Available online at www.sciencedirect.com

SCIENCE DIRECT®

Decision Support Systems xx (2004) xxx–xxx

Decision Support Systems

www.elsevier.com/locate/dsw

Agent-oriented compositional approaches to services-based cross-organizational workflow

M. Brian Blake[a,*,1], Hassan Gomaa[b]

[a] *Georgetown University, Room 234, Reiss Science Building, 37th and O Street, NW, Washington, DC 20057, USA*
[b] *George Mason University, 4400 University Drive, Mail Stop 4A4, Fairfax, VA 22030-4444, USA*

Abstract

With the sophistication and maturity of distributed component-based services and semantic web services, the idea of specification-driven service composition is becoming a reality. One such approach is workflow composition of services that span multiple, distributed web-accessible locations. Given the dynamic nature of this domain, the adaptation of software agents represents a possible solution for the composition and enactment of cross-organizational services. This paper details design aspects of an architecture that would support this evolvable service-based workflow composition. The internal coordination and control aspects of such an architecture is addressed. These agent developmental processes are aligned with industry-standard software engineering processes.

Keywords: Agent architectures; Workflow modeling; Coordination; UML; Web services

1. Introduction

Online businesses are beginning to adopt a developmental paradigm where high-level component-based services and semantic web services [37] 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)[5], 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)). This term is sometimes associated with the idea of a third-party organization that composes the services of multiple businesses, which is also described as *virtual enterprise* [15].

In general, the major problems in this domain relate to the *dynamic and distributed nature of the Internet environment*. In this environment, business processes and the underlying services are constantly removed and updated. It is a major problem to create systems that operate with respect to these dynamic conditions. A second major issue is related to the distribution of services. Since services are distributed across physical and geographical boundaries, any solution architecture must support an equivalent degree of distribution.

Although there are several related projects that define solutions to the problems of cross-organiza-

* Corresponding author.
E-mail addresses: blakeb@cs.georgetown.edu (M.B. Blake), hgomaa@gmu.edu (H. Gomaa).
[1] For identification purposes only, this author also has an affiliation with The MITRE Corporation, Center for Advanced Aviation System Development, M/S N420, 7515 Colshire Drive, McLean, VA, 22102-7508, USA.

0167-9236/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2004.04.003

tional workflow (which will be described in more detail in Section 2.3), a distinguishing innovation of this work is the use of the autonomy of agent technologies. In applying agent technologies to the problems related to dynamism and distribution, an agent architecture and design is introduced which places emphasis on the specific roles, responsibilities, and actions of the individual agents. A further contribution of this work is the specification and programming of the control mechanisms internal to the agents. A unique feature introduced here is the integration of this specification approach with current, industry-standard software engineering processes and methodologies. Considering the dynamism of the Internet environment, these specification-driven approaches are essential for the dynamic reconfiguration that results from distributed service and process changes.

The paper proceeds in Section 2 with an overview and motivation of the cross-organizational workflow domain with respect to the integration of distributed services. In Section 3, the *Workflow Automation through Agent-based Reflective Processes* (WARP) is introduced to support the SCW domain. In Section 4, there is a discussion of the integration of independent services and, in Section 5, the agent-based modeling approach to support the community of services in the SCW domain. Section 6 contains details of the general agent interaction protocols that support this environment. Finally, Section 7 discusses the WARP prototype and its performance.

2. The SCW environment and web services

The SCW approaches are a natural extension of the related areas of component-based software engineering, in particular component composition. In traditional component composition research [19], components, such as CORBA, COM+, J2EE, Enterprise Java Beans, and .NET, and their interfaces are modeled using formal (text and visual) languages. Consequently, these specifications are evaluated and deployed to support automated component composition. Currently, these components can be specified with web service technologies to facilitate large-scale electronic market interoperability.

2.1. Web service technologies

Components, which are specified with web services technologies, have the capability of being discovered and accessed from distributed locations. These web services are specified with the Web Services Description Language (WSDL) [37] and can be invoked using the Simple Object Access Protocol (SOAP) [31]. Traditional WSDL supports a syntactical form of specifying web services, while the term, *semantic web services* refers to an extended Web of machine-readable information and automated services. The DARPA Agent Markup Language for Services (DAML-S) and the Web Ontology Language (OWL-S) [26] both provide an ontology of services to allow automated support for components such as agents to locate, select, employ, compose, and monitor Web-based services. Distributed registries, such as the Universal Description, Discovery, and Integration (UDDI) [34,35] architectures, advertise the specifications of distributed services universally. In addition, other Extensible Markup Language (XML)-based languages, such as the Web Services Flow Language (WSFL), Business Process Execution for Web Services (BPEL4WS), and the Business Process Modeling Language (BPML), specify the process-based composition of these services as described in Refs. [7,25,32,40]. However, these process languages are specified with a text-based approach that tends to conflict with the visual design notations commonly accepted for software development, in particular the use of the Unified Modeling Language (UML) [6,17]. A main goal of this work, which will be discussed in detail in Section 3, is to support service-based composition using accepted *visual* developmental approaches coupled with agent-based protocols.

2.2. A sample SCW environment

The SCW environment described in this paper incorporates the interoperability of general web services. In Fig. 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

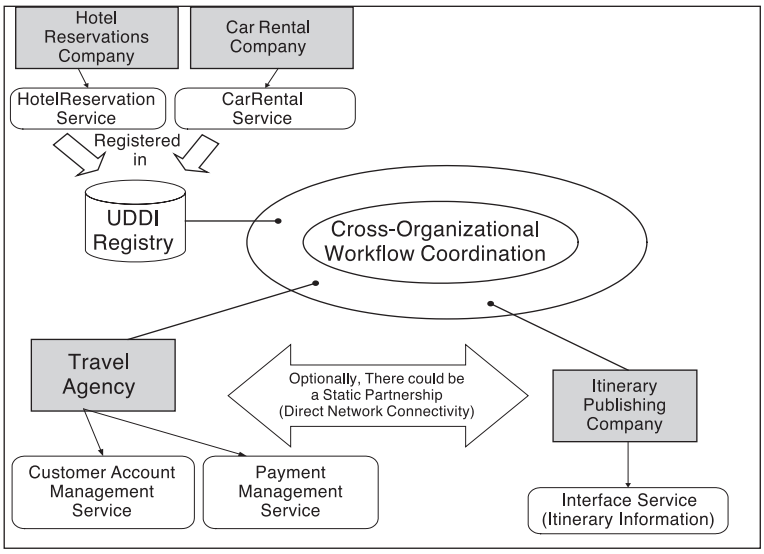

Fig. 1. An example of the SCW environment.

reservations. The Hotel Reservation and Car Rental companies register their offerings as web services in a distributed registry, such as a UDDI registry. 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. Problems occur in this domain when the online companies update or remove their service offerings. This SCW environment requires a framework that supports a methodology for process or workflow-oriented service specification. This framework would allow workflow developers to specify the process sequence and message exchange between local and distributed services. The specification approach must support both functional and nonfunctional concerns. Therefore when the process or services change, the specification can be updated and the supporting architecture automatically reconfigured.

2.3. Related work in the SCW domain

Several related projects directly and indirectly address the dynamism problems associated with the SCW environment. One area of research, which addresses the ability for reconfiguration in such environments, is traditional component composition approaches. Mennie and Pagurek [27] summarize the landscape of service composition and describes a Jini-based architecture for service composition. In this work, they describe an XML-based specification and process to compose services realized as Java Bean components. The CHAIMS project [33] at Stanford University also declares a compilation process and text-based composition language, CLAM [29], for the composition of services. In another approach, Chakraborty et al. [9] define an architecture for service composition in pervasive environments. The motivation for this work, though early in development, is for automated reactive service composition.

Other projects address the problems in the SCW environment, while also considering services as building blocks for workflow and business processes [18]. Casati et al. [8] in the eFlow environment use flowchart approaches to specify the workflow composition of services, with the workflow and individual services both specified in an XML format. Benatallah et al. [1] conduct research that uses UML-based state charts and formal methods for declarative peer-to-peer service composition. This research does not specifically claim to handle workflow, but the process-oriented specification is related to workflow. In later work [41], agents are incorporated as well as formal methods for

process enactment. Benatallah et al.'s work is the most similar to our research and a comparison of the two approaches is given in Section 8.

