimuli for their actions. The focus of this paper is on modeling; more details on the agent operation are given in [4].

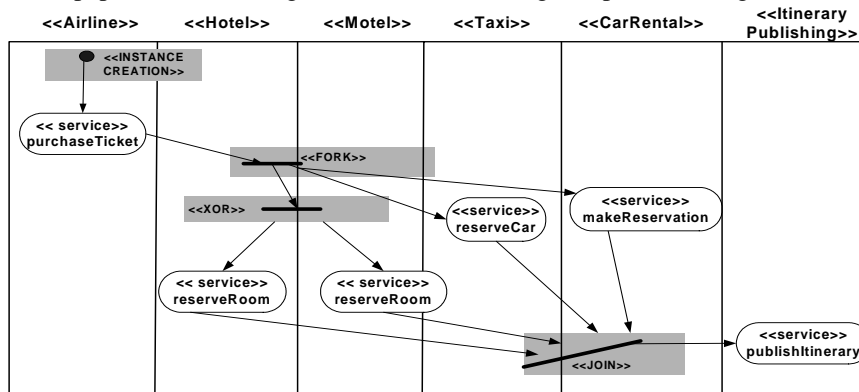


Figure 10. WARP Control Model for the Travel Agency Process.

6. Composition in the Travel Domain: A Workflow Modeling Example

The use of the WARP development process is best demonstrated using a concrete example. Though a full travel agency process, as shown in Figure 10, can be modeled using this approach, in the consideration of space, only a subset can be presented. In this section, three of services from the travel agency domain are defined and modeled for workflow composition with respect to current web services-oriented specifications.

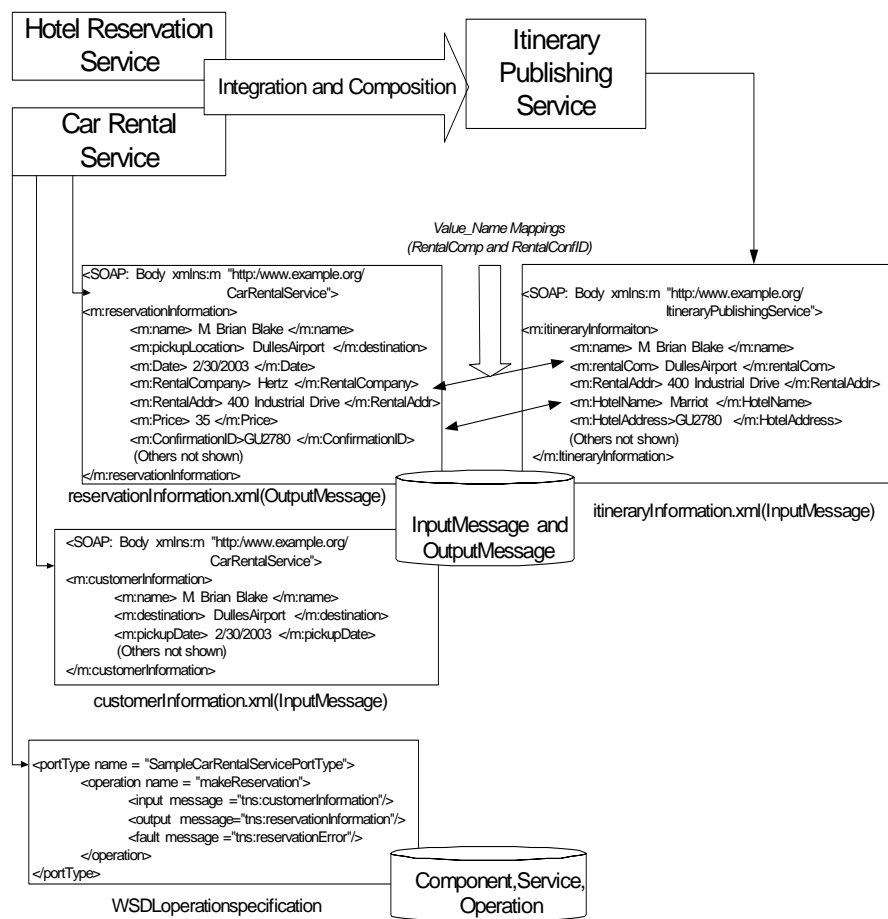


Figure 11. Sample Service Representation for Travel Agency Domain.

