

A SOFTWARE ARCHITECTURE FOR DISTRIBUTED WORKFLOW
ENACTMENT WITH AGENTS AND WEB SERVICES

by

Paul Allen Buhler

Bachelor of Science
The Citadel, 1986

Master of Science
Johns Hopkins University, 1990

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in the
Department of Computer Science and Engineering
College of Engineering and Information Technology
University of South Carolina

2004

Dissertation Advisor

Committee Member

Committee Member

Committee Member

Dean of the Graduate School

Dedication

To my family...

for their seemingly infinite patience.

especially my wife Melanie, for holding things together through my periods of distraction.

to my children: Leslie, Patrick and Heather who have trouble remembering a time when I was not a student.

To my parents...

who first instilled the belief that I could accomplish whatever I set my mind to.

to my father, who will always be the original Dr. Buhler – your battle with cancer has taught me more about perseverance and strength than you know.

Acknowledgements

I would like to thank each of the members of my dissertation committee for their time, commitment and steady encouragement over the years.

I especially wish to acknowledge my dissertation advisor Dr. José Vidal for his financial support, which helped keep me afloat. It is an honor to be your first doctoral student; I feel we have both learned valuable lessons as we have felt our way through this process.

I am indebted to Dr. Michael Huhns who first encouraged me to study at the University of South Carolina. Dr. Huhns provided the first inspiration for this work over a lunchtime conversation at the Barcelo Sants hotel in Barcelona, Spain during the Autonomous Agents 2000 conference.

I appreciate Dr. Larry Stephens for introducing me to the topic of Knowledge Representation and Reasoning. His course has provided foundational knowledge that I intended to leverage in the domain of Semantic Web services.

I express my deepest gratitude to Dr. Christopher Starr for his unwavering friendship and untold support over the years.

Finally, I would be remiss if I did not recognize the encouragement and support I have received from the administration, my colleagues, and my students at the College of Charleston.

Preface

The research described in this dissertation foretells a future in which traditional approaches to writing software via creative processes will be supplanted by new techniques based upon compositional paradigms. This shift toward composition will enable wide-scale reuse of both hardware and software assets. Compositional approaches will open the door to new application areas (e.g., Business Process Management Systems) and change fundamental problem solving approaches in others. Software structures will become flexible and agile, allowing applications to respond intelligently to changes in their operational environment.

The underlying structure of this dissertation is itself driven by the theme of composition. This dissertation is composed in part of a collection of papers that I have authored and assembled into a coherent whole. As such, the individual chapters of this work have the ability to stand-alone; however, when viewed collectively, the proverbial whole is greater than the sum of its parts. An introductory chapter serves to unite the works and provides the lens through which brings the whole into focus. The chapters, based upon an externally reviewed and accepted submission to a journal, conference, or workshop are identified below:

Chapter 2

P. Buhler and J. M. Vidal, "Towards adaptive workflow enactment using multiagent systems," *Information Technology and Management Journal: Special Issue on Universal Enterprise Integration*, Vol 6, No 1, 2005, pg 61 - 87.

Reprinted with the permission of Kluwer Academic Publishers.

Chapter 3

P. Buhler and J. M. Vidal, "Integrating Agent Services into BPEL4WS Defined Workflows," presented at the Fourth International Workshop on Web-Oriented Software Technologies (IWWOST '04), Munich, Germany, 2004.

Chapter 4

P. Buhler and J. M. Vidal, "Enacting BPEL4WS Specified Workflows with Multiagent Systems," presented at the Second Web Services and Agent-based Engineering Workshop (WSABE '04), New York, 2004.

Table of Contents

CHAPTER 1	BACKGROUND AND OVERVIEW.....	1
1.1	Research goals and methodology.....	6
1.1.1	<i>Hypothesis.....</i>	7
1.1.2	<i>Research Methodology.....</i>	7
1.2	Overview of the remaining chapters	10
CHAPTER 2	TOWARDS ADAPTIVE WORKFLOW ENACTMENT USING MULTIAGENT	
	SYSTEMS	12
2.1	Introduction.....	12
2.2	Enterprise Software.....	14
2.2.1	<i>Enterprise Software Architecture Trends</i>	<i>15</i>
2.2.2	<i>Web services as Enterprise Software Components.....</i>	<i>18</i>
2.3	Workflow Management Systems.....	21
2.3.1	<i>Adaptive Workflow in Context</i>	<i>22</i>
2.3.2	<i>Workflow Reference Model.....</i>	<i>24</i>
2.3.3	<i>Workflow Tools</i>	<i>26</i>
2.4	BPEL4WS.....	27
2.5	DAML-S	32
2.6	Agent-based Workflow Approaches.....	35
2.6.1	<i>BPEL4WS for multiagent systems.....</i>	<i>37</i>
2.6.2	<i>Multiagent workflow enactment as an autonomic system.....</i>	<i>43</i>
2.7	Related developments	45
2.8	Conclusion and future work.....	47

CHAPTER 3 INTEGRATING AGENT SERVICES INTO BPEL4WS DEFINED

WORKFLOWS.....	48
3.1 Introduction.....	48
3.2 An Example BPEL4WS Workflow	50
3.3 Infrastructure.....	52
3.3.1 BPWS4J.....	53
3.3.2 WSAG.....	55
3.3.3 JADE.....	56
3.4 End-To-End Demonstration.....	57
3.4.1 <i>The Software Development Process</i>	58
3.4.2 <i>Putting the Pieces Together</i>	59
3.5 Conclusion	63

CHAPTER 4 ENACTING BPEL4WS SPECIFIED WORKFLOWS WITH MULTIAGENT

SYSTEMS	66
4.1 Introduction.....	66
4.2 A Sample BPEL4WS Workflow	67
4.3 Architecture and Design	70
4.4 Coordination of the Workflow Agents	72
4.4.1 <i>Xindice as a Coordination Medium</i>	73
4.4.2 <i>CPNs as a Flow Control Mechanism</i>	75
4.5 Implementation Details.....	79
4.5.1 <i>Target Agents</i>	79
4.5.2 <i>Distributed Workflow Agents</i>	80

4.6	System Configuration	84
4.6.1	<i>Configuring the WSAG</i>	84
4.6.2	<i>Configuring the Workflow Agents</i>	86
4.7	Conclusion	89
CHAPTER 5	CONCLUSION AND FUTURE WORK	90
5.1	Major Research Contribution.....	90
5.2	Future Research Directions	92
5.2.1	<i>Externalization of Business Rules</i>	92
5.2.2	<i>Dynamic Business Partner Selection</i>	92
5.2.3	<i>Automated Petri Net Creation</i>	93
5.2.4	<i>Semantic Service Replacement</i>	93
5.2.5	<i>Multiagent System Design Methodology</i>	94

List of Tables

Table 1.1 Research questions in software engineering	8
Table 1.2 Research results in software engineering.....	9
Table 1.3 Validation techniques in software engineering.....	9
Table 2.1 Component Model Standards.....	20

List of Figures

Figure 1.1 The evolution of system assumptions.....	3
Figure 1.2 The ingredients of a Universal Business Integration Platform.....	4
Figure 1.3 The dissertation work as an act of synthesis	5
Figure 1.4 Relative maturities of relevant technologies	6
Figure 2.1 Evolution of programming paradigms.....	16
Figure 2.2 Evolution of Enterprise Software Architectures.....	17
Figure 2.3 Workflow Perspectives.....	23
Figure 2.4 WfMC Reference Architecture © WfMC	25
Figure 2.5 A WSDL file for a currency exchange rate Web service	28
Figure 2.7 An Example BPEL4WS Workflow Description	31
Figure 2.8 Service-Oriented Architecture Model	32
Figure 2.9 Building Blocks Mapped to BPEL4WS Activities	39
Figure 2.10 Architectural Components of the Multiagent Enactment Mechanism	42
Figure 3.1 A Use Case Maps model of the BPEL4WS example.....	52
Figure 3.2 BPWS4J viewed as a layered model.	53
Figure 3.3 The WSAG enables messaging between Web services and agents.	55
Figure 3.4 Jade’s Remote Agent Management utility	56
Figure 3.5 Invoking the workflow from the command-line	59
Figure 3.6 WSAG Deployed Web services screen	62
Figure 3.7 BPWS4J Partner Identification Screen.....	62
Figure 4.1 A UCM diagram for the example workflow.	70
Figure 4.2 Example documents stored in Xindice	75

Figure 4.3 A PN Model for the example workflow.....	76
Figure 4.4 Refinement of P2 with a subnet.....	78
Figure 4.5 UML Sequence diagram with sample messages	79
Figure 4.6 UCM diagram of a Target Agent.....	80
Figure 4.7 UCM diagram of a Distributed Workflow Agent.....	81
Figure 4.8 The components of the distributed enactment mechanism.....	84
Figure 4.9 Configuration of the Gateway Agent	85
Figure 4.10 The collection of workflow agents in the system.....	89
Figure 5.1 Various multiagent workflow adaptation strategies	91

Chapter 1

Background and Overview

Today's software systems are becoming more net-centric, distributed, and heterogeneous. Moore's law (processor power), Gilder's law (bandwidth expansion), and Metcalfe's law (network dynamics) predict a future that will require us to change our current perceptions about computing [1, pg 268][2]. Hardware, software and networking technology will combine in a milieu in which they become ubiquitous and inseparable. The acceleration of technology and time-to-market pressures makes it increasingly difficult to produce software. In order to achieve the promise of the information age, software developers will require new abstractions that will allow them to manage the overwhelming complexity of this digital landscape.

In 1999, NSF sponsored a workshop to discuss software engineering research strategies. The participants at the workshop drew several conclusions about software engineering research, one of which is particularly relevant to the vision of my work. This conclusion is summarized by the metaphor "skate to where the puck is going," meaning researchers need to be more forward thinking. "Heterogeneous distributed systems, dynamically changing software structures, and interactions among autonomous agents," were explicitly mentioned as requiring focused research [3].

Traditional software engineering methodologies are giving way to new software development paradigms. Component-based software engineering and agent-oriented

software engineering are two paradigms that are garnering much attention. Although typically thought of as separate disciplines, it is likely that they are not only related, but also ultimately dependent upon one another. I believe that passive software components can be liberated by the proactive and social nature of agents. In effect agent-based technologies will provide the mechanism for components to seek work, enter into cooperative agreements and thus otherwise address the requirements of dynamic, heterogeneous environments.

In order to turn this vision into reality, the complexity involved in the combination of software components and agents must be managed via carefully constructed software architecture. In fact, the overwhelming complexity of contemporary software systems has revealed that principled software architecture is a separate discipline from process-focused software engineering. This should come as no surprise since architecture and engineering have long been viewed as separate professions. The following quote helps illustrate the difference between these two domains and their distinct points of view.

*Consider the role of an architect versus that of an engineer in a construction project. Engineers think about the techniques and skills in **constructing** the building. Architects think about the art and technique of **designing** the building. Engineers think about how they can build better doors and stronger beams. Architects think about how the building will respond functionally and aesthetically to the needs of its inhabitants. Engineers think in terms of **analysis** – the process of solving a problem by breaking it into components. Architects think in terms of **synthesis** – the process of putting components together into a coherent system. Engineers implement and construct. Architects create and configure. You need both to create a building that is solidly built and yet creatively envisioned. But architecture must precede engineering. And as the complexity and scope of*

the building increases, the task of the architect becomes more important and difficult. [1]

Underlying the now central role of software architecture is the changing landscape of software systems. The emerging ubiquity and distribution of computational resources is forcing the development community to grapple with a new set of system assumptions.

Figure 1.1 [4] highlights these fundamental changes.

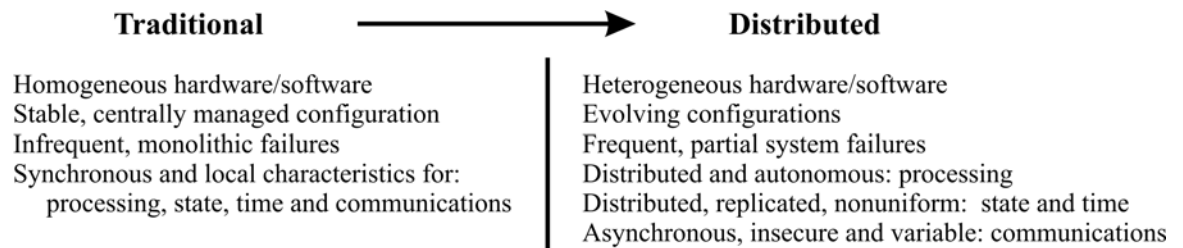


Figure 1.1 The evolution of system assumptions

The adoption of distributed system assumptions is dramatically affecting information technology deployments within the corporate enterprise. Globalization and competitive pressures have forced business to turn toward technological solutions that enable agile business processes that are responsive to changes in the marketplace and enterprise environment. Information technology is no longer considered a necessary evil, but rather an essential lubricant – reducing integration friction between applications and business functions. In fact, many in the business process integration community believe that the foundations of a Universal Business Integration Platform (UBIP) are emerging [5].

Figure 1.2 depicts the components of a UBIP.

Traditionally, Enterprise Applications Integration (EAI) and Enterprise Information Integration (EII) were viewed as the essential ingredients of an integration strategy. This view has expanded to include both Service-Oriented Computing (SOC) and Business Process Management Systems (BPMS). In Figure 1.2, SOC is represented by Service

Oriented Architectures (SOA), Web services, and Service Oriented Development of Applications (SODA). BPMS is depicted as Business Process Orchestration (BPO), Business Activity Monitoring (BAM) and Business Intelligence (BI).

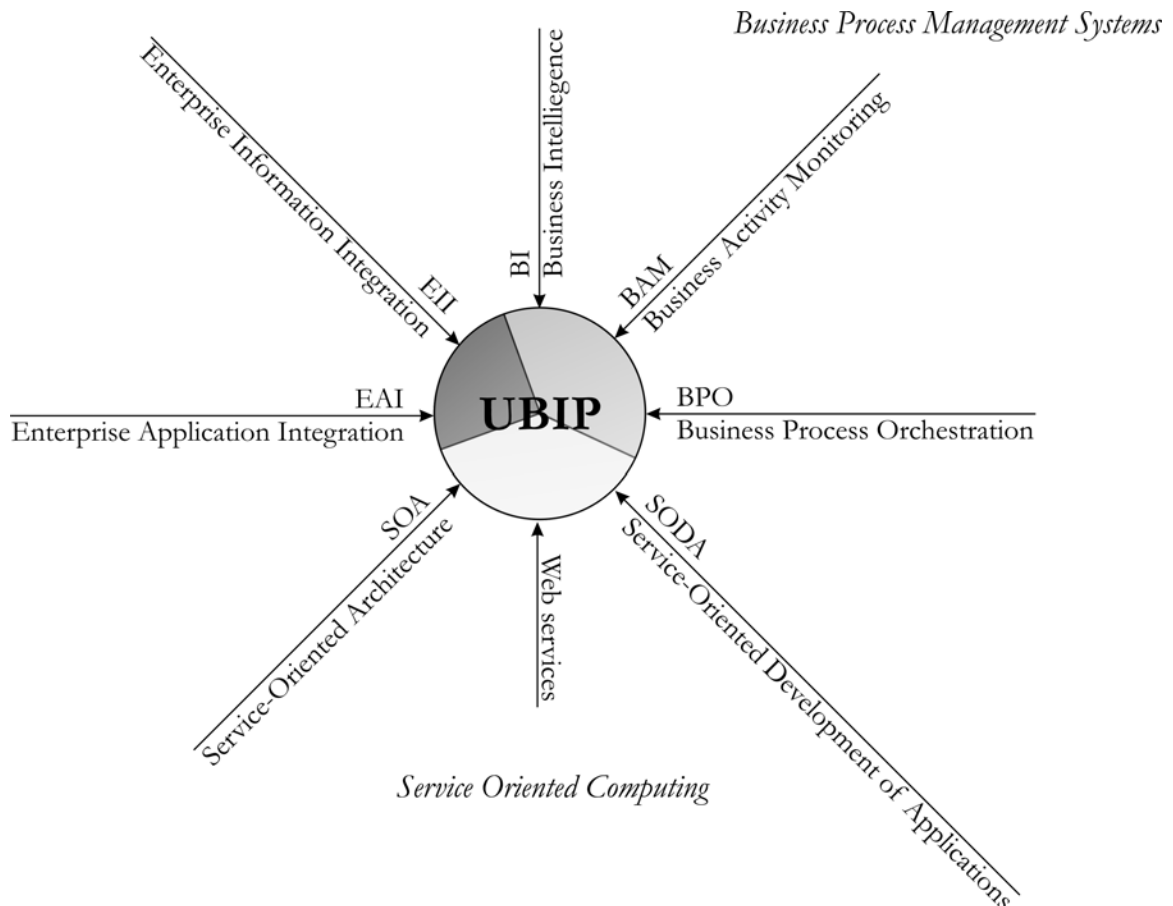


Figure 1.2 The ingredients of a Universal Business Integration Platform

Although the availability of a full-fledged UBIP is still distant, active research in SOC and BPMS has established enough technological underpinnings to allow corporations to begin to explore dynamic business processes in the form of virtual enterprises. Virtual enterprises integrate business functions across organizational boundaries; in other words, business units from various organizations align themselves to perform work. The alignment mechanism is a shared workflow or process definition, which structures the activities performed by the participating partners. The partnering

arrangement within a virtual enterprise is dynamic; partners can change over time in response to environmental changes.

The dynamic and temporal characteristics of workflow processes in a virtual enterprise create many research opportunities and it is against this backdrop that I place my own work. Ultimately, as depicted in Figure 1.3, my research is a synthesis of the many areas of study, each important in its own right.