Other research projects use agent theories to address the SCW problems. Helal et al. [20] use an agent architecture for workflow enactment with consideration to web services using SOAP and UDDI-registered services. Chen et al. [10] also consider the use of agents for workflow with semi-structured specification languages. Finally, Singh et al. [30] discuss the workflow composition of services as a community of services, with emphasis on an approach to the discovery of services.

Considering related research in the SCW area, all projects support the idea of multiple layers to address the dynamic nature. In general terms, there must be a configuration layer and a layer where the actual compositional enactment occurs, as illustrated in Fig. 2. Sometimes the configuration layer and enactment layer are further decomposed. In most cases, there is an understood benefit of allowing services to remain on the providers' servers while the composition occurs locally using distributed invocation. Another common approach is the use of some proxy component or agent that represents the services. These proxy components/agents encapsulate the knowledge of service capabilities and how to execute the services. With the use of proxy component/agents to wrap services, there must be a manager/coordination/process component or agent to control the composition. In all cases, there is some language (either text-based or visual) that is used to allow the specification of the composition which is then later compiled or interpreted by the manager or coordination component.

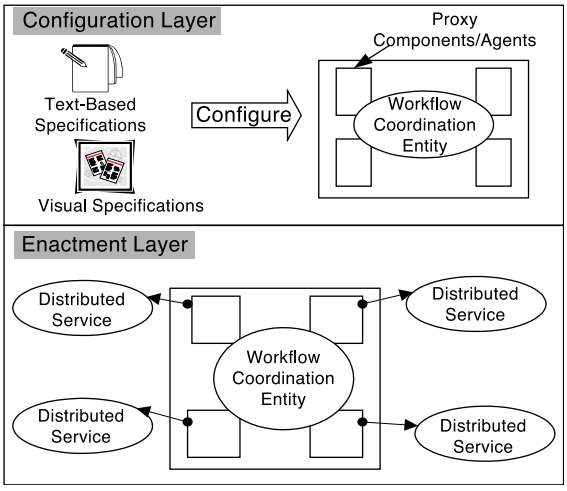

Fig. 2. General state of the art for architectures supporting SCW issues.

3. The WARP approach

Workflow Automation for Agent-Based Reflective Processes (WARP) is an agent-based middleware architecture used to implement the SCW framework. In addition, a standard software engineering process and language supports the specification of the workflow processes and control.

3.1. The WARP architecture

The WARP architecture consists of software agents that are configured to control the workflow operation of distributed services. The WARP architecture is divided into two layers, the *application coordination layer* and the *automated configuration layer*, as shown in Fig. 3. 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 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. RMAs do not directly communicate and collaborate, but the events that they propagate implicitly direct the actions of their peer RMAs. The WMA has similar functionality, but instead registers for overall workflow level events (i.e. workflow

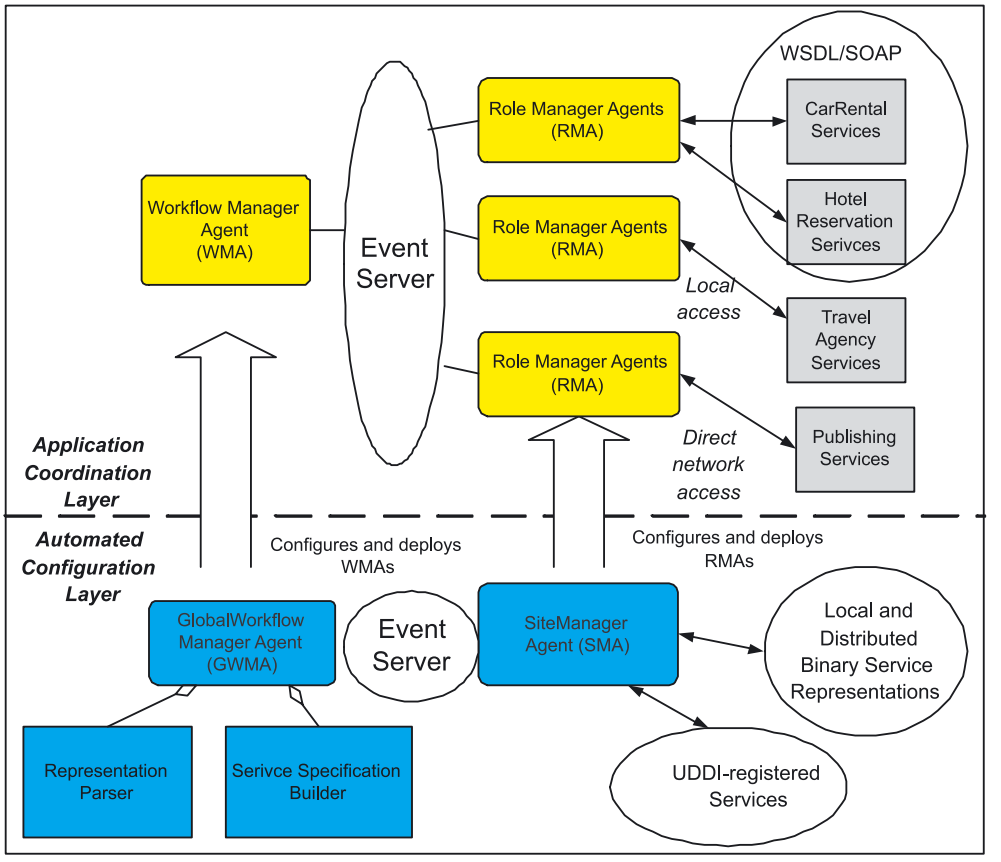

Fig. 3. The WARP architecture.

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 (Fig. 3), agents accept new process specifications and deploy application coordination layer agents with the 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 provide service representations to the GWMAs. This discovery can occur reflectively for local services or from a UDDI registry for distributed services. The GWMAs accept both of these inputs and write the workflow policy to the centralized database. The GWMA then configures and deploys WMAs to play certain workflow-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.

This paper highlights three major areas of the WARP approach which are *data management for distributed services*, *workflow modeling of services*, and *agent interactions for service modeling and composition*. Similar to related work [1,30], 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 Section 4, agents are shown to characterize these services and manage data dealing with composition. In this research, a service community refers to the workflow composition of elementary services. In WARP, this service community is a

virtual community, since services are distributed. Accessible to the agents is a data repository, which contains the information that defines the workflow-oriented composition specifications for this virtual community. A major focus of this paper is the workflow modeling approach, which uses industry-standard software modeling approaches as described in Section 5. In Section 6, there is a discussion of the significance of using agents and the interactions that multiple agents undertake to realize the idea of service composition.

4. Data management for distributed services

With the WARP approach, an elementary service repository is not a repository of the services but a repository of service descriptions. Since services are stored at the providers' locations, there are certain characteristics of the services that must be captured and stored locally before the agent-oriented workflow composition can be executed.

4.1. Elementary services for the WARP environment

There are two elementary service types considered in the WARP approach, invocation-based and web services-oriented. In both cases, these services have a "call-and-return" style of operation. The environment incorporates web services accessible using SOAP protocols in addition to Java-based components and .NET components. Not considered in this work is the area of event-based component services.

Similar to the current state of the art in service specification [13,29,32,39], WARP-mediated services are specified with three parts known as the *operation*, the *input parameters*, and the *returned parameters*. The operation is the identification information of the service, which includes the location and the execution procedure of that service. The input parameters specify the information required for operation and the returned

Fig. 4. Decomposing an elementary service.

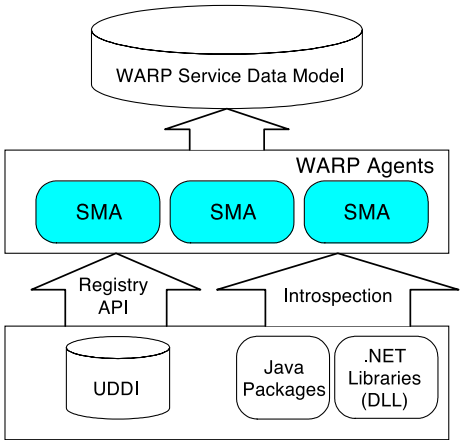

Fig. 5. WARP service population procedure.

parameters consist of the output information. These concepts, which underlie the service specification, are common to web services and traditional invocation-based services. In addition, input and returned parameters may be associated with other system-oriented pre- and post-conditions of the service. An illustration of the parts of the elementary service is shown in Fig. 4.