6.1 Concrete Service Information and Mapping to WARP Data Entities

The travel agency example considers a car rental service, a hotel reservation service, and an itinerary publishing service. In Figure 11, we show the car rental and hotel reservation services to be composed and integrated with the itinerary publishing service. The input and output messages are represented using sample SOAP messages and the service itself is specified using WSDL notations. These are basic examples for illustration purposes as real services would consist of more information. As discussed in Section 5.1, an agent-generated *Value_Name* is used as a key to describe the mapping between two like elements that are not named identically (such as ConfirmationID and RentalID). Also illustrated are the WARP data entities that relate to the particular input and output SOAP messages and the service-oriented WSDL notation.

In Figure 12a, there is a control model that shows that the car rental and hotel reservation services will be executed concurrently and the output of both services will be used to execute the itinerary publishing service. Also in Figure 12b, the class notations show the subset of information that will be transferred between the services. In these class representations, the value_names are used as opposed to the actual element names from service messages. Agents incorporate the knowledge to map similar fields with different cross-organizational naming. Information is extracted from both of these models to further populate the WARP data entities to later serve as a knowledge base for the WARP agents.

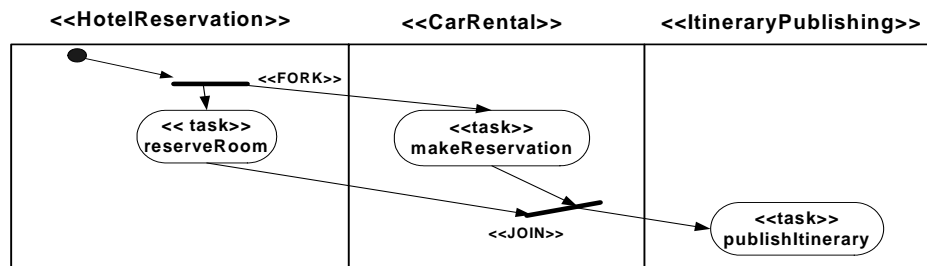


Figure 12a. Control Model for the Travel Agency Domain

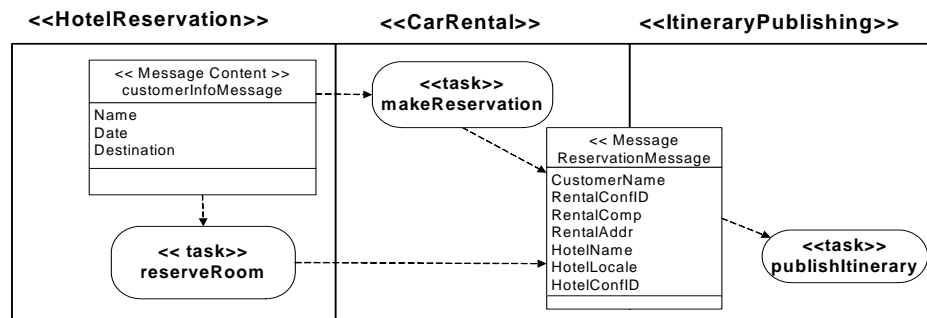


Figure 12b. Role Collaboration Model for the Travel Agency Domain

7. The WARP Prototype

To evaluate the WARP architecture, a WARP prototype and experiment was developed. In Figure 13, the upper portion illustrates the high-level software design of the agent framework and lower portion illustrates the operational environment. For the initial experimentation, the prototype used only invocation-based services. Java Bean components were chosen for the implementation of component-based services. Using Java Beans, Java introspection was used to simulate UDDI calls; invocation over the Java's RMI registry among multiple Windows 2000 workstations was used to simulate distributed services. Event-based communication was realized using JavaSpaces technology, which additionally had seamless integration to Java Beans. The integrated data model is replicated on all machines using Oracle8iLite, a relational database management system that is operational in the personal computer environment. Java Bean services were registered both on Java's RMI registry and given web accessibility using the Tomcat web server.