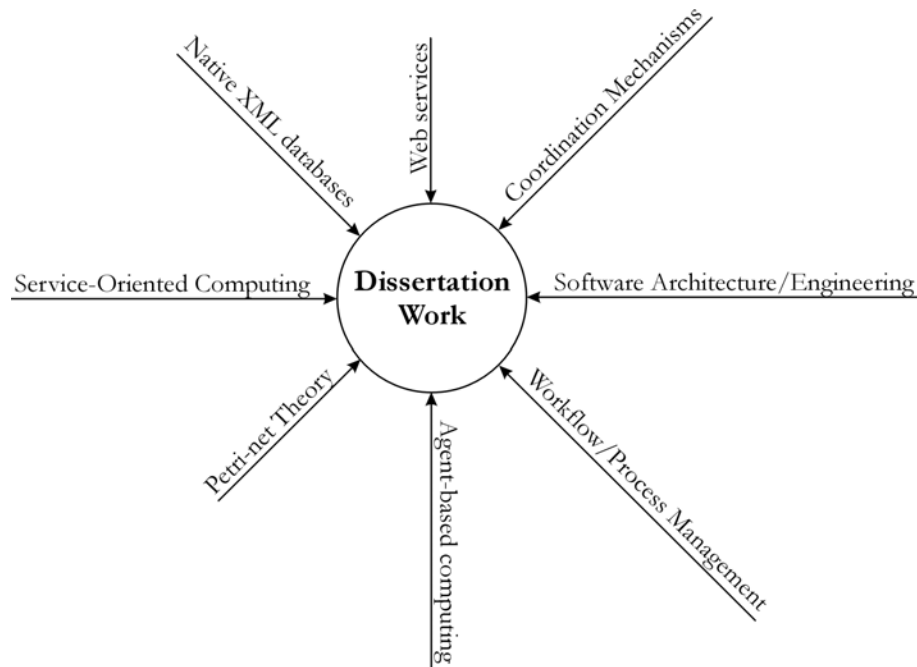


Figure 1.3 The dissertation work as an act of synthesis

As an aid to further explore the nature of the dissertation work, Figure 1.4 uses the Redwine-Riddle software technology maturation phases [6] to categorize concepts and technologies that are relevant to dynamic, agent-based workflow enactment engines. This diagram clearly illustrates that the solution space is complicated due to the integration of several technologies, each of which evolves independently and varies in its level of maturity.

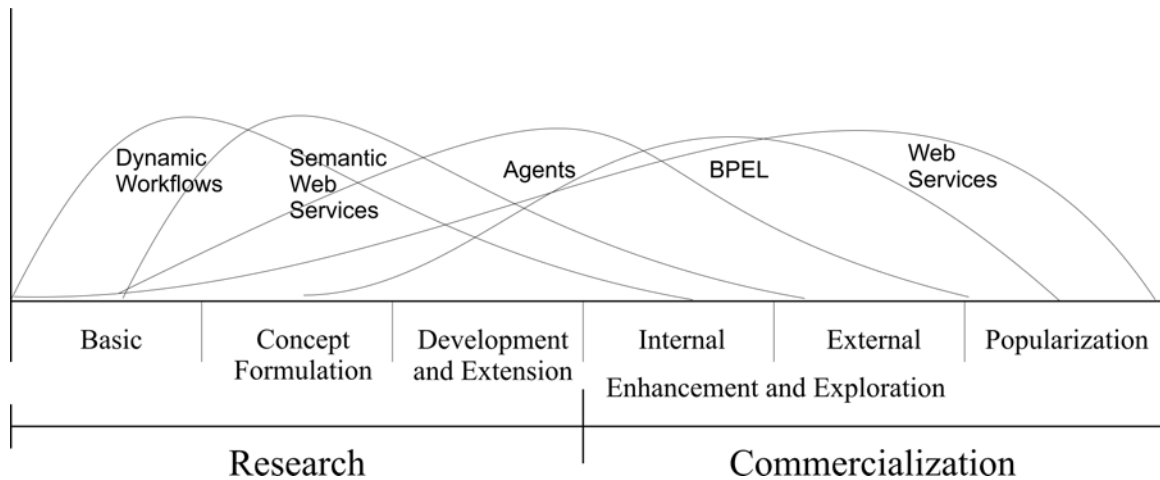


Figure 1.4 Relative maturities of relevant technologies

Another insight that can be gleaned from Figure 1.4 is that traditional top-down research is occurring from the left to right, whereas bottom-up integrative research proceeds from right to left. Both approaches are valuable and essential to the innovation process. Perhaps as a generalization it could be said that science advances from the left, whereas engineering advances from the right. Software engineering, and more specifically software architecture is critical when taking an integrative approach to research. In fact, software architecture research has been defined as “the principled study of the overall structure of software systems, especially the relations among subsystems and components” [7].

1.1 *Research goals and methodology*

This dissertation explores the relationship between software agents and component technologies, specifically Web services. Generally, it discusses mechanisms that allow software agents to be loosely coupled to the behaviors they possess. This loose coupling takes the form of run-time binding to Web services, which define individual behaviors. The long-term benefit of this approach will be that an agent can use its autonomy to

realign its behaviors by binding to alternative Web services in response to environmental dynamics. Likewise, an agent could also have the ability to use a coalition of similar services, in conjunction with a consensus forming mechanism, to establish a behavior that exhibits enhanced robustness and resiliency via redundancy [8].

1.1.1 Hypothesis

The fundamental question behind the work described in this dissertation is whether it is possible to create a software architecture for functionally equivalent workflow enactment, combining agent-based and service-oriented computing concepts and utilizing models of weak coordination and loosely coupled interaction.

Incidentally, an earlier hypothesis of this work was that the relationships between the participant Web services in a business process could be used as a specification of the initial social order of agents in a multiagent system, wherein each agent acts as a proxy for a Web service participating in the process. I have concluded that it will not be possible to establish the true sociality of agents until further advancements in the area of semantically described Web services takes place. For if an agent does not possess a semantically rich description of the behaviors it possesses, it has no basis for negotiating social commitments with other agents.

1.1.2 Research Methodology

At the International Conference of Software Engineering held in 2001, Mary Shaw presented a keynote speech titled *The Coming-of-age of Software Architecture Research*. Her presentation and supporting papers [7, 9] present a classification scheme that captures the methodologies through which software architecture research is conducted. She also provides a framework for addressing whether the combination of research question, research strategy and validation establish compelling results.

The classifications of research questions that Shaw identifies are found in Table 1.1 [9]. The research that I have conducted primarily falls into the *Feasibility* category based upon the research question being asked. Regarding research strategy/results, Shaw's categories are found in Table 1.2 [9]. The research strategy described in this dissertation falls within the *Specific solution* category, and uses the system building approach where the running system embodies the result. Finally, Shaw categorizes validation techniques; some of the defined categories can be seen in Table 1.3 [7]. Of interest to my dissertation work is the *Implementation* category, which captures the essence of the systems that I have constructed.

The research methodology I followed is one of the combinations prescribed by Shaw. Specifically, to address questions of feasibility, use a system building approach and allow the implementation itself to serve as a validation of the work and a confirmation of the hypothesis.

Table 1.1 Research questions in software engineering

Type of question	Examples
Method or means of development	How can we do/create (or automate doing) X? What is a better way to do/create X?
Method for analysis	How can I evaluate the quality/correctness of X? How do I choose between X and Y?
Design, evaluation, or analysis of a particular instance	What is a (better) design or implementation for application X? What is property X of artifact/method Y? How does X compare to Y? What is the current state of X/practice of Y?
Generalization or characterization	Given X, what will Y (necessarily) be? What, exactly, do we mean by X? What are the important characteristics of X? What is a good formal empirical model for X? What are the varieties of X, how are they related?
Feasibility	Does X even exist and if so what is it like? Is it possible to accomplish X at all?

Table 1.2 Research results in software engineering

Type of result	Examples
Procedure or technique	New or better way to do some task, such as design, implementation, measurement, evaluation, selection from alternatives. Includes operational techniques for implementation, representation, management, and analysis, but not advice or guidance.
Qualitative or descriptive model	Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis Well-grounded checklists, well-argued informal generalizations, guidance for integrating other results.
Empirical model	Empirical predictive model based on observed data
Analytic model	Structural model precise enough to support formal analysis or automatic manipulation
Notation or tool	Formal language to support technique or model (should have a calculus, semantics, or other basis for computing or inference) Implemented tool that embodies a technique
Specific solution	Solution to application problem that shows use of software engineering principles – may be design, rather than implementation Careful analysis of a system or its development Running system that embodies a result; it may be the carrier of the result or its implementation may illustrate a principle that can be applied elsewhere
Answer or judgment	Result of a specific analysis, evaluation, or comparison
Report	Interesting observations, rule of thumb

Table 1.3 Validation techniques in software engineering

Technique	Character of validation
Persuasion Technique Design Example	I have thought hard about this, and I believe thatif you do it in the following way, then... ...a system constructed like this would... ...walking through this example show how my idea works.
Implementation System Technique	Here is a prototype of a system that... ...exists in code or other concrete form ...is represented as a set of procedures
Evaluation Descriptive model Qualitative model Empirical quantitative model	Given these criteria, here's how an object rates... ...in a comparison of many objects ...by making subjective judgments against a checklist ...by counting or measuring something
Analysis Analytic formal model Empirical predictive model	Given the facts, these consequences... ...are rigorous, usually symbolic, in the form of derivation and proof ...are predicted by the model in a controlled situation (usually with statistical analysis)

1.2 Overview of the remaining chapters

The organization for the remainder of the dissertation is as follows:

- Chapter 2 provides a critical survey of workflow, workflow description languages, web services and agent technologies. It proposes that workflow description languages and their associated design tools can be used to specify a multiagent system. Specifically, it advances the idea that the Business Process Execution Language for Web Services (BPEL4WS) can be used as a design specification language for a multiagent system, which can then intelligently adapt to changing environmental conditions.
- Chapter 3 details the development of a demonstration system that explores the use of the Web Service Agent Gateway (WSAG) for generating Web service interfaces to software agents. These agent services can then be transparently integrated into BPEL4WS defined workflows. These workflow specifications can subsequently be enacted with existing workflow execution technologies that exhibit properties of strong, centralized coordination. The development of this demonstration system illustrates the power of compositional approaches to system creation. It also serves to reinforce the importance of open standards, since the integration of the separate components is dependent upon the interoperability that standards provide. This work is an important first step toward fully integrated agent-based workflow management systems.
- Chapter 4 describes the development of a distributed, functionally equivalent agent-based workflow enactment mechanism. This system demonstrates that a BPEL4WS workflow description can be viewed as a definition for a multiagent system in which the agents serve as proactive proxies for the underlying passive

Web services. Although the Semantic Web initiative is working toward semantically rich descriptions of Web services, which can be reasoned about by agents, the current state-of-the-art does not yet allow for collections of agents representing semantic Web services to organize themselves to enact workflows. Therefore, this work is critically important as it serves as a bridge from existing, static views of workflow enactment to future, agent-based, dynamic workflow engines.

- Chapter 5 consists of a brief conclusion and discussion of ongoing and future work.

Chapter 2

Towards Adaptive Workflow Enactment Using Multiagent Systems

2.1 Introduction

Advances in Information Technology (IT) are creating opportunities for business enterprises to redesign their information and process management systems. Foundational technologies for a universal enterprise integration platform are emerging. The refinement of service-oriented architectures and the emergence of web-enabled, semantically described services allow us to envision a future where these Web services become the next generation of enterprise components. Recently, the term servicization [1] has been coined to discuss the act of converting existing enterprise applications into Web services. This new enterprise software vision will require new integration strategies. “The traditional programmed interactions between people and software are [being] replaced by task-focused interactions that are dynamic and flexible” [1, pg 216]. This places new demands on software architectures because they will need to support computing with “dynamically-formed, task-specific, coalitions of distributed autonomous resources” [10, pg 99]. These changes are a logical consequence of the seminal work in coordination technology done by Gelertner. As a result of Gelertner’s work, the old computer science adage *Applications = Algorithms + Data Structures* is being replaced by *Applications = Computation + Coordination* [11].

It is now generally accepted that Gelertner was correct when he theorized that computation was orthogonal to coordination [12]. This orthogonality was implied by DeRemer, who wrote in 1976, “Structuring a large collection of modules to form a ‘system’ is an essentially distinct and different intellectual activity from the construction of the individual modules [themselves]” [13]. From these perspectives, a software system is viewed as an ensemble of coordinables and their orchestrated interactions.

Coordinables are entities that function as independent units of computation. The coordinated interaction of the computational units produces the desired behavior of the system. Obvious parallels to workflow systems exist; the workflow activities are the coordinables and business processes coordinate their interaction.

Leymann asserts that workflow construction can be viewed as a two-level programming problem [14, pg 217]. His view is that the implementation of workflow activities is akin to traditional programming, or programming in the small. Activities encapsulate well-defined functionality that typically involves low-level data access routines and algorithmic processing. In contrast, the building of the workflow’s process model is akin to programming in the large. The process model prescribes coordination rules by providing a means to express the sequencing of the activities and the flow of data amongst them.

I advocate the synthesis of Gelertner’s and Leymann’s points of view. I believe that the statements *workflow = activities + processes* and *applications = computation + coordination* are equivalent. The chapter presents my contribution to enterprise integration, the use of multiagent systems for flexible enactment of enterprise workflows. My view can be summarized by the aphorism *Adaptive Workflow Engines = Web*

services + Agents. In this context, the Web services provide the computational resources and the Agents provide the coordination framework. I propose the use of the Business Process Execution Language for Web Services (BPEL4WS) as a specification language for expressing the initial social order of the multiagent system.

In this chapter a brief background of enterprise software is presented. This is followed by a section that examines the relationship between predominate software abstractions and enterprise software architecture. An examination of workflow systems motivates the discussion of BPEL4WS, DAML-S and agents. Finally, the chapter concludes with a discussion of related and future work.

2.2 Enterprise Software

Business enterprises are organizations that perform collective work. In order to achieve necessary operational efficiencies, the work needs to be governed by processes that define the flow of work throughout the organization. Regulated business processes are important because they reduce transactional costs when compared to ad-hoc approaches. In fact, according to Coase [15], the existence of the enterprise itself is dependent upon its ability to achieve lower internal transactional costs than the cost of performing the same work in open markets. Of course enterprises cannot service every need internally, because at some point the overhead burden of these operations exceeds the acquisition cost from the marketplace. When this occurs, businesses form partnerships and cooperative agreements with one another.

Economic arguments and competitive pressures have stimulated business to spend heavily on IT. IT holds the promise of reducing transactional costs via management of business processes and information. Initial IT spending was used to procure enterprise

applications that manage information relating customers, suppliers, partners and employees; these key entities are those with whom the enterprise interacts and transacts. Enterprise applications are categorized by the types of information and interactions they manage. For example, Customer Relationship Management (CRM), Supply Chain Management (SCM) and Enterprise Resource Planning (ERP) are the traditional categories of enterprise applications.

After enterprise applications were in place, IT spending targeted Enterprise Application Integration (EAI) initiatives. EAI allows businesses to further leverage their investment in enterprise software by providing the infrastructure to enable the sharing of data across organizational, system, and application boundaries. EAI improves the visibility and flow of information within the organization, thus increasing the enterprises responsiveness to marketplace demands.


2.2.1 Enterprise Software Architecture Trends

Since enterprise integration solutions are software driven, it is important to examine them within the context of software development abstractions. This discussion will illuminate the interdependence of enterprise software architecture, IT, and software development abstractions.

The evolution of programming paradigms has been occurring for half a century. Programming has transitioned from the earliest days of hard-wired machines to today's component and agent-based approaches. Figure 2.1, a modified version of [16, Table 1], clearly demonstrates that software abstractions have been evolving toward implementation methodologies that feature increasing levels of localization. Localization is a side effect of encapsulation and it is important because it reduces the interdependence between units of code. When units of code are tightly coupled, they become tangled and

difficult to independently deploy. Ideally, software developers work with abstractions that enable the design and implementation of systems with units of code that exhibit loose coupling and high functional cohesion [17]. These features are desirable because they enable greater software reuse.

	Monolithic	Structured	Object Oriented	Component Based	Agent Oriented
How does a unit behave? (Code)	External	Local	Local	Local	Local
What does a unit do when it runs? (State)	External	External	Local	Mixed	Local
When does a unit run? (Control)	External	External	External	External	Local



 Increasing Software Localization

Figure 2.1 Evolution of programming paradigms¹

Over time, enterprise software architecture has evolved in step with IT trends and the influence of software abstractions; Figure 2.2, based in part upon [18, Figure 1], charts this evolution. In the mid-1970's structured programming allowed developers to modularize their software in more controllable ways. In the late 1970's and early 80's, relational databases entered the marketplace. These two developments enabled software architectures that separated the application logic from the data that the application processed. This separation was leveraged by the client/server computing model which became predominate with the introduction of Local Area Network (LAN) technology and the shift toward desktop computing.

¹ Components are predominately stateless; however, stateful components do exist, e.g. stateful session beans. For this reason, State is designated as 'Mixed' in the Component-Based column.