4.2. Automated population into the agent-accessible data model

Site Manager Agents (SMA) are responsible for the automated capturing of service characteristics and assigning Role Manager Agents (RMA) in the WARP environment. The SMAs have two basic functions for gathering services, registry access and introspection. In registry access (UDDI), SMAs use the access methods provided by the registry, such as *find_service* and *get_serviceDetail* methods in the UDDI specification [34]. In this case, SMAs are required to know the specific service name. For other services, the SMA gathers the service characteristics directly from the binary representations of the services using introspection, which is a reflective capability for determining service characteristics from binary source code at runtime. Using this approach, the SMAs are able to gather operations, pre- and post-conditions directly from the previously compiled services, without the requirement of having initial source code or specifications. An in-depth discussion of introspection is contained in previous work [2]. Fig. 5 shows an overview of the

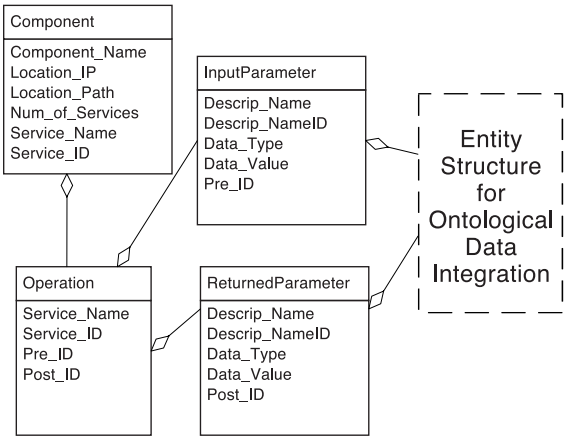

Fig. 6. Service specification data model.

operations of the SMA to populate service specifications into a local repository.

The SMAs populate a data repository consisting four entities, *Component*, *Operation*, *InputParameter*, and *ReturnedParameter*. The services data model is shown in Fig. 6. The Component entity defines the characteristics of the components that are available for composition, such as location and network path. Because an individual component may contain multiple elementary services, the Component entity is an aggregate of the Operation entity, which specifies the independent elementary services. Each service consists of an aggregation of InputParameter and ReturnedParameter. In these entities, the input and returnedParameters are further defined by their data type. Throughout all entities, unique identification numbers are used to distinguish components, services, and input and returnedparameters that may be named similarly. Input and returnedParameters can use ontological approaches to assist in data integration [38].

5. Workflow modeling of services

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 requires human intervention 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 [6,17].

5.1. Workflow terminology in WARP

The workflow language here follows workflow terminology used commonly by researchers, as in Lei and Singh [24]. 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 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.

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 in specification of conditions that are directly related to the workflow process. 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. One simplification made in this work is that a task is directly related to one type of service. Based on this simplification, roles are directly connected to services as opposed to tasks. This does not significantly limit the *modeling approaches*, which are the focus of this paper, since many roles can be modeled for the same service. However in future work, it may be necessary for services of a specific task to dynamically change without changing roles.

5.2. Workflow interaction in the WARP environment

Fundamentally, the design of workflow processes and complex interactions have been investigated in great detail. Van der Aalst [36] maintains a list of workflow interactions called workflow patterns. At the most general level, there are two concepts in workflow that seems to be adopted universally. In specifying workflow, the modeler must be able to show the sequence of control as the workflow executes, in addition to the ability to show how data is passed throughout the process. These two concepts can be referred to as *control flow* and *data flow* or *message flow*. The most common workflow interactions can be characterized in terms of control flow and message flow sequences. However, shortcomings in modeling approaches are discovered when the workflow interactions are complex combinations of both control flow and message flow. This complexity is extended further as modeling must occur across multiple workflow instances. The WARP approach implements the basic workflow control patterns consisting of *normal sequence*, *parallel split*, *synchronization*, *exclusive choice*, and *simple merge* as defined in Ref. [36].

5.3. WARP workflow modeling approaches

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 concern* [23] in workflow specification, thereby reducing complexity. In addition, multiple agents can be deployed to implement the workflow process with respect to individual concerns.

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 aspect, structurally the description of underlying services must be considered in addition to their binding to workflow roles. The dynamic concerns incorporate the control flow and message flow. 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 [22], *failure atomicity* and *execution atomicity*. In the failure atomicity grouping, models must define the sequence of 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 three workflow concerns can be modeled using multiple models and views in UML (predominantly class and activity diagrams). These models and views are summarized in Fig. 7. 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

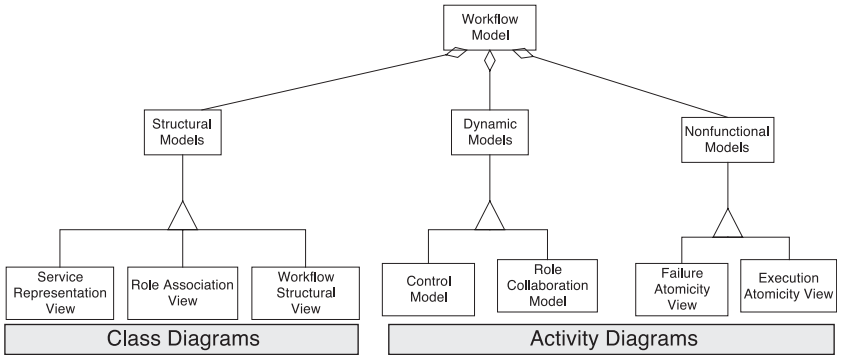

Fig. 7. Summary of WARP workflow models.

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 View*, and *Execution Atomicity View*. In the following subsections, we briefly summarize these views for completeness as additional in-depth discussions have been made in previous work [2].

5.3.1. Structural and dynamic models

WARP structural models are views of the SCW environment representing the physical agents and services involved with respect to the configured workflow processes. The structural views consist of the *Service Representation View*, the *Role Association View*, and the *Workflow Structural View*. These views are constructed using UML class diagrams. Since the focus of this paper is the dynamic process, the reader is directed to related work for further detail [2] on the structural models.

The dynamic models are defined by the *Control Model* and the Role *Collaboration Model*. The semantics for the Control Model and the Role Collaboration Model follow closely to related work using activity diagrams for workflow [11,12]. Each role, as initially specified in the structural models, is illustrated with a new UML *swim lane*, as illustrated in Fig. 8. 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. A message sent by one service to another is modeled by means of a class. The class name represents a message as defined by the modeler. The attributes of this class are a list of the Parameter names, which act as the returned parameter components of one service and the input parameter requirements of the subsequent service.

5.3.2. Nonfunctional models

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 are the same as the Control Model and the Role Collaboration Model. The major difference is that default values are stipulated with the Param_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. Agents in the WARP architecture monitor messages for the existence of these exception-driven values. The existence of these values serves as the trigger for the execution of the

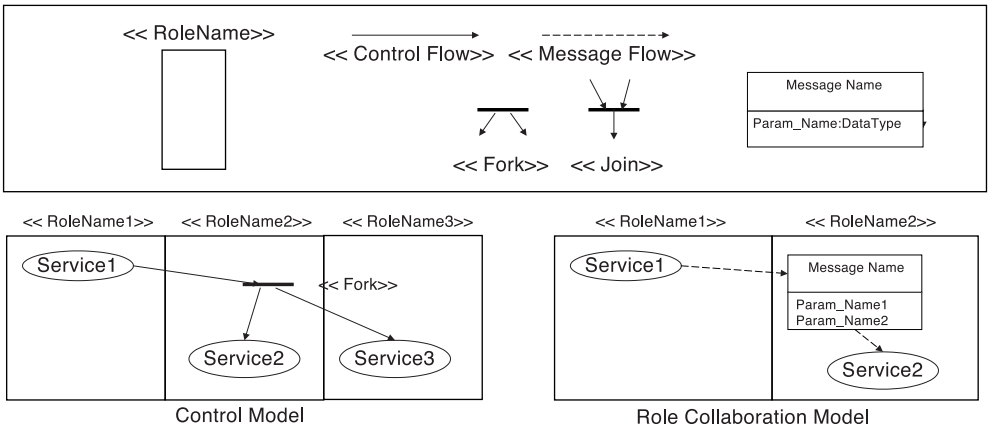

Fig. 8. Dynamic models.