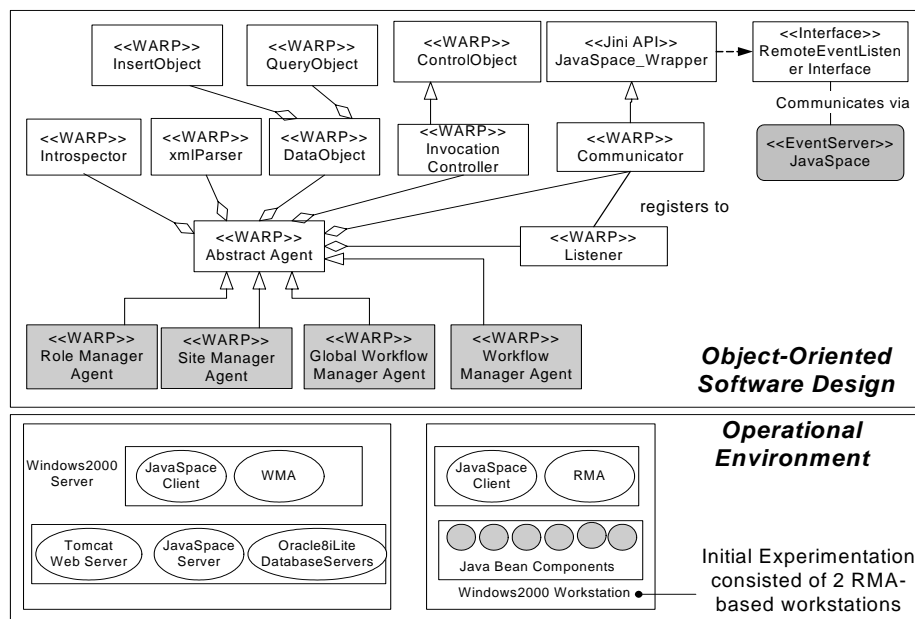


Figure 13. The WARP Prototype Environment and Software Design

The agent architecture was developed in UML using strict object-oriented design practices. The WARP Representations were captured using the Rational Rose developer tool. Using the Rose Extensibility Interface (REI), agents access the representations via the model (.mdl) file created by Rational Rose. In this way, the Rational Rose tool was used as the workflow modeling and capturing tool.

Each type of WARP agent implicitly has all the functions of the *Abstract Agent*. Each agent can transmit events over the event server, register for events, and receive events. These communication 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 or subscription. The implementation of the full WARP prototype is approximately 20,000 lines of program code and scripting.

Important experimentation was performed on the prototype to determine the efficiency of three modes of operation. Since the WARP approach consists of a centralized database, the system can work in three modes of operation. The first mode is configuration of agents at system initialization for both service availability and business process schema. Another mode is the reconfiguration at the beginning of each process. The final mode is re-configuration at the completion of each workflow step (service completion). For the three modes, the overhead in the WARP prototype was approximately 11%, 14%, and 25%, respectively. These figures demonstrate the additional overhead when compared with a completely hard-coded system. These numerical figures are promising with respect to reconfigurable systems that use technologies such as CORBA or COM+. Further details are given in [4].

8. Discussion

The first major contribution of this work is a reusable agent-based architecture and software process that supports the composition of distributed services. This is one of few projects that develop approaches using industry standard modeling notations such as UML [11]. Unlike related work in this area [1][8], we introduce the concept of multiple UML views to separate concerns in the modeling of the agent system that supports this SCW environment. By using multiple views and software agents to extract the operational data, service evolution and process evolution can occur independently. Furthermore, a workflow designer can control this change by visually modeling the workflow design using COTS object-oriented software engineering tools. This approach has been applied to a WARP prototype [5] where several successful scenarios were implemented with encouraging performance results. Consistency checking between these views is an important area of future work, which has been addressed by the authors in related research [10].

Earlier work by Casati and Benatallah [2][6] consider complex workflow-oriented interactions with visual and text-based specification approaches. Both adopt approaches to the service composition but have fairly rudimentary, non-agent-oriented interactions in the operation of their respective internal architectures. The work of Helal [13], Chen [7], and Singh [17] all concentrate on the agent-oriented interactions for service composition. However, the service composition specification languages used to program their agents are neither visual nor do they seem capable of handling the complex workflow interactions that are supported in our work or the support given in the work of Casati and Benatallah.

The WARP approach follows the state of the art closely with the layered approach to proxy agents, coordination agents, and specification language. In addition, our approach follows the state of the art of other related research (as Casati and Benatallah) by incorporating the use of both a visual language and a corresponding textual language for specification. We extend the approaches of Casati and Benatallah by considering modeling and agent support for both the complex workflow-oriented interactions among the services and the complex interactions of agents internal to our architecture. Prior related work has concentrated on either one or the other. Another innovation in the WARP approach is the workflow based software developmental process in the creation of these composite systems, which is consistent with industry-standard software engineering lifecycles. In fact, the use of industry-standard modeling languages, such as UML, makes the integration into industrial processes more realizable.

9. Conclusions and Future Work

This paper has described an object-oriented UML-based developmental process for modeling workflow-based service composition and incorporating a distributed agent architecture. Since input/output messages can be represented in heterogeneous formats, such as plain text, concrete objects, or XML, modeling approaches and agent support have a challenge to create operational patterns to deal with these formats. Another problem is support for heterogeneous methods of error-handling. The approach taken in this work is to create new models for each error-handling case. We believe these cases could be more efficiently integrated into the standard operational models. Future research would investigate extending this approach with advanced workflow patterns and interactions. Further research is also to investigate agent-based negotiation and performance modeling of the Quality of Service scenarios.