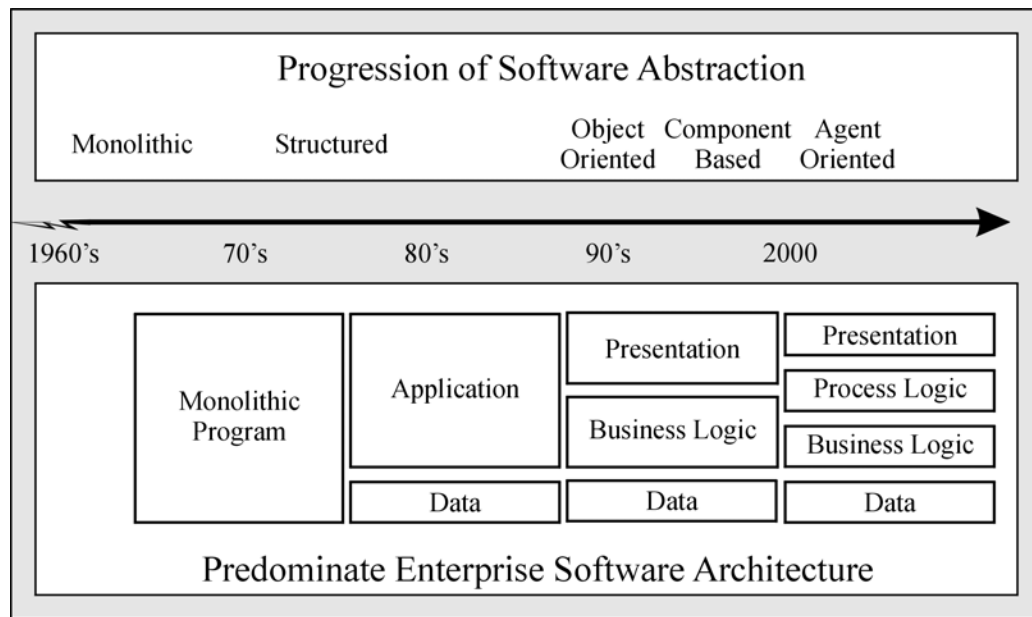


Figure 2.2 Evolution of Enterprise Software Architectures

In the late 80's and early 90's, the widespread adoption of networks and Graphical User Interfaces (GUIs) brought about the next architectural shift. GUI programming employs an event-driven programming model, which is best managed by the Model-View-Controller (MVC) software architecture. MVC enforces separation of concerns: the Model encapsulates the application state which is maintained as data; the Controller defines the application's behavior in response to GUI events; and the View is responsible for the presentation of the model to the user. MVC architectures are deeply rooted in object-oriented technology.

In the late 90's and early 2000's, the Internet became an integration platform for enterprise applications. The MVC architecture is the basis for the familiar N-tier server-side architecture found in today's Internet based enterprise applications. Many of these server-side applications utilize the Java 2 Platform, Enterprise Edition (J2EE™) [19], which provides a component-based modular architecture. As an integration platform, the

Internet has proven to be very flexible. Open standards, which reduce vendor lock-in and increase interoperability, are enabling corporate e-business initiatives. As businesses integrate across organizational boundaries, it becomes important to separate the ‘public’ process logic from the ‘private’ business logic. The process logic specifies the order and conditions under which things get done; whereas, the business logic specifies what gets done. Business Process Management (BPM) software is an emerging classification of integration software that treats business processes as first-class entities.

In Figure 2.1, it can be seen that the agent-oriented software abstraction is destined to have an impact on enterprise software architectures. Currently, much attention is being focused on Web services and their suitability to BPM applications. A closer examination of Web services in the context of enterprise software follows.

2.2.2 Web services as Enterprise Software Components

Software components are created in order that they may be composed. As stated by Szyperski [20], “Composition enables prefabricated ‘things’ to be reused by rearranging them in ever new composites.” Composing an application via reuse of existing software assets can dramatically reduce the development time. Likewise, the quality of the application will also increase, if the reused software has previously been proven through testing and successful deployment. Obviously any software engineering technique that has the ability to reduce development time while simultaneously increasing the quality of the product is noteworthy.

The range of Component-Based Software Engineering (CBSE) practice can be constrained by the definition of a software component. The definition of a software component is hotly debated; a sampling of common definitions can be found in [20-23]. For the purposes of this chapter, the definition presented by Heineman will be used. This

definition was selected for several reasons: it has undergone extensive review and revision; the definition is architecturally neutral in that it does not favor any specific implementation language or component model; and it is abstract enough to be inclusive of the other commonly referenced definitions of a software component. The software component definition found in [21, pg 7] is:

A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

A component model consists of a collection of standards that govern the interaction and composition of software components that conform to the model. Standards are essential to the concept of open systems, which are simply a collection of interacting software and hardware components. The interaction within the open system is defined by interface specifications that are complete, publicly available and non-proprietary [24]. Table 2.1 [25] summarizes the basic elements of a component model.

From a workflow perspective, a composite software system can be viewed as a sequence of services operating upon data. Ideally these services should be language, platform and location independent [26]. Such services would then be interoperable, where interoperability is characterized by the “ability of two or more software components to cooperate despite differences in language, interface, and execution platform” [27]. Web services represent a new class of interoperable, web-enabled software service. Several specifications have been developed that form the basis of a component model for Web services; specifically, SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery, and Integration). These specifications are used to invoke,

describe, publish, and discover Web services. They also embrace an open systems point of view: XML (eXtensible Markup Language) is utilized to exchange data in a neutral format and component communication occurs via open transport protocol like HTTP.

Table 2.1 Component Model Standards

Standards for	Description
Interfaces	Specification of component behavior and properties; definition of Interface Description Languages (IDL)
Naming	Global unique names for interfaces and components
Meta Data	Information about components, interfaces, and their relationships; APIs to services providing such information
Interoperability	Communication and data exchange among components from different vendors, implemented in different languages
Customization	Interfaces for customizing components. User-friendly customization tools will use these interfaces
Composition	Interfaces and rules for combining components to create larger structures and for substituting and adding components to existing structures
Evolution Support	Rules and services for replacing components or interfaces by newer versions
Packaging and Deployment	Packaging implementation and resources needed for installing and configuring a component

Two important features of Web services are that they have a network-addressable interface and they can be used to represent business concepts or services. These two characteristics are essential to the concept of enterprise components as introduced in [22]. Web services have become the enterprise components for the next generation of enterprise-level software and are now viewed as the new building blocks for enterprise software systems. As with any building block metaphor, the challenge is how to arrange or compose the building blocks into larger structures. As indicated in Table 2, composition standards are a critical element of a robust component model. From the enterprise application perspective, compositions of Web services can be viewed as a workflow; the next section will introduce appropriate workflow concepts.

2.3 Workflow Management Systems

The Workflow Management Coalition (WfMC) is an international standards-setting organization of workflow vendors, users, analysts and university/research groups. The WfMC has been responsible for the creation of a workflow reference model and a glossary of standardized workflow terminology. These resources will be used to define workflow concepts with more precision. Several key terms are important to understanding the nature of workflow and to provide an underpinning for a discussion of contemporary trends in workflow management systems. The WfMC Terminology and Glossary [28] document provides the following definitions:

Workflow – *the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*

Business Process – *a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.*

Process Definition – *the representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.*

Workflow Management System – *a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.*

In summary, a process definition is an abstract representation of a business process that can be consumed by a workflow management system in order to enact the workflow.

2.3.1 Adaptive Workflow in Context

Traditionally, workflow management systems have not been designed for dynamic environments requiring adaptive response. Currently, the need for adaptive workflow is being driven by the demands of e-commerce in both B2B and B2C space. Initial B2B automation activities were centered on Electronic Data Interchange (EDI) initiatives. More recent work in the B2B space has focused on the development and deployment of ebXML (electronic business XML). With both EDI and ebXML the collaborating business partners predefine the terms of their electronic interaction. As discussed by Jenz, these technologies enforce regulated B2B interaction and as such, they create closed communities of business partners [29]. In comparison, views toward virtual organizations require flexible, on-the-fly alignment of business partners; in other words, adaptive workflow capabilities. These loose collaborations of business partners operate in open, non-regulated B2B/B2C scenarios [29]; intuitively, pre-negotiated collaboration agreements are a hindrance in these environments.

Available workflow management systems span a range of capability. This is not surprising since businesses in any segment can benefit from workflow management. For example the insurance industry benefited greatly from document management systems, which reduce physical paperwork, increase the availability of documents, and control the flow of information during processing. As shown in Figure 2.3(a), adapted from [14, pg 10], workflows can be broadly categorized by their business value and repetition rates. Focusing on the two highest value workflow categories, Figure 2.3(b), a modification of [30, 193], differentiates collaborative and production-oriented workflows.

Collaborative and production-oriented workflows are distinguished by measures of structure and centricity. Collaborative workflows are information centric. Typically, human interpretation of information drives the workflow in a loosely structured manner. Collaborative workflows are sometimes described by the term Computer Supported Cooperative Work (CSCW); groupware and other shared workspace tools are often the vehicles for CSCW. In comparison, production workflows are process driven due to their highly repetitive nature. To achieve the efficiency required of production workflow, the processes are highly structured. Today's agile manufacturing environments are controlled by Manufacturing Execution Systems (MES) that schedule production based upon highly detailed process plans. As indicated in Figure 2.3(b), the requirements of adaptive workflow fall between these two broad categories.

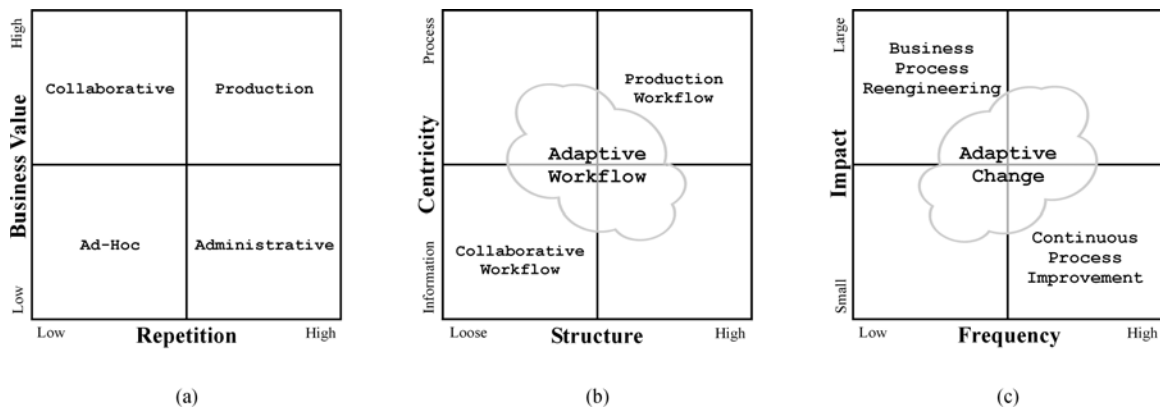


Figure 2.3 Workflow Perspectives

Adaptive workflows need to react to changing environmental conditions. Currently, businesses change their workflows through two primary mechanisms: Business Process Reengineering (BPR) and Continuous Process Improvement (CPI). Figure 2.3(c), adapted from [30, pg 239], illustrates the difference between BPR and CPI. BPR is the periodic analysis and subsequent redesign of the intra- and inter- business processes used by an

organization. BPR is used to overhaul processes in order to create operational efficiencies that improve quality and save time and cost. Conversely, CPI focuses on continuous improvement through the application of small and orderly changes. Workflows are continuously examined in order to find ways to increase quality and reduce waste. Adaptive workflows respond to changing conditions through adaptive change. As shown in Figure 2.3(c), adaptive change is not constrained by measures of frequency or impact.

Current workflow initiatives have embraced the Web service model. Given the current state of technology, Web service based workflows typically are deployed behind corporate firewalls and are used for intra-organizational workflow. The reason for this is that Web service specifications are weak in regards to issues of security, transaction management, internationalization, et al. Inevitably, as standards evolve to address these deficiencies, workflows will transition from the domain of intranets to that of the Internet. This transition will be accompanied by a new set of problems.

When an intranet-based workflow system executes, it does so with a collection of services that are owned and managed by the same organization. In this environment, service interruptions are infrequent and typically scheduled due to consolidated system management. In contrast, Internet-based workflows must be designed for resilient operation as service partners periodically become unavailable due to decentralized system management and the lack of network service guarantees. The evolution from intra- to inter- net based workflows will increase the design and run-time complexity, since the coordination mechanism must become more fault tolerant.

2.3.2 Workflow Reference Model

The Workflow Reference Model describes a generic architecture for workflow management systems [31]. The model, depicted in Figure 2.4, shows the functional

components of a workflow system and identifies the major interfaces between them

Although the workflow reference model was created in 1995, it still provides a relevant architecture for discussing workflow management systems today.

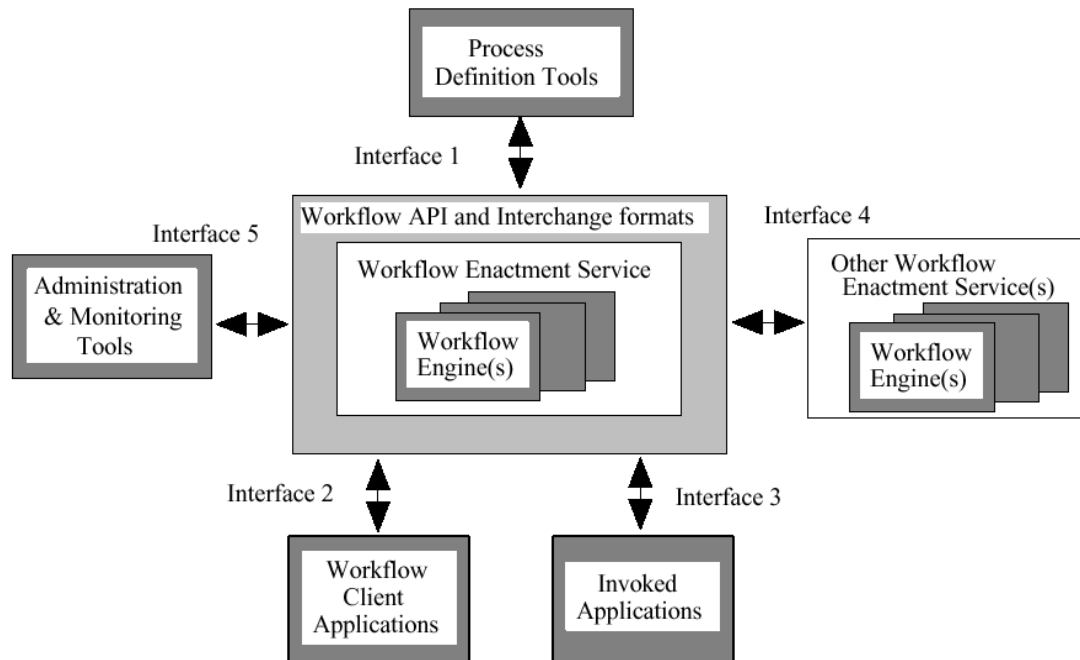


Figure 2.4 WfMC Reference Architecture © WfMC

As illustrated, a process definition of a business process is generated by Process Definition Tools. These tools typically allow a process designer to create a diagrammatic representation of business processes. The diagrams are then saved in a Process Description Language (PDL) that is received by the Workflow Enactment Service via Interface 1. The enactment services utilize one or more local workflow engines that interpret and execute the PDL. While the workflow is being enacted, it can be administered and monitored via interface 5. The executing business process may interact with other automated business processes via interface 4. Interface 2 provides a mechanism for engaging a human participant in the workflow process, whereas interface 3 is designed for invoking applications without human intervention.

2.3.3 *Workflow Tools*

As discussed in section 2.2, enterprise software architecture has evolved over time. Likewise, according to Singh and Huhns, the concept of workflow technology has developed in a generational manner; each generation leveraging the computational capabilities and management theories of their time. They identify four generations of workflow advancement and project a fifth generation based upon agent technology. The four generations they identify are: manual, closed, database-centric, and workflow tools oriented [32]. Contemporary approaches to business process automation are primarily focused on 4th generation approaches, that is, they are workflow tools oriented.

The majority of workflow tools target process definition. Business processes can be complex and difficult to comprehend. If a business process is incomprehensible, it cannot be effectively communicated, analyzed nor checked for correctness or completeness. UML activity diagrams can be used to represent workflows. These diagrams provide a high-level graphical description of the various task dependencies. The diagrams were designed to be human-readable. The activity diagram notation supports the expression of concurrent activity flows, the use of conditional branching, and the mapping of activities to specific actors. However, activity diagrams are insufficient for the purposes of enactment by an automatic system.

In the past few years, PDLs have been heavily influenced by XML and Web services. There are several ongoing initiatives that are defining XML-based PDLs to describe workflows composed of Web services. Although XML is human readable, its strength is in providing an unambiguous mechanism for the exchange of information. Fortunately, workflow design tools insulate the modeler from the complexity of the underlying PDL. The current generation of workflow design tools allows the process

modeler to generate executable workflows directly from their visual representation.

BPEL4WS, an XML-based language for expressing the composition of Web services, is poised to become the target PDL for the next generation of workflow design tools.

2.4 BPEL4WS

During the summer of 2002, IBM, Microsoft and BEA released a new PDL named BPEL4WS [33]. BPEL4WS represents the merger of two other PDLs, IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG. BPEL4WS provides both graph-based and block-based control structures, making it capable of representing a wide range of control flows. Aalst has compared the expressiveness of several PDLs, and has confirmed that BPEL4WS represents the union of WSFL and XLANG [34]. This merger has created the market consolidation necessary to make BPEL4WS the de facto standard for expressing workflows consisting of Web services.

BPEL4WS can be used to describe executable business processes and abstract processes. Abstract processes are used to create behavioral specifications consisting of the mutually visible messages exchanged between transacting parties executing a business protocol. BPEL4WS relies upon the following XML-based specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing.

Structurally, a BPEL4WS file describes a workflow by stating whom the participants are, what services they must implement in order to belong to the workflow, and the control flow of the workflow process. The BPEL4WS process model is built on top of the WSDL 1.1 service model and assumes all primitive actions are described as WSDL portTypes. That is, a BPEL4WS description describes the choreography of a set of messages all of which are described by their WSDL definitions. Importantly, WSDL is

also used to describe the external interface to the workflow. This allows BPEL4WS to be compositionally complete, which means that the composition of Web services are exposed as a single Web service eligible to participate in other compositions [11].