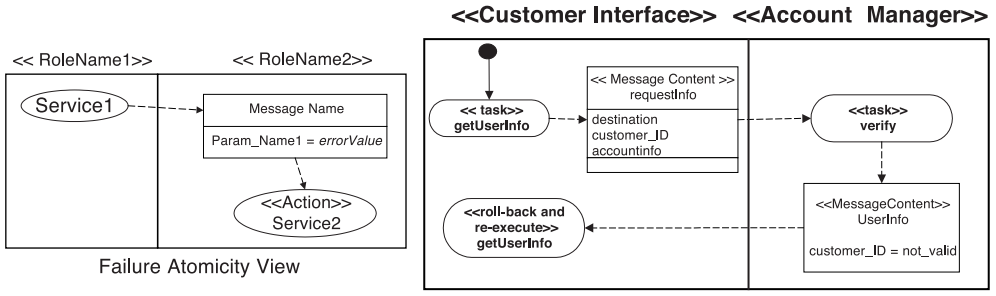

Fig. 9. Failure Atomicity View notations and concrete example.

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≫. In Fig. 9, there are the semantics of the Failure Atomicity View and a concrete example of an incorrectly formatted customer identification scenario is shown.

There was one major assumption made in this approach. This assumption is that services contain basic error-handling functionality. Actions such as roll-back and re-execute have to be supported by the services internally. It is not unrealistic to assume that services will be constructed with internal error correcting capabilities. However, the WARP architecture also requires that these error-correcting actions be implemented in a relatively uniform manner so that agents can invoke them universally. Though web service messages have support for this, this manner of development is not the current state of heterogeneous services developed in different enterprises. This is a weakness in the WARP approach that serves as an avenue for future research.

5.4. Capturing the WARP models

In the WARP architecture, GWMAs operate on information extracted from the WARP models and views as represented by the data model in Fig. 10.

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* tables are 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. The process of parsing the WARP models and views is a systematic process. The data model presented in Fig. 10 maps directly to the various models and views. These views also map to a WARP XML-based text schema. This visual, textual, and database view of the SCW processes is an innovation discussed here and in related work [3].

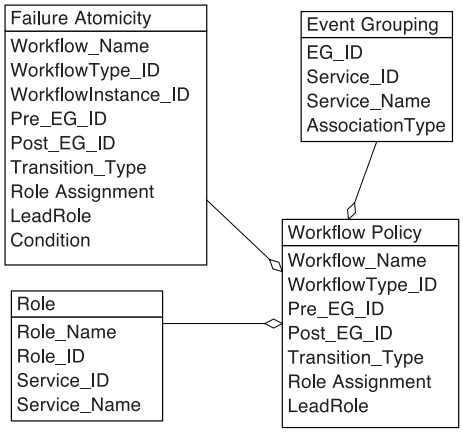

Fig. 10. WARP process-oriented data model.

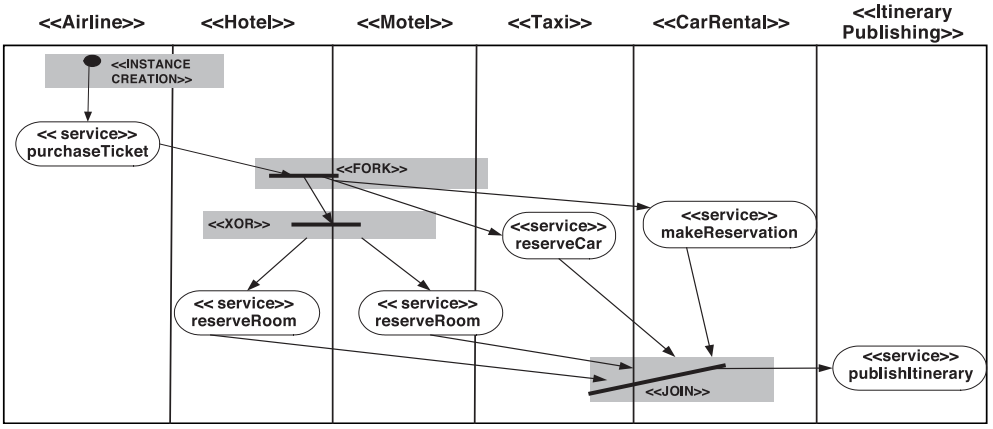

Fig. 11. A control model for the travel agency domain.

5.5. A concrete example of the WARP modeling approach

A concrete example of WARP service modeling is motivated in the travel agency domain, which was first described in Section 2. In Fig. 11, we show the control model for a travel agency scenario for purchasing an airline ticket (purchaseTicket), reserving a taxi (reserveCar), reserving a hotel or motel room (reserveRoom), reserving a rental car (makeReservation), and receiving an email-delivered itinerary (publishItinerary). In this simple scenario, the fork, join, and other workflow-oriented transitions, in particular instance creation and exclusive-or relations are illustrated by the shaded regions. Though the initial studies emphasize fork and join relations, other relations, as discussed Section 5.2, will be investigated in future work.

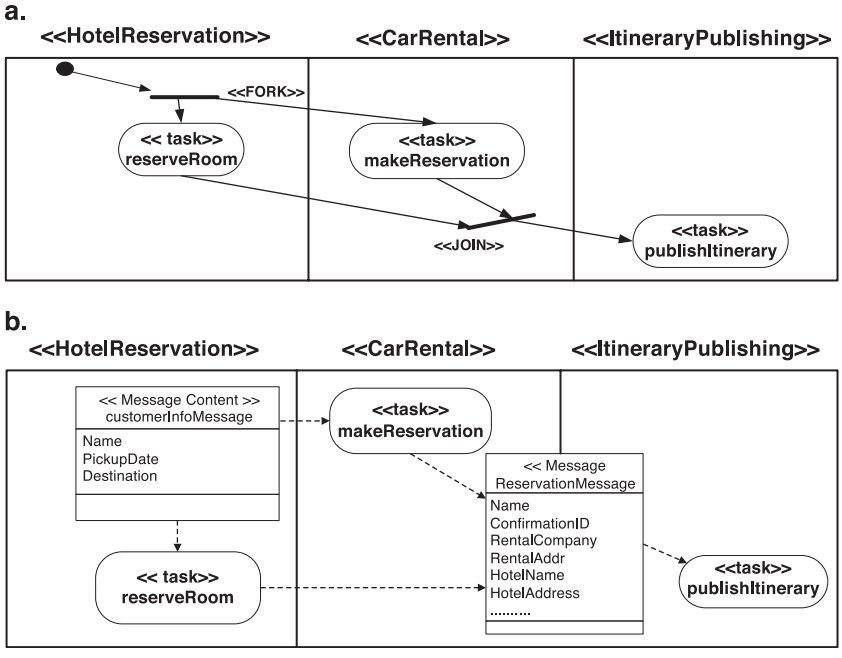

Fig. 12. (a) Control model for the travel agency domain. (b) Role collaboration model for the travel agency domain.

Multiple agents coordinate the composition and enactment of services based on this travel agency scenario. This scenario starts when a user issues a new job with travel preferences. A new instance is created and the first service executed is the purchaseTicket service. This service returns with a flight ticket and times. The flight information serves as the input parameters for the next three services concurrently. There are two reserveRoom services where one service is in a hotel reservation system and the other is in a motel reservation system. The exclusive-or relation specifies that just one of the reservation services will be executed concurrently with services for making an appointment for a taxi (reserveCar) to take the person to and from the airport in the originating city and for reserving a rental car (makeReservation) in the destination city. Once the three concurrent services have completed, then the returned parameters of the services can be used as input to the publishItinerary service.

The relation of the WARP models to an actual web service specification is shown with a subset of the travel agency domain. Considering only the services for reserving a hotel room, making a car rental reservation, and publishing the itinerary, there is the corresponding control model and role collaboration model illustrated in Fig. 12a and b. In Fig. 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 Fig. 12b, the class notations show the subset of information that will be transferred between the services. In these class representations, the attributes represent the information shared between the services. 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.