10. References

- [1] Amyot, D., Buhr, R.J.A., Gray, T., and Logrippo, L. Use Case Maps for the Capture and Validation of Distributed Systems Requirements. In: RE'99, Fourth IEEE International Symposium on Requirements Engineering, Limerick, Ireland, June 1999, 44-53.
- [2] Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A. Declarative composition and peer-to-peer provisioning of dynamic web services. ICDE 2002, San Jose, CA Feb. 2002
- [3] Blake, M.B. "B2B Electronic Commerce: Where Do Agents Fit In?" AAAI-2002 Workshop on Agent-Based Approaches to B2B Electronic Commerce, Edmonton Canada, August 2002
- [4] Blake, M.B. *Agent-based Workflow Modeling for Distributed Component Configuration and Coordination*, Ph.D Dissertation, George Mason University, 2000 accessible at: <http://www.cs.georgetown.edu/~blakeb/pubs/diss.zip>
- [5] Blake, M.B. and Gomaa, H. " Object-Oriented Modeling Approaches to Agent-Based Cross-Organizational Workflow ", Workshop on Software Engineering for Large-Scale Multi-Agent Systems in conjunction with the International Conference on Software Engineering (ICSE2003), Portland, Oregon 2003

- [6] Casati, F., Jin, L., Ilnicki, S. and Shan, M.C. An Open, Flexible, and Configurable System for Service Composition. HPL technical report HPL-2000-41.
- [7] Chen, Q. Dayal, U., Hsu, M., and Griss, M.L.: Dynamic-Agents, Workflow and XML for E-Commerce Automation. EC-Web 2000: 314-323, London, UK
- [8] Dumas, M. and ter Hofstede, A.H.M., "UML Activity Diagrams as Workflow Specification Language", [Lecture Notes in Computer Science](#), Vol. 2185 Springer 2001
- [9] Gijsen, J.W.J., Szirbik, N.B., and Wagner, G. Agent Technologies for Virtual Enterprises in the One-of-a-Kind-Production Industry, International Journal of Electronic Commerce, Vol. 7, No.1 pp 9-34, October 2002
- [10] Gomaa, H. and Shin, M.E. "Multiple-View Meta-Modeling of Software Product Lines", Proceedings IEEE International Conference on the Engineering of Complex Computer Systems, Greenbelt, MD, December 2002
- [11] Gomaa, H. Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, Boston, MA 2000
- [12] Gomaa, H., "Inter-Agent Communication in Cooperative Information Agent-Based Systems" in Cooperative Information Agents III, Vol. 1652, Lecture Notes in Artificial Intelligence, Berlin, Springer-Verlag, 1999
- [13] Helal, A., Wang, A.M., Jagatheesan, A., and Krithivasan, R. "Brokering Based Self Organizing E-Service Communities". In 5th International Symposium on Autonomous Decentralized Systems (ISADS) March 26-28, 2001 Dallas, Texas
- [14] Jennings, N.R., Sycara, K. P., and Wooldridge, M., "A Roadmap of Agent Research and Development", Journal of Autonomous Agents and Multi-Agent Systems. 1(1), pp 7-36. 1998.
- [15] Kamath, M. and Ramamrithan, K., "Correctness Issues in Workflow Management," Distributed Systems Engineering Journal 3(4): 213-221, December 1996
- [16] Lara, R., Lausen, L., Arroyo, S., Bruijn, J., and Fensel, D. Semantic Web Services: Descriptions, Requirements and Current Technologies" International Workshop on Agents and Semantic Web Services (ICEC2003), Pittsburgh, PA, October 2003
- [17] Singh, M.P., Yu, B., and Venkatraman, M.: Community-based service location. CACM 44(4): 49-54 (2001)
- [18] UDDI Specification 2.04 (2002) : <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf>
- [19] Van der Aalst, W.M.P. "Don't go with the flow: Web Services composition standards exposed", IEEE Intelligent, February 2003
- [20] Web Services (2002) <http://www.w3.org/2002/ws/desc/>
- [21] Williams, A.B., Padmanabhan, A., and Blake, M.B., "Local Consensus Ontologies for B2B-Oriented Service Composition", Proceedings of the AAMAS2003, July 2003
- [22] Zeng, L., Ngu, A., Benatallah, B., and O'Dell, M. An Agent-based Approach for Supporting Inter-enterprise Workflows. Proceedings of the 12th Australasian Database Conference, IEEE Society, Gold Coast, Australia, February 2001.