Since BPEL4WS is highly dependent upon WSDL, a closer examination of the structure of a WSDL file is in order. Figure 2.5 presents the contents and an explanation of a WSDL file for a publicly available currency exchange rate Web service.

<p><definitions> specifies one or more services.</p> <p><types> section provides information about any complex data types used in the document. Note: the <types> section is not present in this example because only simple types are used.</p> <p><message> is an abstract definition of the data being communicated. This Web service defines two messages: getRateRequest and getRateResponse.</p> <p><portType> provides an abstract set of operations supported by the endpoints.</p> <p><operation> describes the action provided by the service. This service has an operation named getQuote that takes a getRateRequest message and returns a getRateResponse message.</p> <p><binding> describes how the operation is invoked. It specifies the protocol and data format for the operations and messages.</p> <p><service> specifies the port address(es) of the binding.</p> <p><port> defines a communication endpoint.</p>	<pre> <?xml version="1.0"?> <definitions name="CurrencyExchangeService" targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl" xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/"> <message name="getRateRequest"> <part name="country1" type="xsd:string"/> <part name="country2" type="xsd:string"/> </message> <message name="getRateResponse"> <part name="Result" type="xsd:float"/> </message> <portType name="CurrencyExchangePortType"> <operation name="getRate"> <input message="tns:getRateRequest" /> <output message="tns:getRateResponse" /> </operation> </portType> <binding name="CurrencyExchangeBinding" type="tns:CurrencyExchangePortType"> <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/> <operation name="getRate"> <soap:operation soapAction="" /> <input> <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </input> <output> <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </output> </operation> </binding> <service name="CurrencyExchangeService"> <documentation> Returns the exchange rate between two currencies </documentation> <port name="CurrencyExchangePort" binding="tns:CurrencyExchangeBinding"> <soap:address location="http://services.xmethods.net:80/soap"/> </port> </service> </definitions> </pre>
--	---

Figure 2.5 A WSDL file for a currency exchange rate Web service

Figure 2.6 presents a skeletal view of the primary sections of a BPEL4WS file. The <partners> section declares the different parties that participate in the workflow process. Each partner is given a service link type and the role it will perform as part of the service link. The service link types are named entities that provide a mapping

between role names and WSDL portTypes that the role must support. The `<variables>` section defines the variables used by the process. Variables store past messages and are needed to maintain the state of the workflow process as it executes. The `<faultHandlers>` section describes the fault handlers used by the workflow. All faults generated by the workflow must be given a name. The fault handlers define the activities that will be performed when a fault is raised. The process definition of the workflow occurs after the fault handlers section and before the close process tag.

<p><process> specifies a process.</p> <p><partners> section declares the different parties that participate in the workflow. <partner> defines a workflow participant</p> <p><variables> section defines the data variables used by the process. <variable> defines a named variable, associated with a WSDL message type.</p> <p><faultHandlers> section which defines the activities that should execute in response to a fault which occurs during the enactment of the process. <catch> handles a specific fault <catchAll> default handler for faults that are not specifically caught.</p>	<pre> <process> <partners> <partner/> </partners> <variables> <variable/> </variables> <faultHandlers> <catch/> <catchAll/> </faultHandlers> <!-- Workflow Definition Occurs Here --> </process> </pre>
--	---

Figure 2.6 Primary BPEL4WS Sections

A workflow process is defined in BPEL4WS using activity constructs. The primary constructs are:

- `<sequence>` specifies that its contents must be executed in the order presented;
- `<flow>` specifies that the contents are executed in parallel;
- `<while>` indicates that an activity is repeated until a given criteria has been met;
- `<switch>` allows the conditional execution of one of many activities;
- `<pick>` blocks until a specified message arrives or a time-out occurs; when either one occurs its associated activity is executed;
- `<receive>` designates that a message is to be received;

`<reply>` sends a message;
`<invoke>` construct invokes an operation on a specified partner, portType pair;
`<throw>` is used to raise a fault;
`<wait>` pauses the execution for a specified time

Figure 2.7 contains a sample BPEL4WS workflow definition with the structuring activity tags in boldface. This service provides the ability to obtain a stock quote from the NYSE in any currency. For example, the service could be called to obtain a quote for shares of IBM expressed in Swiss Francs. An analysis of this workflow reveals that the service is composed from three other Web services. The three external Web services are the stockQuoteProvider, the currencyExchangeProvider, and the simpleFloatMultProvider. A narrative description of the steps in the workflow follows:

1. a service request message is received from the requestor.
2. the stock symbol is copied from the request message and the stockQuoteProvider is invoked to obtain the quote.
3. the country name is copied from the request message and the currencyExchangeProvider is invoked to obtain the rate of exchange between US dollars and the foreign currency
4. the returned quote and the exchange rate are sent to the simpleFloatMultProvider, which multiplies these values to convert the quote to the foreign currency
5. the converted quote is copied into the reply message which is sent to the service requester.

Note that steps 2 and 3 have no data dependencies; therefore they can execute concurrently. This is specified in the BPEL4WS by placing steps 2 and 3 within a `<flow>` block.

```

<sequence>
  <receive name="request" partner="requestor"
    portType="tns:stockLookupProcessPT" operation="requestLookup"
    variable="request" createInstance="yes">
  </receive>
  <flow>
    <assign>
      <copy>
        <from variable="request" part="symbol"/>
        <to variable="stockQuoteProviderRequest" part="symbol"/>
      </copy>
    </assign>
    <invoke name="getStockQuote" partner="stockQuoteProvider"
      portType="ns1:StockQuotePortType" operation="getQuote"
      inputVariable="stockQuoteProviderRequest"
      outputVariable="stockQuoteProviderResponse">
    </invoke>
    <assign>
      <copy>
        <from expression="'usa'"/>
        <to variable="currencyExchangeProviderRequest" part="country1"/>
      </copy>
      <copy>
        <from variable="request" part="country"/>
        <to variable="currencyExchangeProviderRequest" part="country2"/>
      </copy>
    </assign>
    <invoke name="getCurrencyExchange" partner="currencyExchangeProvider"
      portType="ns2:CurrencyExchangePortType" operation="getRate"
      inputVariable="currencyExchangeProviderRequest"
      outputVariable="currencyExchangeProviderResponse">
    </invoke>
  </flow>
  <assign>
    <copy>
      <from variable="stockQuoteProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f1"/>
    </copy>
    <copy>
      <from variable="currencyExchangeProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f2"/>
    </copy>
  </assign>
  <invoke name="doSimpleFloatMult" partner="simpleFloatMultProvider"
    portType="ns3:SimpleFloatMult" operation="multiply"
    inputVariable="simpleFloatMultProviderRequest"
    outputVariable="simpleFloatMultProviderResponse">
  </invoke>
  <assign>
    <copy>
      <from variable="simpleFloatMultProviderResponse" part="multiplyReturn"/>
      <to variable="response" part="Result"/>
    </copy>
  </assign>
  <reply name="response" partner="requestor"
    portType="tns:stockLookupProcessPT" operation="requestLookup"
    variable="response">
  </reply>
</sequence>

```

Figure 2.7 An Example BPEL4WS Workflow Description

2.5 DAML-S

As compelling a technology as BPEL4WS is, it does not fully address the challenge of composing workflow systems from Web services. BPEL4WS is focused on expressing the mechanics of the workflow, but not its semantics. Web services are compatible with Service-Oriented Architectures (SOA), which support run-time discovery and invocation. Ultimately, SOAs require semantic service descriptions in order to be broadly functional.

Figure 2.8 illustrates the elements of a SOA.

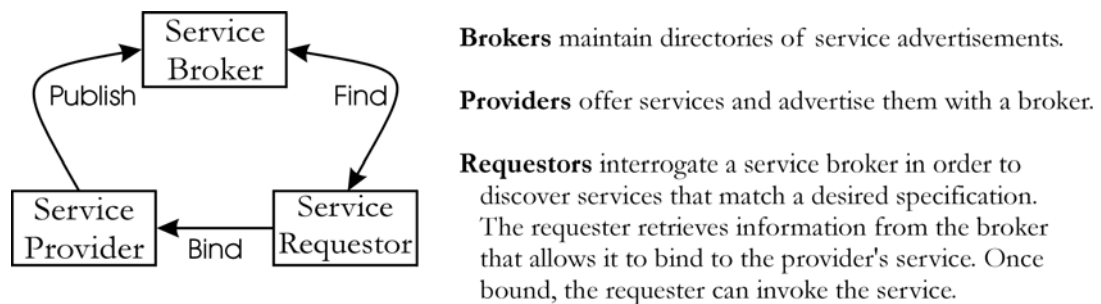


Figure 2.8 Service-Oriented Architecture Model

The semantic web initiative is developing technologies for locating web resources based upon their semantic content. Included in this vision is DAML-S, a specification for providing semantic markup for Web Services. DAML-S is being designed to support the following Web Service related tasks: discovery, invocation, composition and interoperation, and execution monitoring [35]. DAML-S provides a machine-interpretable, ontology-backed semantic description of both atomic and composite Web services. As stated in [36], the markup provides:

declarative advertisements for service properties and capabilities which can be used for automatic service discovery;

declarative APIs for individual Web Services that are necessary for automatic Web Service execution; and

declarative specifications of the prerequisites and consequences of

individual service use that are necessary for automatic service composition and interoperation

As discussed, DAML-S is being designed to ease composition issues by providing information that will allow Web Services to smartly interoperate. For a discussion of the relationship of DAML-S to other standards like UDDI, WSDL, and ebXML see [37].

A short example will provide justification of the need for a semantic markup language like DAML-S. Consider the following use-case scenario: “A London-based firm that wants to automate purchasing. Upon execution of an initial request for the purchase of 1,000 units of some item, the resulting query passes over the network to a Web service run by the supplier in Germany. That supplier in turn calls a Web service that calculates shipping cost, a second that computes tax, and a third that converts between pounds and euros. The German supplier then returns a consolidated quote to the original caller in London” [38]. Intuitively, an automated workflow engine could make use of the CurrencyExchangeService Web service, corresponding to the WSDL file presented in Figure 2.5, to convert the price from euros to pounds. Or could it?

Without DAML-S or some other semantic annotation, the answer is no. Even if the service were located via a UDDI search, an automated workflow engine would not know what data to provide upon invocation. To the human reader, the names (country1 and country2) and the types (string) of the arguments in the getRateRequest message provides some contextual clues as to what these variables should contain; however, to an automated process this syntactic information is meaningless. However, if the service was described with a semantically rich notation, it might declare something like:

CurrencyExchangeService is a Web service that accepts two ISO 3166-1-Alpha-2 country codes and returns an exchange rate whose value is a floating point number.

Given the proper ontological structures, it would be possible to perform automated reasoning. From the structure of the requester's address, it would be likely to conclude that London is a city. Inference rules that establish that cities are found in countries could derive that London is found in a country named United Kingdom. Finally, countries have attributes and in my ontology, one of them is the ISO 3166-1-Alpha-2 country codes, which for the United Kingdom is GB.

Semantic markup of the service also provides information about what function the service performs. If the workflow enactment mechanism had knowledge about the role the CurrencyExchangeService played in the overall process, it would be possible to adapt the workflow in intelligent ways. If the German parts supplier in my example were supplying parts to another company in Germany, the CurrencyExchangeService obviously would not be required. A more sophisticated level of reasoning would need to occur if the German company was selling parts to a customer in France. In the ontology, countries also have an attribute of ISO 4217 Currency Type Symbol. This information could provide the adaptive workflow mechanism knowledge that both Germany and France use the same EUR currency type; therefore invoking the CurrencyExchangeService would be an unnecessary operation.

Given the demonstrated importance of semantic markup, it is worth considering the relationship between DAML-S and BPEL4WS. The DAML-S service model overlaps with BPEL4WS functionality; specifically, the Process Ontology contains the concept of a Composite Process. In DAML-S, composite processes can be recursively decomposed into a set of Web services that are related to one another via control constructs. The DAML-S control constructs are block-structured and therefore lacks the ordering

flexibility provided by BPEL4WS links. I suggest that BPEL4WS and DAML-S are compatible due to the fact that BPEL4WS is compositionally complete. BPEL4WS exposes a single WSDL interface for the composite process it contains and could therefore be marked-up in DAML-S as an atomic process. This results in the composite process itself, rather than its internal processing being described in DAML-S. This is the same mechanism used by DAML-S for handling DAML-S composite processes. A DAML-S composite process is transformed into a simple process via the *collapse* property. The simple process is mapped to an atomic process by the *realizedBy* property. In DAML-S, atomic processes are grounded to WSDL operations [35].

2.6 Agent-based Workflow Approaches

In my earlier work [39], I established a relationship between semantically described Web services and agents. My vision is to use Semantic Web services as external behaviors for proactive agents. Huhns further distinguishes between Web services and agents. Some of the distinctions he provides are: Web services know only about themselves, they do not possess any meta-level awareness; Web services are not designed to utilize or understand ontologies; Web services are not capable of autonomous action, intentional communication, or deliberately cooperative behavior [40]. In contrast, agents possess all of these capabilities.

As previously introduced, Singh and Huhns, anticipate that 5th generation workflow systems will employ agent-based technologies [32]. Others share this view, specifically [41-43]. To place this in perspective, an agent is a system that exhibits properties like: situatedness, autonomy, reactivity, pro-activeness, and social ability [44]. The social metaphor gives power to the agent-oriented paradigm; it is one of the characteristics that

makes the agent abstraction particularly suitable for developing complex, distributed systems [45, 46]. If a collection of sociable agents, representing individual services, cooperate and coordinate they would have the capability to enact any workflow that is composed of the represented services. In other words, agents have the capability to dynamically form social structures through which they share commitments to the common goal of workflow enactment. The individual agents, through their coordinated interactions achieve globally coherent behavior; they act as a collective entity known as a multiagent system.

Workflow enactment by a multiagent system can be viewed as an act of cooperative problem solving. “Cooperative problem solving occurs when a group of autonomous agents choose to work together to achieve a common goal” [47]. For cooperative problem solving to occur, an agent in the multiagent society must recognize that the best path to achieving a goal is to enlist the help of other agents. Social commitments arise when one agent makes a commitment to another. Typically a social commitment comes about due to a social dependency. As defined in [48, pg 113] a social dependence can be defined as:

$$(SocialDependence\ x\ y\ a\ p) \equiv (Goal\ x\ p) \wedge \neg(CanDo\ x\ a) \wedge (CanDo\ y\ a) \wedge ((DoneBy\ y\ a) \Rightarrow Eventually\ p)$$

[Meaning] agent x depends on agent y with regard to act a for realizing state p, when p is a goal of x and x is unable to realize p while y is able to do so.

As indicated, for such a social dependency to be established, agent x and agent y must be able to reason about their ability to perform act a , and have knowledge that the performance of a will establish state p . The concept of first-order ability states that for agent x to have first-order ability regarding the establishment of state p , it must know

explicitly whether $\exists a((\text{CanDo } x, a) \wedge ((\text{DoneBy } x a) \Rightarrow \text{Eventually } p))$ [47, pg 150].. If agent x desires to achieve state p , but knows $\neg(\text{FirstOrderAbility } x, p)$, then it must solicit assistance in order to attain the goal.

I believe that the advent of the semantic web and the emergence of a Web Services component model can facilitate agent-based workflow management in dynamic real-time environments. If agents use semantically described Web Services, then the semantic service descriptions become the basis for determining the agent's first-order abilities. Likewise, a common semantic markup for Web Services will facilitate effective communication between agents. Social agents that have access to an ontology-backed semantic description of their behaviors should be better able to proactively coordinate themselves at the macro-level.

2.6.1 BPEL4WS for multiagent systems.

Unfortunately, despite the promise of the semantic web, its application in deployable, commercial applications is still distant. Given this fact, is the concept of using multiagent systems for workflow enactment still viable? The answer to this question is a resounding yes, for even without semantic information, agents can employ their autonomy to perform local workflow optimizations. Of course, a troubling issue remains. As discussed in the previous section, agents need semantic descriptions of their behaviors in order to reason about their own capabilities and the capabilities of the other agents in the system. This reasoning ability is crucial for cooperative problem solving to occur. If agents are unaware of their first-order ability, how is it possible for a multiagent system to organize itself?

I propose a novel approach, which is to use a BPEL4WS process description to impose an initial social order upon a collection of agents. Since BPEL4WS describes the

relationships between the Web services in the workflow, agents representing the Web services would know their relationships a priori. Notably, the relationships between the Web services in the workflow are embedded in the process logic of the BPEL4WS file. A mechanism to extract this relational information is required if it is to be used to coordinate the interactions of the agents. My strategy is to construct a Petri Net (PN) for the workflow, which is then partitioned based upon partner information. Agents within a multiagent system represent each partner and enact the workflow in a distributed manner.

PNs are recognized as a useful workflow modeling tool. The reader is referred to [49] for a collection of PN related material. Their unambiguous and precise semantics allow a workflow model to be analyzed. Such analysis can prove many properties about a workflow process, including the absence of livelock and deadlock conditions. For my purposes, an assumption is made that the BPEL4WS process description represents a well-formed workflow process. Aalst presents a methodology to generate PNs via the repetitive replacement of elemental PNs with other PNs [30]. In order to leverage this replacement property, a collection of elemental building blocks is required. Figure 2.9 based in part on [30, Figure 4.11], illustrates a collection of PN building blocks along with a mapping to the appropriate BPEL4WS activity.