The models in Fig. 12a and b are constructed from concrete web services and SOAP specifications.

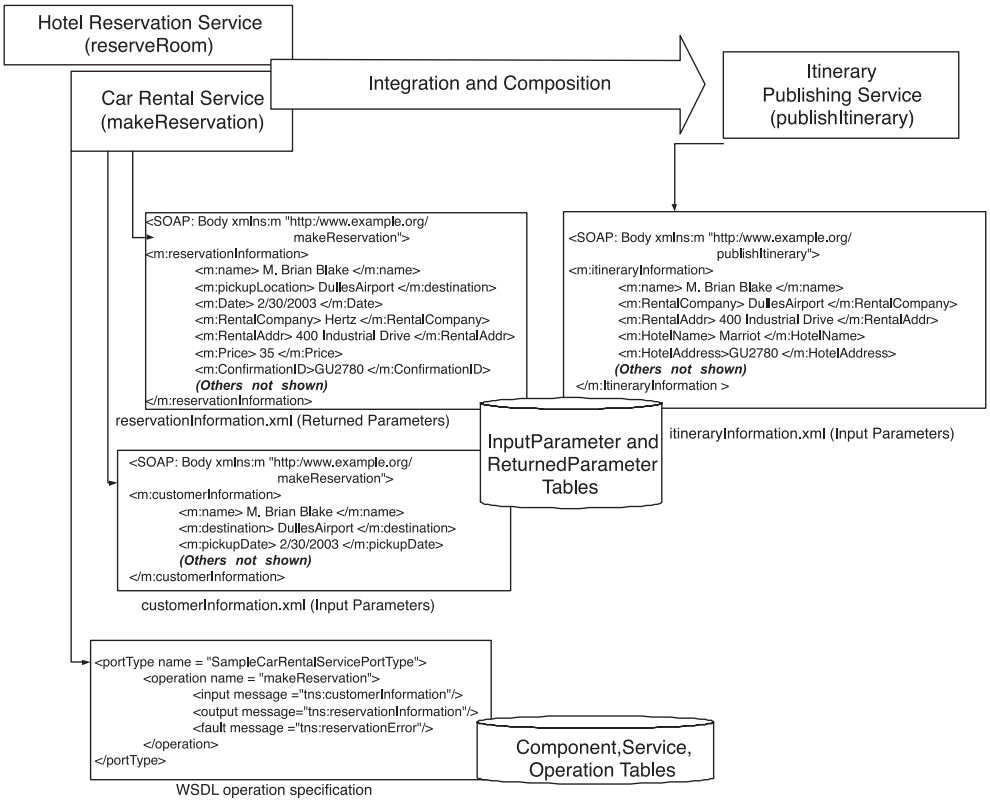

Fig. 13. Composing web services based on the travel agency control model.

Examples of these specifications are illustrated in Fig. 13. In Fig. 13, the reserveRoom and makeReservation web services are composed and integrated with the publishItinerary service. The input and output parameters are grouped and represented using sample SOAP messages customerInformation.xml and reservationInformation.xml respectively. The service is specified using WSDL specification (i.e. label as the WSDLOperationSpecification). There are SOAP and WSDL message examples shown for the input/returned parameters of the car rental service, and another SOAP message is shown for the input parameters to the itinerary publishing service.

The database illustrations show the specific WARP data entities that relate to the input and returned parameters represented in the SOAP messages. The swimlane stereotypes, HotelReservation, CarRental, and ItineraryPublishing, represent the location of the web service. The independent activities, such as reserveRoom, makeReservation, and publishItinerary, represent the actual web services. Each activity would be documented with a WSDL specification and two SOAP representations that describe input and returned parameters. The class notations (customerMessage and reservationMessage) between the activities contain the subset of parameters from the outgoing and incoming SOAP messages (i.e. the customerInformation.xml and reservationInformation.xml specifications). This subset of parameter information is exchanged among the web services.

Summarizing the WARP scenario, a set of WSDL and SOAP specifications, as in Fig. 13, are accessible to a set of information agents. These agents present the available services to a user within an initial development environment consisting of UML-based service representation views, perhaps using a case tool such as Rational Rose. The human user then creates the control model and role collaboration models, as in Fig. 12a and b, based on the type of service composition that is desired. The WARP data entities, also illustrated in Fig. 13, are populated and used in subsequent agent configuration processes prior to the real-time web service composition. This is a simple example that illustrates the use of the WARP models and web service specifications. The assumption in this example is that services in this domain will typically be produced at separate businesses. The focus of this paper is not the data integration aspects of this work, although further work toward solving these problems is explained in related research [38].

6. Agents and interactions for service modeling and composition

A major characteristic of the WARP architecture is the use of agent-oriented programming approaches to realize the workflow composition and management of services exploiting the adaptability of agents. Software entities have been defined as agents when they possess certain characteristics. The grouping of these characteristics has been defined as levels of *agency* [21]. Entities classified as weak agency tend to possess characteristics such as autonomy, social ability, reactiveness, and/or proactiveness. However, strong agency dictates that, in addition to the weak agency characteristics, software entities must also possess mental abilities, even emotions, which are usually attributed to human behavior.

WARP agents can be classified as possessing weak agency as they are autonomous software entities that have knowledge of their environment to reactively and proactively mediate service executions and process management. WARP agents have independent assignments to invoke specific services or manage specific business processes, reacting to both functional and nonfunctional conditions of the workflow. In addition, these agents are programmed with general proactive abilities to effect change when negative nonfunctional events occur. The following sections describe the software design process used to program the WARP agents for the SCW domain, a formal definition of the WARP agents, and a description of the interactions among agents.

6.1. Agent-oriented software design process for service modeling and composition

Software design and configuration in the WARP environment, in a general sense, consists of five steps as illustrated in Fig. 14. In the first step, SMAs 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 locally registered component-based services. These SMAs search for relevant services, and,

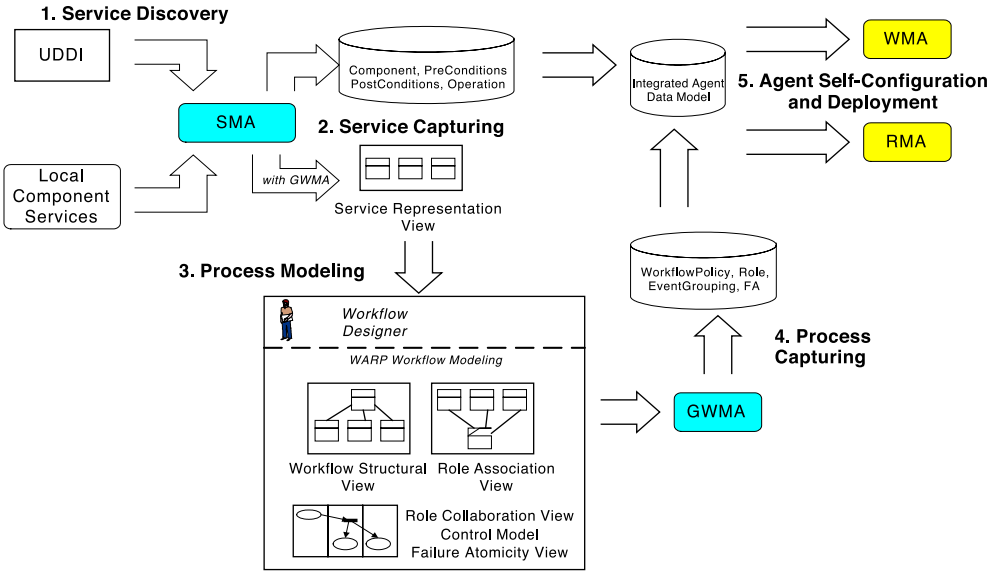

Fig. 14. A software engineering process for developing the WARP environment.

in the second step, save the service characteristics in the service-oriented data model as illustrated in Fig. 6.