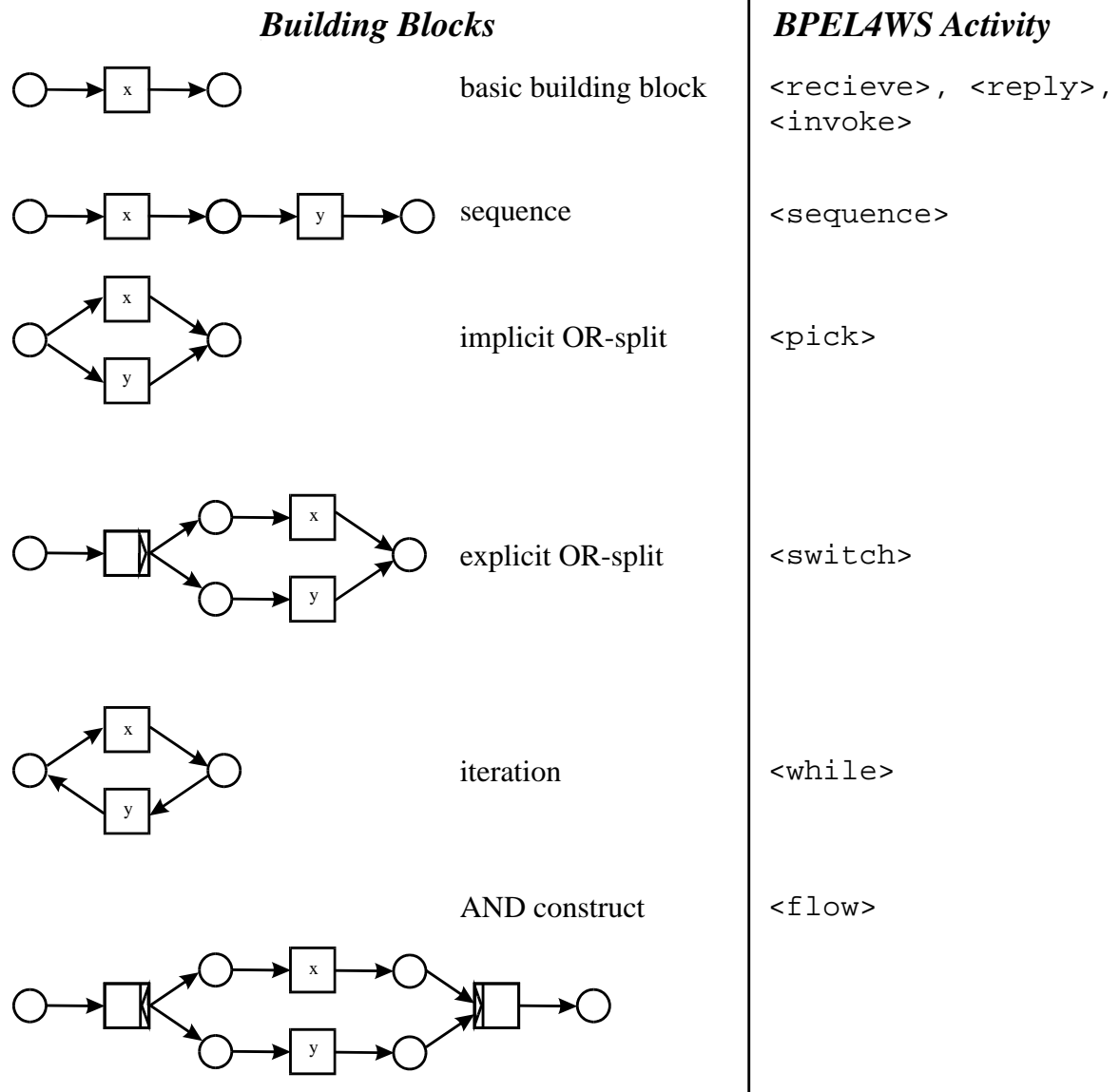
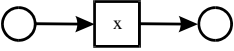
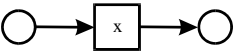
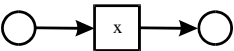
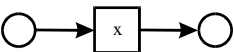
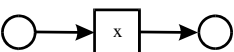


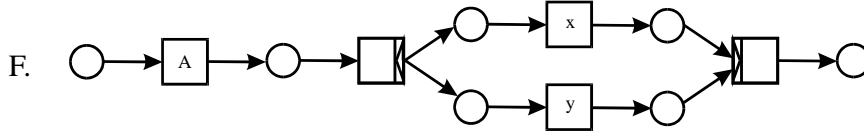
Figure 2.9 Building Blocks Mapped to BPEL4WS Activities

Although the mapping from building blocks to BPEL4WS activities is incomplete, it is sufficient to represent many workflows. It is my opinion that after the basic workflow process is converted to PN form, it can be augmented with the <link>, <terminate>, <wait>, et al. activities. To demonstrate the application of the replacement property, the workflow description found in Figure 2.7, will be used to illustrate the conversion of

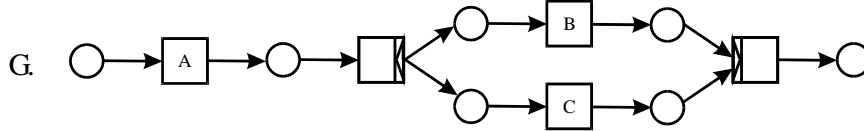
BPEL4WS to PN form. To begin, represent each BPEL4WS basic activity as a building block.

- A.  where x is the <receive> activity named request.
- B.  where x is the <invoke> activity named getStockQuote.
- C.  where x is the <invoke> activity named getCurrencyExchange.
- D.  where x is the <invoke> activity named doSimpleFloatMult.
- E.  where x is the <reply> activity named response.

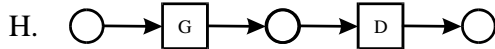
Utilizing the mapping presented in Figure 8, repeatedly apply the replacement property, consuming the BPEL4WS process description. Starting with a sequence building block, replace x with A and replace y with a flow construct, yielding F.



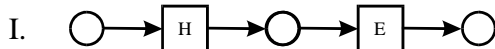
Starting with F, replace x with B and replace y with net C, yielding G.



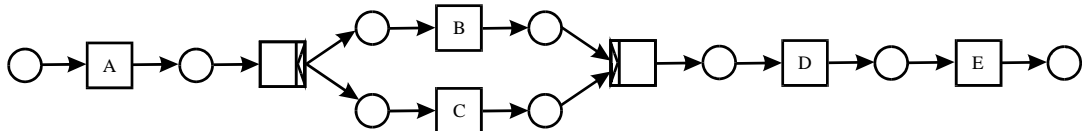
In a sequence building block, replace x with G and y with D, yielding H.



In a sequence building block, replace x with H and y with E, resulting in I.



The final PN in expanded form:



After the BPEL4WS file has been converted to PN form, it can be decomposed to establish the relationships between the Web services. This is accomplished by traversing

the PN in the forward direction, noting the input and output tokens consumed and sent by each partner. This information, along with variable manipulation instructions is sufficient to allow the agents to coordinate the enactment of the workflow.

The architecture for the multiagent enactment of the example workflow is illustrated in Figure 2.10. This architecture relies upon the work of The Foundation for Intelligent Physical Agents (FIPA), which can be thought of as a component model that enables agents from heterogeneous origins to collaborate in open environments [50]. In this architecture, the following communications pathways exist:

- agent to agent communication occurs via FIPA's Agent Communication Language (ACL) and is facilitated by a FIPA compliant Agent Management System (AMS).
- agent to Web service communication is accomplished via SOAP messages.
- agent to shared dataspace communication utilizes appropriate protocols/interfaces provided by the dataspace. The dataspace is used to store BPEL4WS process variables, which maintain the state of the workflow process.

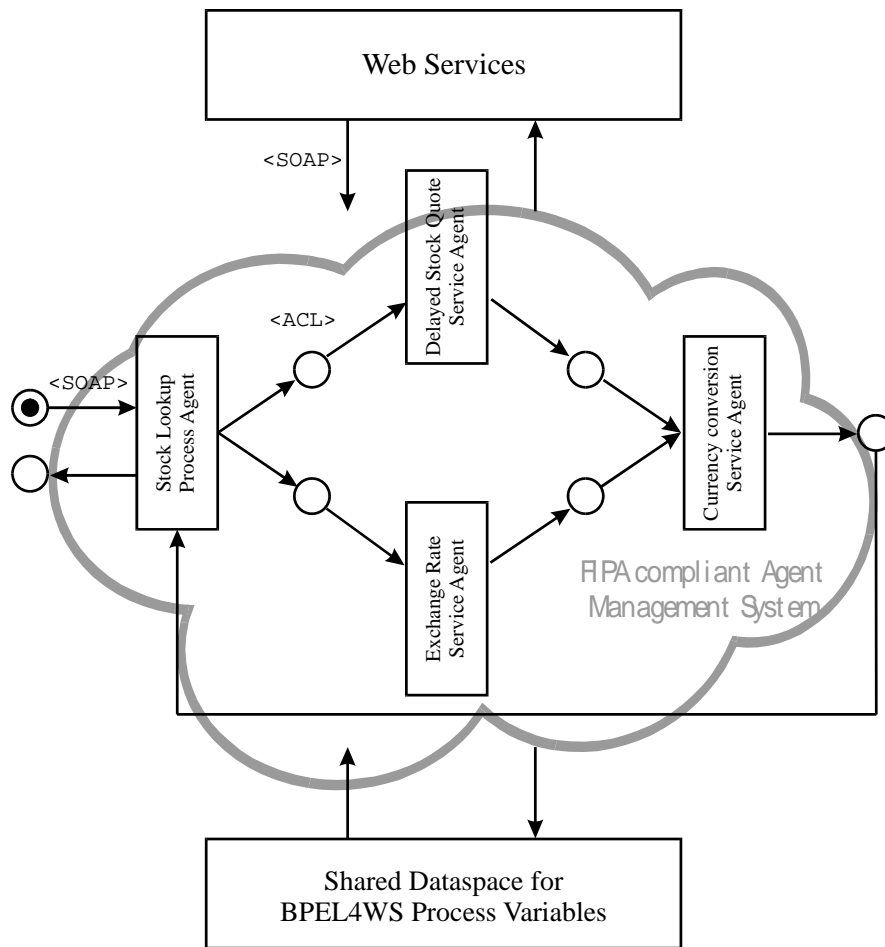


Figure 2.10 Architectural Components of the Multiagent Enactment Mechanism

A few example scenarios will demonstrate how the agents can adapt the workflow based upon knowledge acquired from: cached interaction histories with their respective Web services, semantic markup, or coded by the application developer.

1. at time zero, a request for a quote of IBM stock in Euro currency is received. No interaction history is present and the entire workflow executes.
2. a request for IBM in US dollars occurs two minutes later. Since the conversion is between like currencies, the Exchange Rate Service Agent doesn't invoke the exchange rate Web service. Likewise, since the delayed stock quote Web service has a QOS guarantee of 15 minutes, the Delayed Stock Quote Service Agent

determines that the cached quote for IBM is valid; therefore the underlying Web service is not invoked.

3. five minutes later a request containing IBM and Swiss Francs arrives. This invocation of the workflow utilizes the cached quote for IBM, but requires that the currency exchange rate be computed.

2.6.2 Multiagent workflow enactment as an autonomic system

I believe that IBM's Autonomic Computing initiative provides an interesting vantage point from which to consider adaptive workflow. As noted in IBM's Autonomic Computing Manifesto [51], complexity itself is a byproduct of automation; workflow management systems by their very definition are the automation of a business process. One of the tenants of the autonomic computing initiative is to remove the complexity from the end-user and embed it in the infrastructure of the system. Sophisticated self-governing processes then manage the infrastructure. These processes possess several key characteristics; among them are: self-configuration, self-optimization, self-healing, and self-preservation. Each of these characteristics speaks to the need for adaptation that is designed to achieve specific goals.

Using multiagent systems for workflow enactment is only the first step that enables the exploration of many other fundamental questions. As noted in [52] autonomic systems will consist of autonomic elements that will have policy driven relationships with one another. If the BPEL4WS workflow description were interpreted as a strict policy statement, then a static enactment mechanism is appropriate; however, if interpreted as a policy guideline, multiagent enactment mechanisms provide a degree of process agility.

A sampling of some of the questions to be explored:

- How might the concept of adjustable autonomy be used to enable multiagent enactment across the spectrum of workflow types, from collaboration to production? In production workflows, multiagent implementation may provide execution-monitoring advantages; even without the agents possessing a high-degree's of autonomy. On the other end of the spectrum, agents that monitor the interaction of the participants in a CSCW scenario could potentially discover interaction patterns, formalize process rules and utilize their autonomy to enact elements of the ad-hoc workflow without manual intervention (self-configuring).
- How might agents leverage workflow design tools that can capture the business logic and rationale for service selection and flow? This meta-process information could latter be utilized by the autonomous agents for process redesign (self-optimization). Having a design specification for the MAS provides self-knowledge, which could be leveraged for self-optimization. For example, agents can use the workflow description to determine the impact of hypothetical changes, or use it, along with knowledge of available resources, to find other resources that can be exploited.
- How might BPEL4WS be extended to allow the specification of multiple, functionally equivalent partners at each end of the service link? In a supply chain management scenario, the agents could use this information to tailor the workflow to deliver different QOS levels based upon cost, time or quality constraints (self-configuring, self-optimizing). Likewise, the list of partners might represent primary and secondary service providers; in the event of primary partner failure, the workflow could automatically engage the secondary partner (self-healing).

- How might an agent's active monitoring of service invocation patterns be useful for the purposes of detecting/correcting inappropriate service access? (self-protection)
Perhaps, agents could use a BPEL4WS process description to identify normal behavior and signal everything else as abnormal. Abnormal behaviors would have to be further analyzed to determine if they are a real threat or a legitimate deviation enacted by the agents in an effort to optimize the system's behavior.
- How might the abstract process notion be useful as a specification that can be instantiated by agents? (self-configuring) An abstract process definition is non-deterministic and does not specify under what conditions each branch is chosen. As such, it can be used by agents to determine the set of "legal" actions and leaves the choice to the agent's reasoning. One can envision the use of abstract specifications (if made very flexible) as very high-level system behavioral limits. The agents would then be free to implement any specific system behavior that falls within this space.

2.7 Related developments

Adaptive workflow capabilities, achieved through multiagent enactment mechanisms, will be influenced by developments related to: BPM software and PDL developments, Web services, the semantic web, and Agent-Oriented Software Engineering (AOSE). The pace of change in each of these areas is quickening as commercial entities strive to capture early market share and consortia like WfMC, BPMP, and W3C struggle to maintain their relevance. In the BPM solution space, this scramble is being driven by market analysis that predicts the BPM market will be worth \$6.32 billion in 2005, up from \$2.26 billion in 2001 [53]. Interestingly, evidence that establishes the need for self-configuring, self-optimizing BPM systems is found in this same research report, which

shows that for every dollar spent on BPM software in 2001, three dollars were spent on related professional integration services.

In the domain of PDL development, I feel that BPEL4WS will become the de facto standard as soon as Microsoft and IBM retool their product offerings for release in 2003. IBM has released BPWS4J on their Alphaworks site [54]. BPWS4J provides a preview of the capabilities that will be available in future versions of WebSphere Studio Application Developer Integration Edition and the WebSphere Application Server. BPWS4J consists of an Eclipse based graphical editor for designing workflows expressed in BPEL4WS PDL. The BPEL4WS workflow descriptions can then be executed on the BPEL4WS workflow engine. Both the WfMC and BPMI have release statements indicating that their own process description languages, XPDL and BPML respectively, are more capable than the BPEL4WS specification; however, they embrace BPEL4WS as a positive development for the BPM industry [55, 56].

Although not at the same frenetic pace, developments are also occurring in the space of Web services, the semantic web and AOSE. Regarding Web services, the WSDL and SOAP specifications are completing an update cycle. The semantic web is transitioning ontology languages from DAML+OIL to the new Web Ontology Language (OWL). The field of AOSE is beginning to pay close attention to the Web service developments. FIPA has formed a technical committee to propose an integration strategy for FIPA compliant agents to interoperate with Web services.

On the academic front, several researchers are working at the intersection of agents and workflow. Specifically, [32, 41, 43] have written about the potential benefits of introducing agent technology into workflow enactment mechanisms. In [43, pg 575],

Marinescu discusses the use of the Bond agent architecture to enact a workflow description captured in XPDL. Most closely related to my vision of using contemporary BPM tools and Web services for multiagent systems design is the work described in [57]. In this paper, Korhonen, et al. describes the creation of a workflow ontology that is used to describe both agents and Web services. They hope to build a workflow enactment mechanism that can utilize the ontology to bridge the communications gap between agents and Web services.

2.8 Conclusion and future work

In this chapter, my goal has been to contextualize thoughts of multiagent systems as a workflow enactment mechanism. I predict that the landscape of enterprise integration will undergo dramatic changes in the next 3-7 years as Web services usher in a new era and BPM applications replace traditional EAI efforts. In these turbulent times historic perspective is necessary for setting appropriate expectation levels. For the reasons presented in Sections 2.2 and 2.6, I firmly believe that the historic trajectory of software development paradigms and IT advancements will establish multiagent systems as the workflow enactment mechanism of the future. The business community, while being inundated with Web service hype and PDL confusion need to remain vigilant for the emergence of disruptive technologies in the BPM application area. Agent-oriented software is maturing at the same time as the semantic web activity. The combination of these two technologies may truly establish the Internet connected virtual enterprises of the future.

Chapter 3

Integrating Agent Services into BPEL4WS Defined Workflows

3.1 Introduction

Business processes, which are rooted in the concept of workflows, are essentially social networks; as such, they lend themselves to analysis via agent-based simulation. Recently, several large corporations, such as Procter & Gamble, Southwest Airlines, Merck, and Ford Motor Company have touted the benefits of agent-based simulations which have identified ways to optimize their operations. In these simulations, software agents represent the individual components of the system. The agent's behaviors are modeled after their real-world counterparts. After validating the accuracy of the simulation, by comparing its performance to the real-world system, individual agent's behavior rules can be modified to assess the impact of the change on the system. Procter & Gamble claims that agent-based simulations have been used to identify optimizations of its supply chain, which are saving the company US \$300 million annually [58].