Also in the second step, with help from the GWMA, the service characteristics are captured in WARP models. In the third step, a workflow designer accesses the available services as Service Representation Views. The workflow designer then creates a set of cross-organizational process models. Once the process modeling is complete, in the fourth step, the GWMA captures the WARP models and extracts the process information. This process information is stored in the process-oriented data model as illustrated in Fig. 10. Consequently, an integrated data model of both service and process data models is ready for agent access. In the fifth and final step, application-layer agents (Workflow Manager Agents and Role Manager Agents) access the integrated data model and configure themselves for workflow enactment in the SCW environment.

6.2. Formal definition of the application-layer agents

In a structural sense, the RMA is composed of a number of workflow roles that it has responsibility to proxy. The operational responsibilities of an RMA consist of the invocation of distributed services at the appropriate time, acknowledgement of completions to other RMAs and WMAs, monitoring of the condition of the services, reporting of known errors, and the proactive reporting of perceived errors. Formally, the RMA can be defined by the set of one or many workflow roles, W_R, such that RMA=$\{W_{R1}, W_{R2}, \ldots, W_{Rn}\}$. Therefore, the workflow role is defined as the tuple W_R:

$$W_R = (Rn_i, S, G_i) \text{ where:} \quad (1)$$

1. Rn_i is the unique name for the workflow role.
2. S is the set of service: S=$\{s_1, s_2, s_3, \ldots, s_n\}$.
3. G_i is the grouping or transition type for multiple services, which can be AND or XOR. An AND type specifies that multiple services are executed concurrently, while an XOR type specifies that one service must be chosen from among multiple possible services (typically based on past performance).

An atomic service can be represented by the tuple,

$$S_i\text{: } S_i = (Ns_i, O_i, T_i) \text{ where:} \quad (2)$$

1. Ns_i is the combination of the name and an unique identification that distinguishes the service from the set of available services.

2. O_i is the organization name (routing) of the service used for the distributed access to the service.
3. T_i is the type of service, which can be webservice or invocation. The webservice types of services are accessed from the UDDI registry and executed using SOAP messages. The invocation types of services, at this point, are access locally using reflective functionality.

Given the definition of the RMAs, a straightforward definition of the WMA would be the sequential set of RMAs and the workflow name or identification. However, if WMAs were defined in this manner, they would have global knowledge of the workflow process and consequently global bias. An innovation in the WARP environment, which further distinguishes it from other approaches, is the fact that the WMA does not control the workflow process. As such, there is no centralized controller that may result in a single point of failure. The major responsibilities of the WMAs are to initiate the workflow instance and monitor for nonfunctional changes. In this way, the workflow process is enacted through the coordination among the RMAs and the process control is distributed. Given this explanation, the WMA can be defined as the tuple WMA:

$$\text{WMA} = (Wn_i, J, Lr_i, Fr_i, E) \text{ where:} \quad (3)$$

1. Wn_i is the unique name for the workflow process.
2. J is the set of job events: $J=\{j_1, j_2, j_3, \ldots, j_n\}$; j_i is an atomic event that triggers the WMA to initiate a new workflow process.
3. Lr_i is the name of the first workflow role in a new workflow process, and Fr_i is the name of the final workflow role.
4. E is the set of exception-handling workflows: $E=\{e_1, e_2, e_3, \ldots, e_n\}$.

Agents must be programmed to handle errors or exceptions in the workflow execution. Agents execute a new series of actions, an *exception-handling workflow*, to make corrections for exceptions. The exception-handling workflow defines a workflow in a similar fashion to the WMA tuple. An exception-handling workflow is triggered when an agent perceives workflow errors resulting from improper data exchange. The exception-handling workflow is defined by the value of the erroneous data, V, as opposed to the job event, J. An assumption in this approach is that the exception-handling workflows are atomic; therefore there are no nested exception-handling workflows. The exception-handling workflow e_i, is defined as the tuple:

$$e_i = (Wn_i, V, Lr_i) \text{ where:} \quad (4)$$

1. Wn_i and Lr_i are the workflow name and lead role name as in the *WMA* tuple.
2. V is the set of value names: $V=\{v_1, v_2, v_3, \ldots, v_n\}$. v_i is defined with the Value_Name identifier, Nv_i and the workflow designer-specified error code or improper data value, D_i. Therefore, v_i, can be defined as the tuple v_i:

$$v_i = (Nv_i, D_i). \quad (5)$$

6.3. The operation of WARP agents in the SCW environment

The WARP environment uses event-driven communication similar to the publish/subscribe communication mechanism specified in Linda and other approaches [14]. In addition, the WARP architecture is configured using a process repository that agents access at run-time. This connectivity supports the run-time evolution of the workflow process and service-oriented bindings. In developing the WARP architecture, many basic workflow processes were implemented, such as pre-run-time and run-time configuration, normal and advanced workflow enactment modes, exception-handling enactment, AND and XOR condition enactment, synchronization, and workflow instance management.

WARP agents also have several operational modes in realizing the above workflow operations. An innovation in the WARP architecture is its flexibility to support the loose and tight-coupling of agents to the pre-established process repository. In a loosely coupled mode, agents only access the process repository at configuration prior to real-time execution. In a tightly coupled mode, agents access the process repository at configuration, at the initiation of each workflow, and during each workflow step.

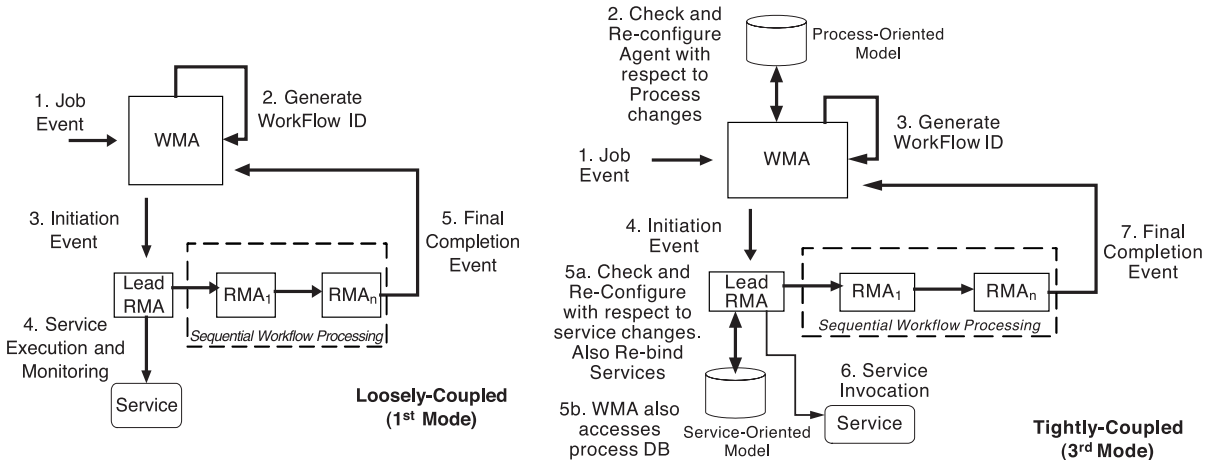

Fig. 15. Agent interactions for two modes of workflow enactment.

In the WARP approach, there are three modes of operation with various degrees of coupling. The first mode is the most loosely coupled. Both WMAs and RMAs are configured from the process repository at system initiation. In the second mode, the RMAs verify and reconfigure the service bindings during each workflow step while the WMAs remain as defined in the first mode. In the third mode, the WMAs and the RMAs access the repository for each action. Therefore in the third mode, WMAs can re-configure the process at the beginning of job and at each step. In addition, the RMAs reconfigure service bindings during each workflow step. The first mode and the third mode are illustrated in Fig. 15.

The three operational modes in the WARP environment are designed to support different organizational needs. In settings where all distributed services are within one enterprise, business process and service changes may be less frequent. In this static setting, the business can take advantage of the faster performance of the first operational mode that limits access to the process and service repositories at run-time. However, for business processes that are predominantly defined by the distributed services of other *provider* businesses, the services may change frequently outside of the control of the originating organization. In these dynamic settings, operational modes 2 and 3 may be optimal despite the fact that the system performs slower. The third operational mode also supports businesses that change their processes in real-time, for example, businesses that change their processes based on the change of customer demand for products or capabilities. In Section 7, there is a discussion of the performance overheard associated with each of the operational modes using the WARP prototype.

7. The WARP prototype and operational evaluation

To evaluate the WARP architecture, a WARP prototype and experiment was developed as illustrated in Fig. 16. For initial purposes, the prototype used only invocation-based services. JavaBean components were chosen for the implementation of component-based services. The reason for this choice was the ability to use reflective processes to emulate local invocations, in addition Java introspection could be used to simulate UDDI calls. To implement the event-based communication, JavaSpaces technology was incorporated. The choice to use JavaSpaces as opposed to other implementations, such as IBM's TupleSpaces, was because of the seamless integration with Java Beans [4,16]. In addition, JavaSpaces is freely available. To simulate cross-organizational interactions, RMAs were distributed across multiple Windows 2000 Professional machines. The integrated data model was replicated on all machines using Oracle8iLite, which is a relational database management system that is operational in the personal computer environment. Finally, services were registered both on Java's RMI registry and given web accessibility using the Tomcat web server.

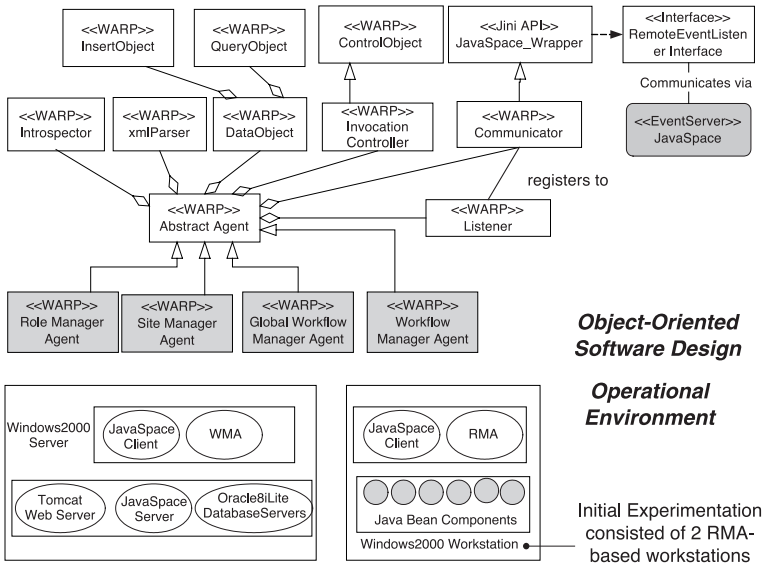

Fig. 16. The WARP prototype environment and low-level object-oriented design.

The WARP agents were built using pure Java as the programming language. All agents are JavaBean/Applets, meaning they have both event-based and call-return style of communication. The agents have the graphical user interface and Internet functionality inherent to JavaBean/Applets. The agents communicate through the JavaSpaces server. The agent architecture was developed in UML using strict object-oriented design practices. The WARP Representations were captured using Rational Rose developer tool. Using the Rose Extensibility Interface (REI), agents can access the representations via the model file (.mdl) created by Rational Rose. In this way, the Rational Rose tool was used as the workflow modeling and capturing tool. The design of this prototype follows the WARP architectural design explicitly. The functionality of the Abstract Agent is the most important functionality in the design. Using the publish/subscribe communication mechanism, 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 class. 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.

An experiment was conducted using the WARP prototype to evaluate the impact of the overhead caused by the three modes of real-time configuration as discussed in Section 6.3. The various modes were evaluated against a baseline system that uses the same technology but has hard-coded interactions. This baseline system has less functionality than any of the WARP operational modes and was created with the sole purpose to evaluate performance. The baseline system is static and only performs one specific 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 was proven through experimentation. In this baseline case, workflow interactions are hard-coded in memory.

The system was then configured for the operational modes as in Section 6.3. The first case was a WARP system that performed pre-run-time configuration of the workflow processes and available services, as Operational Mode 1. The second and third modes (Operational Modes 2 and 3) are more tightly coupled to the database, but more evolvable. In Operational Mode 2, RMAs check services at each

Table 1
Evaluating the operational modes of the WARP environment (percent latency)

Number of workflow steps	Java components with memory-based workflow look-up tables (base case) (%)	WARP configured agents that check database only at configuration (Operational Mode 1) (%)	WARP configured agents that check services at each step (Operational Mode 2) (%)	WARP configured agents that check policy and service each step (Operational Mode 3) (%)
3	2	9	13	23
5	2	10.7	14	27
50	2	12	15	37

step in addition to the initial configuration. In Operational Mode 3, the RMAs and WMAs check services and the workflow process, respectively, at the completion of each workflow step. The overheard results are shown in Table 1.

The increase in overhead by operational mode was anticipated. As the operational mode was changed from mode 1 to mode 3, the system queries the database more times, thus increasing the latency on the completion of the workflow process. Therefore, an important aspect is that the latency is directly dependent on the size and performance of the underlying database.

Another result was that the overhead increased with the increase of workflow steps. This was relatively unexpected since the latency was constant in the base case as the number of workflow steps was increased. One explanation of this increase of overhead was the degradation of the database connection using the chosen database driver as more queries were executed. However, further experiments demonstrated that this could not account for all of the additional delay. The performance of the reflective calls between the agents and the Java Beans components also varied with load. In future work, we plan to run additional experiments to prove that this increase in latency is implementation-related and not design-related.

Even with the latency found in this experiment, this approach is promising with respect to other approaches that use CORBA-based technology. Reports have shown that the communication overhead associated with CORBA is consistent with the overhead associated with the second operational mode [28]. These experiments verify that the WARP approach is a viable SCW solution considering the comparable performance to other middleware technology that lacks the reconfiguration functionality.

8. Discussion and conclusion

The WARP approach to service composition extends the state of the art discussed in Section 2.3. The work of Mennie, CHAIMS, and Chakroborty [9,33] successfully support base-level composition approaches, however consideration of complex workflow interactions is not evident. These approaches tend to be formal, but consider composition at a relatively atomic level. The WARP approach extends the work in these projects by investigating complex workflow interactions. Another variation is their sole use of text-based specification, while the WARP approach uses both visual (software engineering standard) and text-based specifications. The work of Casati and Benatallah [1,8] consider more complex workflow-oriented interactions with visual *and* text-based specification approaches. Both adopt formal approaches to the service composition, but have non-agent-oriented interactions in the operation of their respective internal architectures. The WARP approach extends the approaches of Casati and Benatallah by considering both the complex workflow-oriented interactions among the services, *and* the complex interactions of agents internal to our architecture. The work of Helal et al. [20], Chen et al. [10], and Singh [30] concentrate on agent architectures for service composition and discovery. Though these approaches are promising with respect to their problem domains, their service composition specification languages are neither text-based nor were they intended to address the complex workflow configuration issues supported in the WARP approach.

Another innovation in the WARP approach is the software developmental process (as in Fig. 14) in the creation of these composite systems that respect workflow protocols. This software development lifecycle is consistent with industry-standard software engineering

lifecycles. In fact, the use of industry-standard modeling methodologies (i.e. UML) assists the integration into industry processes. In experiments of the WARP prototype, we determined the system latency of the WARP approach was comparable to related middleware approaches. These performance results verify the appropriateness of the WARP approach considering the major increase in operational flexibility with overhead comparable to related approaches.

The WARP approach is one of few projects that use an industry-standard developmental process and modeling techniques. By using multiple UML-based 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 in available object-oriented software engineering tools. 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. However, there were several issues encountered in this approach that are the source for future investigations. As briefly discussed in earlier sections, we have started investigating agent approaches for the major problem of data integration and modeling. 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 formidable challenge to create operational patterns to deal with these formats. In future work, we intend to investigate DAML-S as a solution to the data management issues.

Another problem is support for error-handling. An inefficient approach taken in this work is to create new models for each error-handling case. These cases can be more efficiently wrapped into the *general* operational models (i.e. Control Model and Role Collaboration Model). Future work would consist of further evaluating this approach with advanced workflow patterns and interactions while also incorporating UDDI and SOAP technologies.

References

[1] B. Benatallah, M. Dumas, Q. Sheng, A. Ngu, Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, 18th International Conference on Data Engineering, 2002 Feb.