Currently, two trends are changing the way businesses interact with their environments. The first of these trends is the incorporation of real-time data into business processes. Corporate leaders believe that having the ability to adapt their processes in near real-time will provide a competitive edge; however, the introduction of environmental dynamics may simply destabilize business processes because the sociality of the business process is not typically recognized. The second trend is the dynamic

realignment of business partners enabled by advances in information technology. The need for adaptive processes is being driven by the demands of e-commerce in both B2B and B2C spaces.

Initial B2B automation activities were centered on Electronic Data Interchange (EDI) initiatives. More recent work in the B2B space has focused on the development and deployment of ebXML (electronic business XML). With both EDI and ebXML the collaborating business partners predefine the terms of their electronic interaction. As discussed by Jenz, these technologies enforce regulated B2B interaction and as such, they create closed communities of business partners. [29]. In comparison, views toward virtual organizations require flexible, on-the-fly alignment of business partners; in other words, adaptive workflow capabilities. These loose collaborations of business partners operate in open, non-regulated B2B/B2C scenarios where pre-negotiated collaboration agreements are a hindrance in these environments [29].

Business process management software is gaining momentum due to the emergence of a de facto standard for describing a business process as compositions of Web services. This standard is named BPEL4WS, which is an acronym for Business Process Execution Language for Web services [33]. In my earlier works [39], [59], [60], [61] I have explored the relationship between Web services, Multiagent Systems (MAS), and workflows. My vision is to create adaptive workflow capability through decentralized workflow enactment mechanisms that combine Web service and agent technologies.

The applicability of MAS to workflow enactment has previously been noted [32]; however, it is only recently that the notion of using passive Web services as externally defined behaviors of proactive agents has become palatable. Besides differentiating Web

services and agents based upon a measure of proactivity, there are several other important distinctions worth noting. Some of the distinguishing characteristics provided by Huhns are: Web services know only about themselves, they do not possess any meta-level awareness; Web services are not designed to utilize or understand ontologies; and Web services are not capable of autonomous action, intentional communication, or deliberately cooperative behavior [40]. In contrast, agents possess all of these capabilities.

This chapter discusses the first step toward moving agents out of the simulation environment and injecting them into the workflow itself. To facilitate this discussion, an example BPEL4WS process is introduced. This sample process serves as a running example throughout the rest of the chapter. Next, the major components of the software infrastructure, which allows the integration of the agents into the workflow, are described. The chapter proceeds to discuss the end-to-end demonstration of the workflow engine calling an agent service. The chapter concludes with a summary of its contribution and a discussion of future directions for this work.

3.2 An Example BPEL4WS Workflow

BPEL4WS is a Web service composition language; as such it allows the specification of a collection of Web services and the coordination of their interaction. When thought of from a business process perspective, BPEL4WS can be said to be a process description language suitable for defining workflows where the activities to be performed consist of Web service invocations. A thorough explanation of BPEL4WS is beyond the scope of this chapter; however, the following sample is provided to help the uninitiated develop an

intuition about BPEL4WS. Please note that namespace and variable information has been removed from the example in an effort to simplify its presentation.

```
<process
  name="stockLookupProcess">
  <partners>
    <partner name="requestor"/>
    <partner name="stockQuoteProvider"/>
    <partner name="currencyExchangeProvider"/>
    <partner name="simpleFloatMultProvider"/>
  </partners>
  <variables>
    ...
  </variables>
  <sequence name="main">
    <receive name="request"></receive>
    <flow name="getQuoteandExchangeRate">
      <invoke name="getStockQuote"
        operation="getQuote"
        inputVariable="stockQuoteRequest"
        outputVariable="stockQuoteResponse">
      </invoke>
      <invoke name="getExchangeRate"
        operation="getRate"
        inputVariable="currencyExchangeRateRequest"
        outputVariable="currencyExchangeRateResponse">
      </invoke>
    </flow>
    <invoke name="multiplyFloat"
      operation="multiply"
      inputVariable="simpleFloatMultRequest"
      outputVariable="simpleFloatMultResponse">
    </invoke>
    <reply name="response"
      operation="requestLookup"
      variable="response">
    </reply>
  </sequence>
</process>
```

The basic functionality of this workflow is to provide a stock quote that is converted into the requestor's local currency. If invoked with the arguments "IBM" and "Switzerland" the stock quote for IBM would be converted into Swiss Francs before being returned. This workflow has four participating partners: the service requestor, who requests the execution of the workflow; the stock quote provider, a Web service that provides delayed stock quotes; the currency exchange rate provider, a Web service that

provides exchange rates between countries; and the simple floating point multiplication provider, a Web service that multiplies two numbers and returns their product. Figure 3.1, provides a graphical view of the structure of the workflow in Use Case Maps (UCM) notation [62]. UCM is intuitive; the line represents the thread of control, which passes through the partners of the workflow. The workflow process starts at the end of the line designated with a ball, which corresponds with the `<sequence>` tag in the process definition. The process ends at the end capped by a line, this aligns with the `</sequence>` tag.. Tracing this line from start to finish provides an accurate account of the temporal ordering of the workflow's activities. Notably, the line splits and joins in the middle of the process, this corresponds to the `<flow>`, `</flow>` tags respectively. In BPEL4WS activities found inside a flow block are executed concurrently.

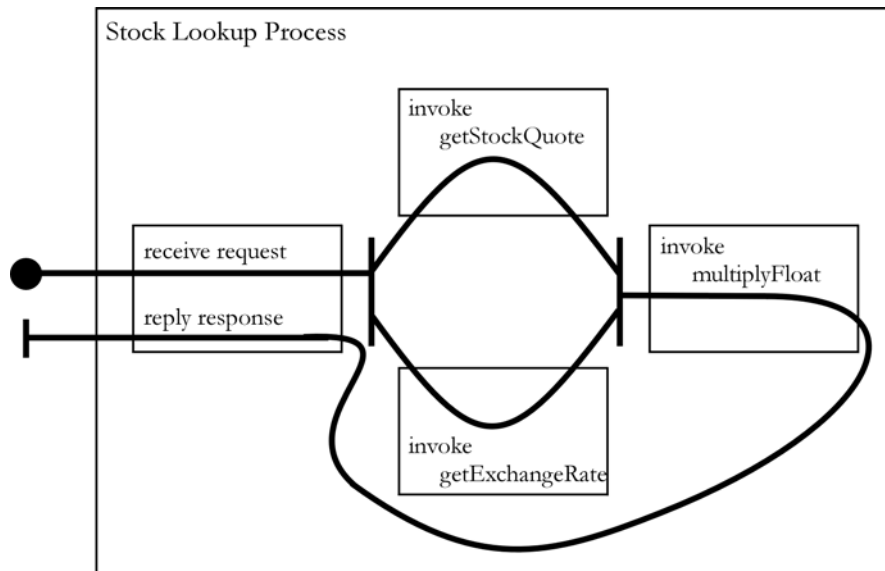


Figure 3.1 A Use Case Maps model of the BPEL4WS example

3.3 Infrastructure

In order to demonstrate the integration of an agent into a workflow, a platform to carry out this research was needed. The major components of the platform are the BPWS4J Editor and Engine 2.0 [54], the Web service Agent Gateway (WSAG) 1.0 [63], and

JADE (Java Agent DEvelopment framework) 3.1 [64]. In addition to these primary components, the following are also used: J2SE 1.4.2, Eclipse 2.1, Tomcat 4.1.29, Axis 1.1, and webMethods Glue Standard Edition 4.1.2. Each of the major components will be briefly discussed in subsequent sections.

3.3.1 BPWS4J

BPWS4J is the common name for the IBM Business Process Execution Language for Web services Java Run Time. BPWS4J provides two essential functions to the infrastructure. The first contribution is the BPWS4J Editor, which is a plug-in for the Eclipse environment. The Editor allows for the graphical creation of a BPEL4WS defined workflow. Having the capability to diagrammatically define the workflow is helpful because it is unwieldy to write programs in XML. The second essential function is the BPWS4J Engine, which provides an enactment service for BPEL4WS workflows. The engine consumes a workflow specification and deploys it as a Web service. When the Web service is invoked, the underlying workflow executes.

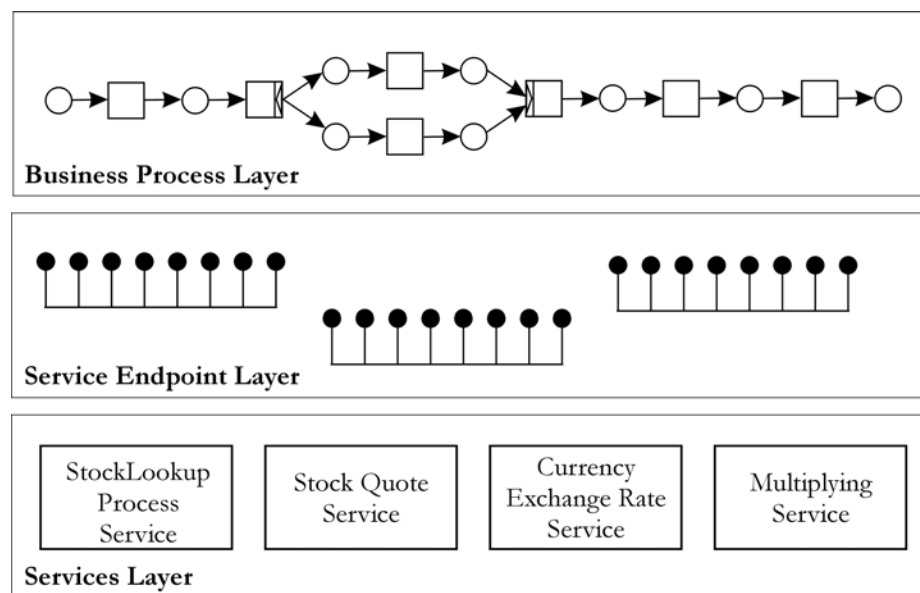


Figure 3.2 BPWS4J viewed as a layered model.

The BPWS4J's Engine coordinates the activities that occur in the Business Process Layer (see Figure 3.2). BPWS4J enacts the business process as encoded in the BPEL4WS file. BPWS4J locates and binds to the service endpoints based upon information found in the user supplied WSDL files. In Figure 3.2, the middle layer contains the service endpoints and the services are found in the lowest layer. The Web services are insulated from their respective endpoints by Web service deployment middleware. Although the business process is separated from the Web services, they are relatively tightly coupled due to the dependence on explicit operation and message syntax embedded in the BPEL4WS workflow description.

The communication between the layers identified in Figure 3.2 occurs in a bi-directional and synchronous manner. Layered models with synchronous communication channels are characteristic of distributed applications [65]. BPWS4J coordinates the execution of workflow as a distributed application; thus it can be concluded that BPWS4J is not an application integration platform. Although the difference between distributed applications and application integration is subtle, it is fundamental to my desire to develop decentralized agent-based workflow enactment mechanisms.

Agents can be viewed as independent applications that provide services to one another through loosely coupled, asynchronous message exchange. Agents are able to take advantage of the non-blocking nature of their messaging by overlapping other processing with their communicative acts. The agent uses its autonomy to determine what work to perform; however, I can envision an agent searching for ways to optimize the workflow in which it is engaged. This might occur through finding other service partners

that provide better quality of service, or learning from its interaction histories with existing partners so as to maximize the utility of their future interactions.

3.3.2 WSAG

The WSAG is a partial implementation of the Agentcities Web services Working Group's Technical Recommendation on the integration of Web services into the Agentcities/openNet platform [66]. Gateway agents reside in the WSAG and are responsible for managing conversations with target agents. The WSAG provides the capability to pass a Web service invocation request through a gateway agent to a target agent. The target agent services the request and responds back through the gateway to the Web service client. Thus the WSAG functions as a translator between SOAP message traffic and ACL (Agent Communication Language) based communicative acts (see Figure 3.3). ACL messages are exchanged asynchronously, whereas SOAP message exchange occurs synchronously.



Figure 3.3 The WSAG enables messaging between Web services and agents.

Copyright © 2002-2003 Agentcities Task Force (ACTF).

The WSAG software consists of two related components: one, a set of tools which greatly assist in the generation of gateway agents; and two, a gateway configuration and deployment application that runs within a Tomcat servlet container. To develop a gateway agent the developer must first write a Java interface, which defines the parameters required by the target agent service. The code generation tools use the Java interface definition to generate the gateway agent. The developer is only responsible for

writing the Java code that moves data to/from the content area of the ACL messages used to communicate with the target agent. Once the gateway agent is created, the management functions of the WSAG software are used to deploy the gateway agent and associate it with the target agent.

3.3.3 JADE

JADE is a popular FIPA compliant, Java-based agent development platform. FIPA (Foundation for Intelligent Physical Agents) is a consortium that produces standards to enhance the interoperability of heterogeneous agents [50]. The WSAG uses JADE for the implementation of the gateway agents. The target agents in the demonstration system were also constructed with JADE; however, any FIPA compliant agent toolkit would work.

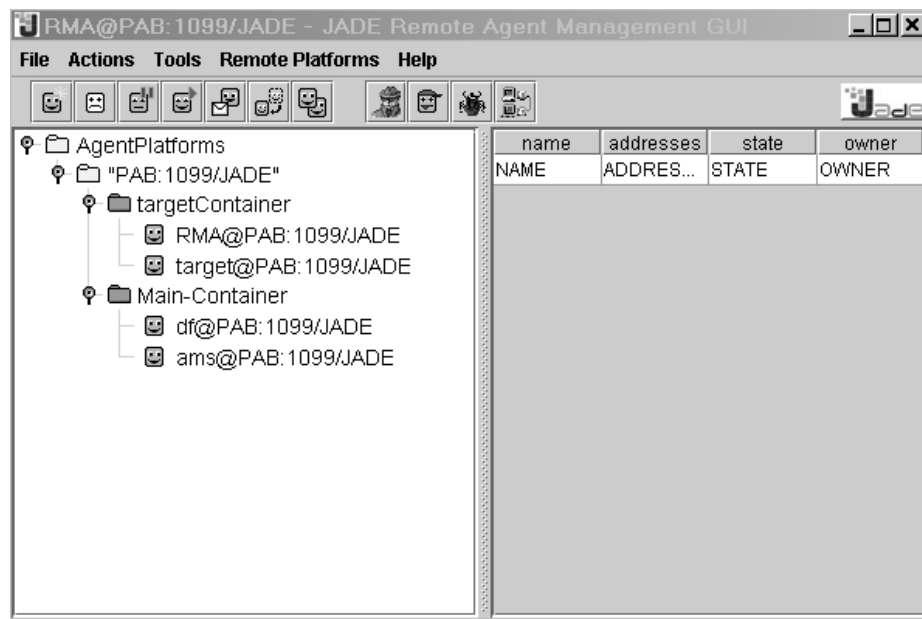


Figure 3.4 Jade's Remote Agent Management utility

JADE implements the FIPA reference model for agent platforms. Figure 3.4 depicts JADE's Remote Agent Management utility which provides facilities for interacting with agents and managing the agent platform. Figure 3.4 shows that one agent platform with

two containers is executing. The main container supplies basic services to the agent platform. The Directory Facilitator (DF) provides yellow page services to the agents running on the platform. The DF also provides the mechanism for agents to advertise their services in the agent directory. The Agent Management System (AMS) provides services to the agent platform that allow the creation, deletion and migration of agents.

3.4 End-To-End Demonstration

An end-to-end demonstration was performed utilizing the software detailed in the previous section. This demonstration shows the feasibility of injecting an agent into a workflow executed by the BPWS4J Engine. The concept is to use the WSAG to slide an agent between the enactment mechanism and a destination Web service. The example workflow introduced in Section 2, was used for the demonstration. In the demonstration, the BPWS4J engine passes control to the target agent, which in turn calls the Stock Quote Web service. The target agent can be seen running in the target container in Figure 3.4.

Since the workflow calls the target agent instead of the Web service directly, an opportunity is created to do something intelligent. One can imagine many ways in which this could be useful. Perhaps the Stock Quote Web service guarantees the quote it provides is current within the past 15 minutes and it charges a micropayment for service access. The agent might check a local cache of interaction histories with the Web service to see if the requested symbol has been quoted within the past few minutes. If so, the agent may choose to return the cached quote value instead of calling the service, thus saving the cost associated with the invocation. If the agent were made aware of the stock exchange schedule, it could use its cache for quote requests that occur after the closing

bell, and on weekends and holidays. Of course many other possibilities exist when semantic service matching is considered.

3.4.1 The Software Development Process

It should be clear the demonstration system was assembled with a development process that was primarily compositional, as opposed to creational. Compositional software development methodologies are by necessity different than those used for bespoke software development. The act of composition requires an iterative process that contains the following ordered steps: identification, selection, installation, integration and evaluation.

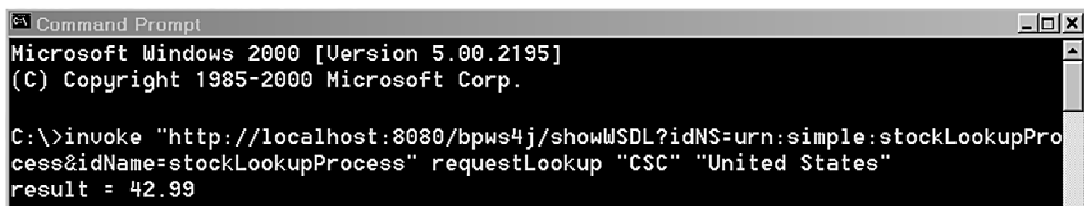
In the identification stage, the marketplace is scanned for relevant components. The most promising candidates are selected based upon criteria that not only include a measure of requirements fit, but also cost, support, and supplier stability. The selected components are then installed and reconciled with the system dependencies of the other components within the solution space. After the components are installed, they must be integrated via configuration and customization. Finally, once the system is assembled, it can be evaluated.