[2] M.B., Blake, Agent-Based Workflow Modeling for Distributed Component Configuration and Coordination, PhD dissertation, George Mason University, 2000 online at: http://www.cs.georgetown.edu/~blakeb/pubs/diss.zip.

[3] M.B. Blake, An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services, Proceedings of the 11th IEEE WETICE, 2002 June, Pittsburgh, PA.

[4] M.B. Blake, Agent-Based Communication for Distributed Workflow Management using Jini Technologies, International Journal on Artificial Intelligence Tools, vol. 12 (1), World Scientific, Toh Tuk Link, Singapore, 2003 March, pp. 81–99.

[5] M.B. Blake, H. Gomaa, Object-Oriented Modeling Approaches to Agent-Based Workflow Services, in: C. Lucena, et al. (Eds.), Software Engineering to Large-Scale Mult-Agent Systems II, LNCS, vol. 2960, Springer-Verlag, Heidelberg, Germany, 2004 Feb.

[6] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, 1999, Addison-Wesley, Reading, MA.

[7] BPEL4WS (2002): http://www.ebpml.org/bpel4ws.htm.

[8] F. Casati, L. Jin, S. Ilnicki, M.C. Shan, An Open, Flexible, and Configurable System for Service Composition, HPL technical report HPL-2000-41, 2000 April.

[9] D. Chakraborty, F. Perich, A. Joshi, T. Finin, Y. Yesha, A Reactive Service Composition Architecture for Pervasive Computing environment, 7th Personal Wireless Communications Conference (PWC 2002), 2002 October.

[10] Q. Chen, U. Dayal, M. Hsu, M.L. Griss, Dynamic-Agents, Workflow and XML for E-Commerce Automation. EC-Web (2000) 314–323, London, UK.

[11] M. Dumas, A.H.M ter Hofstede, UML Activity Diagrams as Workflow Specification Language, The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1–5, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2185, Springer, Heidelberg, Germany, 2001, ISBN 3-540-42667-1.

[12] R. Eshuis, Semantics and Verification of UML Activity Diagrams for Workflow Modelling, PhD thesis, University of Twente, 2002.

[13] D. Florescu, A. Grunhagen, D. Kossman, XL: An XML Programming Language for Web Service Specification and Composition'' WWW 2002, ACM Press, Honolulu, Hawaii, 2002.

[14] D. Gelernter, Current research on Linda, Research Directions in High-Level Parallel Programming Languages (1991) 74–76.

[15] J.W.J Gijsen, N.B. Szirbik, G. Wagner, Agent technologies for virtual enterprises in the one-of-a-kind-production industry, International Journal of Electronic Commerce 7 (1) (2002 October) 9–34.

[16] H. Gomaa, Inter-Agent Communication in Cooperative Information Agent-Based Systems in Cooperative Information Agents III, Lecture Notes in Artificial Intelligence, vol. 1652, Springer-Verlag, Berlin, 1999, pp. 137–149.

[17] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, Boston, MA, 2000.

[18] P. Grefen, K. Aberer, Y. Hoffner, H. Ludwig, CrossFlow: cross-organizational workflow management in dynamic vir-

tual enterprises, International Journal of Computer Systems Science and Engineering 15 (5) (2000) 277–290.
[19] G. Heineman, W. Council, Component-Based Software Engineering Putting the Pieces Together, Addison-Wesley, Reading, MA, 2001, Reading, MA.
[20] A. Helal, M. Wang, A. Jagatheesan, R. Krithivasan, Brokering Based Self Organizing E-Service Communities, Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems (ISADS), March 26–28, 2001, Dallas, Texas, 2001.
[21] N.R. Jennings, K.P. Sycara, M. Wooldridge, A Roadmap of Agent Research and Development In Journal of Autonomous Agents and Multi-Agent Systems 1 (1) (1998 July) 7–36.
[22] M. Kamath, K. Ramamrithan, Correctness issues in workflow management, Distributed Systems Engineering Journal—Special Issue on Workflow Systems 3 (4) (1996 December) 213–221.
[23] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J. Loingtier, J. Irwin, Aspect-Oriented Programming, Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland, LNCS, vol. 1241,Springer-Verlag, Heidelberg, Germany, 1997 June, pp. 75–88.
[24] Y. Lei, M.P. Singh, 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.
[25] F. Leymann, Web Services Flow Language (WSFL 1.0), IBM, 2001 May. http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
[26] P. Massimo, K. Sycara, T. Kawamura, Delivering Semantic Web Services, Proceedings of the WWW2003, Budapest, Hungary, 2003 May.
[27] D.W. Mennie, B. Pagurek, An Architecture to Support Dynamic Composition of Service Components, Proc. of the 5th International Workshop on Component-Oriented Programming–WCOP 2000, Sophia Antipolis, France, 2000 June, pp. 1–8.
[28] C. O'Ryan, D.C. Schmidt, J. Noseworthy, Patterns and Performance of a CORBA Event Service for Large-scale Distributed Interactive Simulations, International Journal of Computer Systems Science and Engineering, vol. 17 (2), CRL Publishing, Leicester, UK, 2002 March, pp. 1–19.
[29] N. Sample, D. Beringer, L. Melloul, G. Wiederhold, CLAM: Composition Language for Autonomous Megamodules, Third Int'l Conference on Coordination Models and Languages, COORD'99, Amsterdam, April 26–28, 1999.
[30] M.P. Singh, B. Yu, M. Venkatraman, Community-based service location, CACM 44 (4) (2001) 49–54.
[31] SOAP (2002): http://www.w3.org/TR/soap12-part0/.
[32] S. Thatte, XLANG: Web Services for Business Process Design, 2001, Microsoft http://www.gotdotnet.com/team/xml-wspecs/xlang-cl/default.htm.
[33] The CHAIMS Project (2002): http://www-db.stanford.edu/CHAIMS/.
[34] UDDI (2002): http://www.uddi.org/.
[35] UDDI Specification 2.04 (2002): http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf.
[36] W.M.P. Van der Aalst, Don't go with the flow: Web Services composition standards exposed, IEEE Intelligent Systems, 2003 Feb, pp. 72-85.
[37] Web Services (2002): http://www.w3.org/2002/ws/desc/.
[38] A.B. Williams, A. Padmanabhan, M.B. Blake, Local Consensus Ontologies for B2B-Oriented Service Composition, Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS2003), ACM Press, Melbourne, Australia, 2003 July, pp. 647–654.
[39] WSDL (2002): http://www.w3.org/TR/wsdl.
[40] WSFL (2002): http://www.ebpml.org/wsfl.htm.
[41] L. Zeng, A. Ngu, B. Benatallah, M. O'Dell, An agent-based approach for supporting cross-enterprise workflows, Proceedings of the 12th Australasian Conference on Database Technologies, 123–130, Queensland, Australia, 2001.

M. Brian Blake is an Assistant Professor in Department of Computer Science at Georgetown University. He received a Bachelor of Electrical Engineering from Georgia Institute of Technology and a Master of Science in Electrical Engineering from Mercer University, both in Atlanta, Georgia. He has a PhD of Information Technology from George Mason University, Fairfax, VA with a concentration in Information and Software Engineering. He has over 10 years experience in projects related to object-oriented software engineering and autonomous distributed systems both in industry and academia. He has consulted for such companies as General Electric, Lockheed Martin, Trident Data Systems, and The MITRE Corporation. He has published over 40 refereed journal papers and conference proceedings in the areas of agents and workflow, enterprise integration, component-based software engineering, distributed data management, and software engineering education.

Hassan Gomaa is Chair and Full Professor in the Department of Information and Software Engineering at George Mason University, Fairfax, VA. He received a BSc (Eng) with First Class Honors in Electrical Engineering from University College, London University, and the DIC and PhD in Computer Science from Imperial College of Science and Technology, London University. He has over 25 years experience in software engineering, both in industry and academia, and has published over 125 technical papers and two textbooks, "Software Design Methods for Concurrent and Real-Time Systems" and "Designing Concurrent, Distributed, and Real-Time Applications with UML", both published by Addison Wesley. His current research interests include object-oriented analysis and design for concurrent, real-time, and distributed systems, software architecture, software product lines, intelligent software agents, and software process models. He also teaches short courses in industry on software design and consults in both the technical and management aspects of software engineering.