It is important to realize that the only part of the system owned by the project is its architecture [67]. The architecture needs to account for the future evolution of the system. This evolutionary dynamic cannot be ignored as new products continuously enter and leave the solution space. Functionally, the architecture of the demonstration systems relies upon workflow design and enactment tools, Web service/agent integration technology, agent construction tools, and miscellaneous supporting infrastructure. Although the overall system is integrated, each of these architectural components can be evolved separately given that their integration points remain consistent.

Open standards play an important role in the stability of a composed software system. The demonstration system depends upon components that adhere to workflow specification standards, Web service standards, and agent standards. BPEL4WS provides the de facto standard for workflow specification, when each of the activities in the workflow can be accessed as a Web service. The core Web service standards of WSDL, SOAP, and HTTP ensure Web service interoperability. In the agent space, the FIPA standards define the basic services that need to be supplied by compliant agent platforms. Adherence to the FIPA standards enables agents from heterogeneous sources to assemble in open systems.

3.4.2 *Putting the Pieces Together*

For demonstration purposes, it is important to establish that the existing workflow is executing successfully, before altering the system to incorporate an agent service. When the BPWS4J engine has deployed the workflow, it can be tested by invoking the workflow's Web service and verifying the response. The webMethods Glue toolkit provides a convenient command-line Web service invocation utility that eliminates the need to code a Web service client. Figure 3.5 shows how this utility can be used to invoke the workflow executing in the BPWS4J engine.



```
Command Prompt
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>invoke "http://localhost:8080/bpws4j/showWSDL?idNS=urn:simple:stockLookupProcess&idName=stockLookupProcess" requestLookup "CSC" "United States"
result = 42.99
```

Figure 3.5 Invoking the workflow from the command-line

Once the workflow has been verified, select a Web service from the workflow to replace with an agent service. In the demonstration system, the delayed stock quote Web

service was selected for this purpose. At this point, a gateway agent for the WSAG needs to be constructed with the same interface as the replaced Web service. The WSAG automates much of this task after a Java interface has been written by the developer. The following code sample defines a Java interface for a target agent service that mimics the stock quote service that is being replaced.

```
package stockQuoteAgent;  
  
public interface StockQuote {  
    Float getQuote( String symbol );  
}
```

As indicated in Section 3.2, the WSAG gateway agent development tools use this interface definition to construct both the gateway agent and its Web service interface. Although the defined StockQuote interface is consistent with the service being replaced, the WSAG tools are unable to generate a WSDL file that is compatible with the BPEL4WS workflow definition. When incompatible WSDL definitions are found by the BPWS4J engine, the deployment of the workflow fails. To illustrate this point, a snippet of the tool generated WSDL file follows:

```
<wsdl:message name="getQuoteResponse">  
    <wsdl:part name="getQuoteReturn" type="xsd:float"/>  
</wsdl:message>  
<wsdl:message name="getQuoteRequest">  
    <wsdl:part name="in0" type="xsd:string"/>  
</wsdl:message>  
<wsdl:portType name="StockQuote">  
    <wsdl:operation name="getQuote"  
        parameterOrder="in0">  
        <wsdl:input message="impl:getQuoteRequest"  
            name="getQuoteRequest"/>  
        <wsdl:output message="impl:getQuoteResponse"  
            name="getQuoteResponse"/>  
        </wsdl:operation>  
    </wsdl:portType>
```

This portion of the WSDL file defines the abstract interface to the Web service. It states that the Web service provides an operation named getQuote that receives a

getQuoteRequest message that contains one variable named in0 whose type is xsd:string. The output of the getQuote operation is a getQuoteResponse message that contains one variable named getQuoteReturn whose type is xsd:float. Although semantically the same as the getQuote service being replaced, the names of the messages and their parameters do not align with the syntax of the replaced service.

Although the BPEL4WS description is decoupled from the services, the underlying services associated with the workflow must have the exact interface definition as the abstract Web services the BPEL4WS was written against. To resolve the deployment issue the incompatibility causes, the WSDL file generated by the WSAG had to be manually edited to reconcile the differences. For comparative purposes, the pertinent section of the new WSDL description follows:

```
<wsdl:message name="getQuoteResponse1">
  <wsdl:part name="Result" type="xsd:float"/>
</wsdl:message>
<wsdl:message name="getQuoteRequest1">
  <wsdl:part name="symbol" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="StockQuote">
  <wsdl:operation name="getQuote"
    parameterOrder="symbol">
    <wsdl:input message="impl:getQuoteRequest1"
      name="getQuoteRequest1"/>
    <wsdl:output message="impl:getQuoteResponse1"
      name="getQuoteResponse1"/>
  </wsdl:operation>
</wsdl:portType>
```

Once the WSDL for the gateway agent is conformant with the WSDL of the service being replaced, the gateway agent can be installed in the WSAG. During the installation process, the address for the target agent is supplied. Figure 3.6, shows a partial screen capture of a WSAG user interface. An examination of Figure 3.6 shows how the gateway agent is configured to communicate with the target agent. The Remote Agent Management utility, shown in Figure 3.4, can be used to obtain the address of the target

agent. Whenever the Web service interface to the gateway agent is invoked, the SOAP message is effectively translated to an ACL message and sent to the target agent.

List of Deployed Web Services

◆ **stockQuoteAgent** | [WSDL interface](#) | [UNDEPLOY](#) | [REGISTER](#) into UDDI

Type	stockQuoteAgent.StockQuote
Target Agent	target@PAB:1099/JADE
Target Platform	http://PAB:7778/acc

Figure 3.6 WSAG Deployed Web services screen

After the gateway agent's Web service interface has been deployed, the workflow needs to be updated to invoke it. This is accomplished by redeploying the workflow through an administrative interface of the BPWS4J Engine. During the deployment process, the path to the WSDL files for each service partner is supplied. Figure 3.7, shows a screen capture of the BPWS4J user interface that is used in this step. Instead of supplying the WSDL to the original stock quote service, the WSDL file for the gateway agent's Web service interface is entered. After this change is made, whenever the workflow needs to call the stock quote service, control will be passed to the gateway agent which in turn communicates with the target agent.

Deploy a Process: Partner Identification

Please enter the name of the WSDL file which corresponds to each of the following partners in the business process.

stockQuoteProvider

simpleFloatMultProvider

currencyExchangeProvider

Figure 3.7 BPWS4J Partner Identification Screen

Currently the demonstration system is only designed to illustrate the incorporation of an agent service into a BPEL4WS workflow. The target agent possesses no intelligence and simply calls the underlying delayed stock quote service that was previously invoked

by BPWS4J. The target agent is a JADE agent and is therefore written in Java. The following Java snippet is self documenting and shows the relevant processing:

```
private void stockQuoteRequestBehavior( ACLMessage requestMsg )
{
    final String wsdlName = "urn:xmethods-delayed-quotes.wsdl";
    final String operation = "getQuote";
    String args[] = new String[1];

    // copy the stock symbol from the ACL msg to args[0]
    args[0] = requestMsg.getContent();

    // invoke the stock quote Web service
    String wsResponse = invokeWebService( wsdlName,
        operation, args );

    // create an ACL message to return the response
    ACLMessage responseMsg = new ACLMessage( ACLMessage.INFORM );

    // set this agent's ID in the sender portion of the msg
    responseMsg.setSender( getAID() );

    // set the receiver to be the address of the sender
    responseMsg.addReceiver( requestMsg.getSender() );

    // place the returned stock quote into the content area
    responseMsg.setContent( wsResponse );

    // send the message back to the requestor
    send( responseMsg );
}
```

The message sent from the target agent, returns to the gateway agent. The gateway agent extracts the stock quote from the ACL content area, and returns it back to the BPWS4J Engine. From the Engine's perspective, the changeover from a Web service to an agent service is transparent.

3.5 Conclusion

Many lessons were learned during the construction of the demonstration system and much work lay ahead. Specifically, a next generation WSAG is being planned. Greater adaptability can be achieved in the next generation WSAG if the coupling between the gateway and target agents were loosened. One approach to looser coupling would require

target agents to register with the agent platform's directory facilitator their ability to handle certain Web service requests. When the gateway agent receives a request it could use the directory facilitator to locate target agents capable of providing the desired service. Additionally, it seems logical to design a content language for Web service/agent interaction. Simply placing the invocation parameters into the ACL content area is insufficient for operation in open agent environments. Perhaps something as straightforward as using SOAP as a content language would fulfill this requirement. An added benefit to using SOAP directly is that namespace information would be preserved. As Web services transition from rpc/literal to doc/literal invocation styles, the namespaces will likely be useful in associating semantic meaning to the message content.

Recently, one other application of the developed platform has been recognized. The Semantic Discovery Service (SDS) described in [68] would benefit from deployment as an agent service rather than as a Web service. As an agent service, the SDS could register itself with the agent platform's directory facilitator enabling its use by other agents. Use of the WSAG also allows the SDS to be the target agent for many gateway agents, each with its own WSDL definition. The gateway agents would be responsible for providing an OWL-S description of the desired service. This semantic description could be passed in the content area of the ACL message along with the invocation parameters. This arrangement allows the use of the SDS to be decoupled from the BPEL4WS process definition, which would no longer have to be modified to support the SDS. Integrating the SDS in this way allows it to be truly agnostic for it would no longer require that it be packaged in a Web service wrapper for each use.

The work presented in this chapter is but a first step toward fully integrated agent-based workflow management systems. Although this step may appear small, it represents a great stride forward since it establishes a research platform upon which further Web service/agent integration activities can be performed. The development of the demonstration system illustrates the power of compositional approaches to system creation. It also serves to reinforce the importance of open standards, since the integration of the separate pieces is dependent upon the interoperability that standards provide.

Chapter 4

Enacting BPEL4WS Specified Workflows with Multiagent Systems

4.1 Introduction

Distributed computing is undergoing revolutionary change as the worlds of Service-Oriented Computing (SOC), Multiagent Systems (MAS), and Business Process Management (BPM) converge. Web services will be the foundational technology that will underpin future distributed, internet-based computing systems. As the Semantic Web matures, Web services will routinely advertise a semantically rich description of their capabilities. These descriptions will likely be encoded in OWL-S, a semantic markup language designed for Web services [69]. Exploitation of these trends will require agile software structures that support the loosely coupled interaction of services that are found and bound at run-time. Much theoretical and practical work remains to transform this vision into reality, as change needs to occur at both the infrastructure and application levels.

This chapter details the design and development of an open, distributed, agent-based workflow enactment mechanism utilizing BPEL4WS [33] as the specification of the Multiagent System (MAS). The impact of this work is broad, as it cuts a swath across many existing and emerging technologies; for example, Business Process Management Systems, Web services, Internet Agents, application integration, and XML-based coordination mediums.

This chapter will first detail a sample BPEL4WS workflow that will serve as a running example throughout the remainder of the chapter. Next, a discussion of the architecture and design of the distributed enactment mechanism is presented. This is followed by an examination of the hybrid coordination model used. The discussion proceeds with detail about the implementation of the workflow agents. The chapter provides information on how the enactment mechanism is configured, including an examination of the configuration data that is consumed by the workflow agents.

4.2 A Sample BPEL4WS Workflow

BPEL4WS is an XML-based defacto standard that allows the specification of a workflow where the activities are defined by Web service invocations. BPEL4WS has been submitted to OASIS for standardization and in the future will be known as WS-BPEL. A complete description of BPEL4WS is beyond the scope of this chapter; however, the following discussion should provide enough background to enable understanding of the sample workflow.

BPEL4WS files specify the coordination of control and data between service partners that represent underlying Web services. Control constructs such as sequence and split-join are represented by XML tags that delineate control blocks. For example, the actions found between a <flow>, </flow> tags are to be executed concurrently. BPEL4WS defers to the underlying WSDL for the specification of the data that is exchanged by the service partners. The messages exchanged with a Web service are designated by variables within the BPEL4WS file. Assignment and copy operations between variables allows data to be manipulated and passed between Web services.

Often initial research efforts are directed toward solving “toy” problems. The example workflow described below serves this purpose. Abstractly, the workflow consumes two parameters, a stock symbol and a country name. The result of the workflow is a quote for the stock localized into the currency of the given country. For example, providing ‘CSC’ and ‘Switzerland’ will return the price for a single share of Computer Sciences Corporation stock in Swiss Francs.

The example workflow encoded in BPEL4WS follows. A few items to note, bold-face text is used to designate the control constructs and workflow activities, the remaining text describes the data-centric coordination of messages exchanged between the partners and their Web services. The BPEL4WS has been simplified by removing attributes that do not help clarify the example.

```
<process>
  <partners>
    <partner name="requestor"/>
    <partner name="stockQuoteProvider"/>
    <partner name="currencyExchangeProvider"/>
    <partner name="simpleFloatMultProvider"/>
  </partners>
  <variables>
    <variable name="request"/>
    <variable name="response"/>
    <variable name="stockQuoteProviderRequest"/>
    <variable name="stockQuoteProviderResponse"/>
    <variable name="currencyExchangeProviderRequest"/>
    <variable name="currencyExchangeProvidrResponse"/>
    <variable name="simpleFloatMultProviderRequest"/>
    <variable name="simpleFloatMultProviderResponse"/>
  </variables>
  <sequence>
    <receive name="request"
      partner="requestor"
      operation="requestLookup"
      variable="request"
      createInstance="yes">
    </receive>
    <assign>
      <copy>
        <from variable="request" part="symbol"/>
        <to variable="stockQuoteProviderRequest" part="symbol"/>
      </copy>
```

```

    <copy>
      <from expression="'usa'"/>
      <to variable="currencyExchangeProviderRequest" part="country1"/>
    </copy>
    <copy>
      <from variable="request" part="country"/>
      <to variable="currencyExchangeProviderRequest" part="country2"/>
    </copy>
  </assign>
  <flow>
    <invoke name="getStockQuote"
      partner="stockQuoteProvider"
      operation="getQuote"
      inputVariable="stockQuoteProviderRequest"
      outputVariable="stockQuoteProviderResponse">
    </invoke>
    <invoke name="getExchangeRate"
      partner="currencyExchangeProvider"
      operation="getRate"
      inputVariable="currencyExchangeProviderRequest"
      outputVariable="currencyExchangeProviderResponse">
    </invoke>
  </flow>
  <assign>
    <copy>
      <from variable="stockQuoteProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f1"/>
    </copy>
    <copy>
      <from variable="currencyExchangeProviderResponse" part="Result"/>
      <to variable="simpleFloatMultProviderRequest" part="f2"/>
    </copy>
  </assign>
  <invoke name="multiplyFloat"
    partner="simpleFloatMultProvider"
    operation="multiply"
    inputVariable=
      "simpleFloatMultProviderRequest"
    outputVariable=
      "simpleFloatMultProviderResponse">
  </invoke>
  <assign>
    <copy>
      <from variable="simpleFloatMultProviderResponse" part="multiplyReturn"/>
      <to variable="response" part="Result"/>
    </copy>
  </assign>
  <reply name="response"
    partner="requestor"
    operation="requestLookup"
    variable="response">
  </reply>
</sequence>
</process>

```

Internally, the workflow definition coordinates the interaction of the four workflow partners named: requestor, stockQuoteProvider, currencyExchangeProvider, and simpleFloatMultProvider. Figure 4.1, provides a graphical view of the structure of the

workflow in Use Case Maps (UCM) notation [62]. UCM is intuitive; the line represents the thread of control, which passes through the partners of the workflow. The workflow process starts at the end of the line designated with a ball. Tracing this line from start to finish provides an accurate account of the temporal ordering of the workflow's activities. Notably, the line splits and joins in the middle of the process, this corresponds to the `<flow>`, `</flow>` tags respectively.

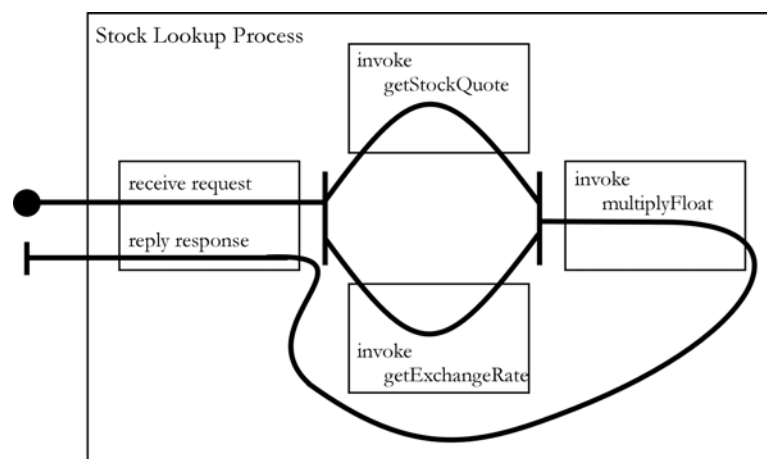


Figure 4.1 A UCM diagram for the example workflow.

4.3 Architecture and Design

Web services and the BPEL4WS have created a resurgence of interest in workflow technologies and process-oriented views of software systems. Traditionally, workflow engines have been based upon the static enactment of workflows under centralized control. This classic approach is at odds with current trends towards real-time enterprises, which closely monitor changing marketplace conditions and events. The ultimate goal is to have this data feedback into the business processes, increasing process responsiveness by allowing adaptive changes to occur. To achieve this type of workflow agility, new enactment mechanisms are required.

Distributed systems possess three dimensions of distribution: computation, control, and data. With BPEL4WS, the Web services are the computational activities, and the control and data dimensions specify the coordination required to manage the process. The BPWS4J Engine is a BPEL4WS enactment engine available from IBM's AlphaWorks site [54]. BPWS4J provides central coordination of the workflow, while the computation is potentially distributed across the Internet. In BPWS4J, each workflow instance has its own thread of control with simulated parallelism, thus the engine enacts the workflow as a distributed application [70]. Distributed applications typically possess a single thread of control and use synchronous communications to transfer control from one component to the next.

My perspective is that the application integration paradigm provides a more appropriate model of Internet based workflow enactment, particularly when inter-organizational workflows are considered. Application integration considers the components to be independently executing applications that are integrated via the asynchronous exchange of data and control. Since Web services are passive entities that don't execute until called, I wrap them in proactive agents that possess their own thread of control. The agents are then integrated to enact the workflow. The agents are coordinated with a shared data space and the asynchronous exchange of messages. This architecture is flexible and loosely coupled.

My goal is to create an open architecture, built atop open standards, for increased interoperability. Just as the primary Web service standards of SOAP, WSDL and UDDI allow for language and platform neutral invocation, I chose to use agent technology based upon the FIPA standards [50], which provide for the interoperability of agents and agent

platforms. Additionally, I chose to use open source or freely available software whenever possible.

Another design goal worth mentioning was the desire to preserve the compositional completeness property inherent to BPEL4WS. In this context, compositional completeness means that the composition of Web services is itself published and accessed as a Web service that can participate in other compositions [11]. Since complex workflows are often viewed as a hierarchy of workflows, the compositional completeness property allows agent-based workflows to be incorporated via BPEL4WS into other workflow definitions.

Based upon my architectural desires and design constraints, the following software components were used in the creation of the distributed enactment mechanism: BPWS4J Editor for the graphical creation of BPEL4WS specified workflows, webMethod's Glue [71] as a high level Web service invocation toolkit, JADE [64] as a FIPA compliant agent development environment, the Web Service Agent Gateway (WSAG) [63] as a bridge between synchronous Web service calls and asynchronous agent messaging, and Xindice [72] as an XML-based coordination medium.

4.4 Coordination of the Workflow Agents

As previously discussed, the domain of coordination encompasses issues of both data and control. The distributed workflow enactment mechanism utilizes a hybrid coordination model, which means that it combines data-centered and control-centered coordination mechanisms [73]. The data is managed via a shared, network addressable XML repository, while the control of the workflow activities is driven by asynchronous

message exchange between the agents. The message exchange pattern for the control messages is derived from a Colored Petri Net (CPN) model of the workflow.

4.4.1 Xindice as a Coordination Medium

Xindice facilitates the storage, retrieval, and sharing of XML data. Xindice is a network addressable native XML database that complies with the XML:DB initiative. Xindice stores XML documents in logical groupings called collections. Data is retrieved from a collection via the evaluation of an XPath [74] query that is evaluated against the documents in a collection. Xindice's features make it an ideal choice as a coordination medium.

Tuple spaces are often the coordination medium of choice for agent-based systems. Tuple spaces allow processes to communicate across space and time, e.g. a process running on one machine can write information to a shared tuple space which is to be read by another process, running on a different machine the day after tomorrow. Tuple spaces provide a form of associative memory. Associative memory is accessed by content, not by address. By way of analogy, SQL is used to retrieve records from a RDBMS that match criteria specified in the 'where' clause of the query. In the same way, a query against a tuple space retrieves records that match criteria specified in a template. With Xindice, XPath can be viewed as a template mechanism that can retrieve specific elements, attributes, or even collections of nodes from an XML document.

An example will provide some insight into how Xindice and XPath are used as a coordination medium for the sharing of data across the distributed workflow agents. In the workflow example, the stockQuoteProvider partner interacts with a stock quote Web service. This interaction occurs with XML-based SOAP messages, which are intercepted and stored in Xindice. A sample of a captured SOAP Response message appears below.

```

<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap...
  xmlns:xsi="http://www.w3.org/2001/...
  xmlns:xsd="http://www.w3.org/2001/...
  xmlns:soapenc="http://schemas.xmls...
  soap:encodingStyle="http://schema... >
  <soap:Body>
    <n:getQuoteResponse
      xmlns:n="urn:xmethods-delayed-quotes">
      <Result xsi:type="xsd:float">
        40.35
      </Result>
    </n:getQuoteResponse>
  </soap:Body>
</soap:Envelope>

```

Downstream in the workflow, the returned stock quote needs to be multiplied against the currency exchange rate to localize the price. For this to occur, the quoted price needs to be extracted from the XML document presented above. The XPath query `string(/n:getQuoteResponse/Result)` retrieves the quote as a string, which can then be converted into its numeric equivalent.

Requests for the execution of the workflow generate unique collections within the Xindice repository. This allows for the clean separation of data between individual workflow cases. Additionally, it assures efficient XPath queries since the number of documents in a given collection remains small. Figure 4.2 depicts two XML documents stored in Xindice and viewed through Xindice's browser-based. The collection named 1081186215373 contains the transaction history between the agents of the distributed enactment mechanism and their underlying Web services for one execution instance (case) of the workflow. In Figure 4.2, the top image shows the data written by the Target Agent into Xindice, the bottom image shows the SOAP message exchange between the agent playing the stockQuoteProvider role and the delayed stock quote Web service.

XML document rooted at Agent. Its id attribute contains the AID of the agent that wrote the document. The role attribute indicates what workflow role the agent was playing at the time. The case attribute relates the document to others in the same workflow instance, while the time attribute indicates when the message was written (in milliseconds since Jan 1, 1970).

db - stockLookupProcess - 1081186215373 -	
73eef7671de43502000000fbbb86b0b	Document '73eef7671de43502000000fbbb86b0b'
73eef7671de43502000000fbbb131f	<?xml version="1.0"?> <agent id="requestor@PAB:1099/JADE" role="requestor"
73eef7671de43502000000fbbb3f2b	case="1081186215373"
73eef7671de43502000000fbbb6441	time="1081186215373"><request><symbol>csc</symbol><country>usa</country></request></agent>

retrieved by the stockQuoteProviderRole with the following xpath query:
string(/agent[@role='requestor']/request//symbol)

retrieved by the currencyExchangeProviderRole with the following xpath query:
string(/agent[@role='requestor']/request//country)

db - stockLookupProcess - 1081186215373 -	
73eef7671de43502000000fbbb86b0b	Document '73eef7671de43502000000fbbb131f'
73eef7671de43502000000fbbb131f	<?xml version="1.0"?> <agent id="stockQuoter@PAB:1099/JADE" role="stockQuoteProvider"
73eef7671de43502000000fbbb3f2b	case="1081186215373" time="1081186390775"><request> <soap:Envelope
73eef7671de43502000000fbbb6441	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
	xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
	xmlns:n="urn:xmethods-delayed-quotes"><soap:Body
	soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><n:getQuote><symbol
	xsi:type="xsd:string">csc</symbol></n:getQuote></soap:Body></soap:Envelope></request><response>
	<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
	soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><soap:Body><n:getQuoteResponse
	xmlns:n="urn:xmethods-delayed-quotes"><Result
	xsi:type="xsd:float">42.3</Result></n:getQuoteResponse></soap:Body></soap:Envelope></response></agent>

retrieved by the simpleFloatMultProvider with the following xpath query:
string(/agent[@role='stockQuoteProvider']/response//Result)

Figure 4.2 Example documents stored in Xindice

4.4.2 CPNs as a Flow Control Mechanism

Petri Nets (PNs) have been used for workflow control since the mid 1990's [75]. PNs, also known as place-transition nets, provide a deceptively simple, yet rigorous, way to model finite state machines. PNs are represented as directed graphs with two types of nodes, places and transitions, which are graphically represented as circles and squares respectively. The state of execution is maintained by tokens that reside in the place nodes

of a PN. A transition is enabled if each of its input places is marked by a token. When a transition is enabled it fires, removing a token from each of the input places and depositing a token in each of the output places. From a workflow perspective, the activities of the process occur at the transition nodes in the net. Figure 4.3 presents the example workflow in PN form, where the transitions correspond with the following activities: A – receive request, B – invoke getStockQuote, C – invoke getExchangeRate, D – invoke multiplyFloat and E – reply response.

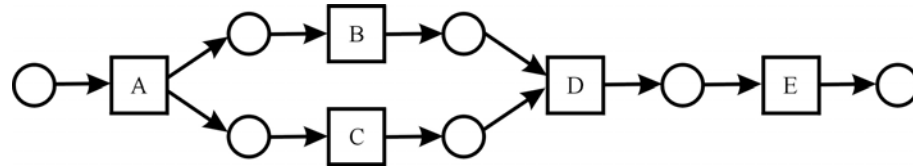


Figure 4.3 A PN Model for the example workflow.

A comparison of the UCM diagram in Figure 4.1 with the PN model in Figure 4.3 reveals that they are equivalent.

CPNs are an extension of basic PNs and include the notion that the tokens carry data. The different colored tokens equate to different data types. The demonstration system utilizes two different colored tokens. The first is used for messaging between the WSAG and the agent-based enactment mechanism. The second is used to communicate control information between the agents as they process a workflow instance. The following is a sample message sent by the WSAG:

```
WSAG:stockLookupProcess:requestor|request:csc:Switzerland
```

The message has a signature indicating that it is being sent by the WSAG. Next, the message identifies the name of the workflow, followed by the partner name the message

is intended for. The vertical bar separates the message header from the payload. The payload of the message indicates that a request is being made for a quote for CSC stock localized into Swiss currency.

An example of a control message exchanged between two agents during workflow enactment follows:

```
DWfA:stockLookupProcess:simpleFloatMultProvider:1080665330511:
...currencyExchangeProvider
```

This message carries the Distributed Workflow Agent (DWfA) signature, identifies the workflow name, and the partner name the message is intended for. The numeric value is a unique ID that is assigned to each workflow instance. This ID is also used to identify the appropriate collection in the Xindice database. The final piece of information is the name of the partner role that sent the message, in this case this message is from the currencyExchangeProvider. Given the PN shown in Figure 4.3, it should be apparent that before the simpleFloatMultProvider can invoke the multiplication Web service, it would need to receive messages for the same workflow instance from both the currencyExchangeProvider and the stockQuoteProvider.

It is easy to imagine using a PN within a centralized workflow enactment mechanism to control the execution order of the workflow activities. However, an interesting question arises regarding the use of a PN for distributed workflow enactment. This question is how is it possible to separate the net into pieces that can be distributed while retaining equivalent behavior. The answer is illustrated in Figure 4.4, which depicts the refinement of a place between two transitions with a simple PN consisting of two places and one transition. Transition T1 sends a token to place P2.1, which serves to enable the

subsequent transition that in turn sends its output token to P2.2 that may reside across a network. P2.2 enables transition T2.

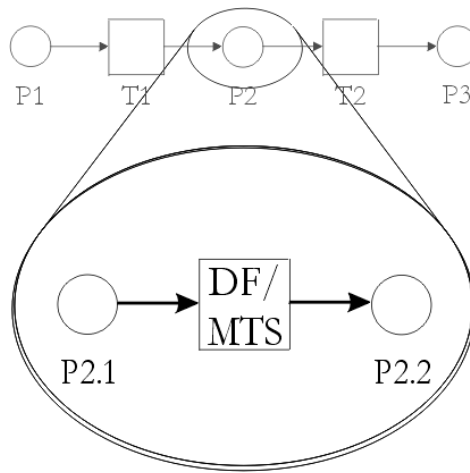


Figure 4.4 Refinement of P2 with a subnet.

More concretely, the transitions in the PN model are agents and the transition containing DF/MTS represents FIPA compliant Directory Facilitator (DF) and Message Transport Service (MTS) components. When an agent in the workflow completes its task, it utilizes the DF to locate the address of the agent that has registered itself as playing the next partner role that needs to receive control. The agent generates an ACL Request message, loads the content area with DWfA signed data, and sends the message to the address returned by the DF. The MTS in turn facilitates the message delivery. Thus the distribution of the CPN is effectively managed by the DF acting as a middle-agent [76].

Figure 4.5 depicts a UML sequence diagram for the message exchange pattern used by the agents during the distributed workflow enactment. Below the sequence diagram is a collection of sample messages. These messages represent actual data collected from the execution of a case through the workflow. The message numbers correspond to the numbers found in the sequence diagram. Note that the gateway agent runs within the

Web Service Agent Gateway. The agent playing the role of the ‘requestor’ is the target agent.

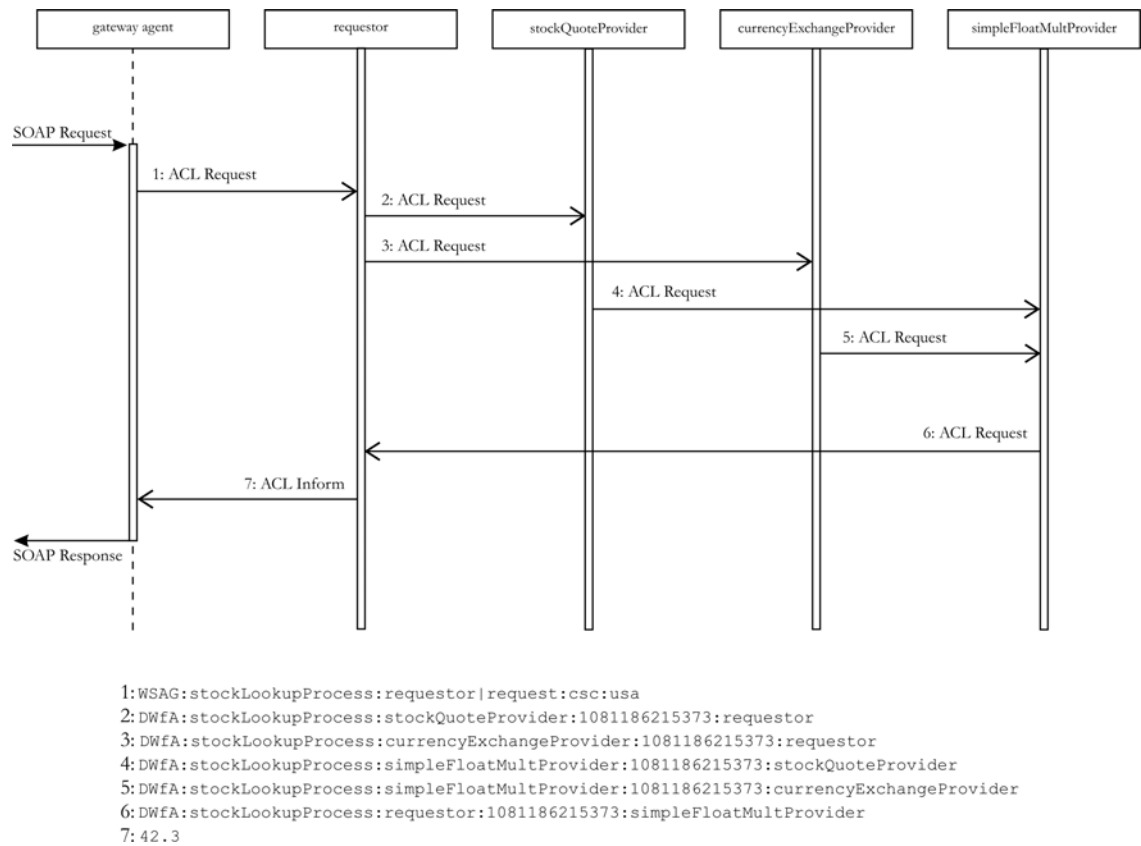


Figure 4.5 UML Sequence diagram with sample messages

4.5 Implementation Details

There are two types of agents that enact the workflow: target agents and distributed workflow agents. A target agent interfaces the distributed workflow agents to the WSAG. The distributed workflow agents are the proactive proxies for the passive Web services they represent. Both types of agents are implemented with JADE and are thus FIPA compliant.

4.5.1 Target Agents

Figure 4.6 illustrates the structure of a target agent in UCM notation. The agent is represented with a parallelogram, which indicates it is an active component in the system.

Target agents receive messages from both the WSAG and other distributed workflow agents; the two distinct execution paths in Figure 4.6 denote this. The boxes found on the execution path simply designate that some processing is occurring, while the two squiggly lines note a “layer fold” in UCM notation. A layer fold is an abstraction that indicates that some complexity is hidden or collapsed along the path. In this case, the layer fold is used to indicate the interaction of the target agent with the middle-agents.

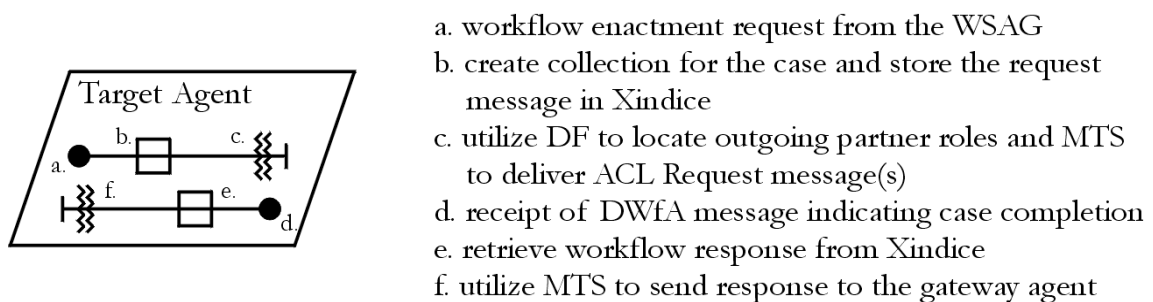


Figure 4.6 UCM diagram of a Target Agent

4.5.2 Distributed Workflow Agents

Figure 4.7 reflects the implementation of the distributed workflow agents. The only new UCM notation is the dashed rounded rectangle, which is a placeholder symbol for a passive component. The distributed workflow agents share the same code base; they are simply instantiated with different workflow partner information. This is consistent with the fact that the primary distinction between these agents is the Web service they represent. In order to achieve the run-time assignment and dynamic invocation of Web services, the capability for robust stubless Web service invocation is required.

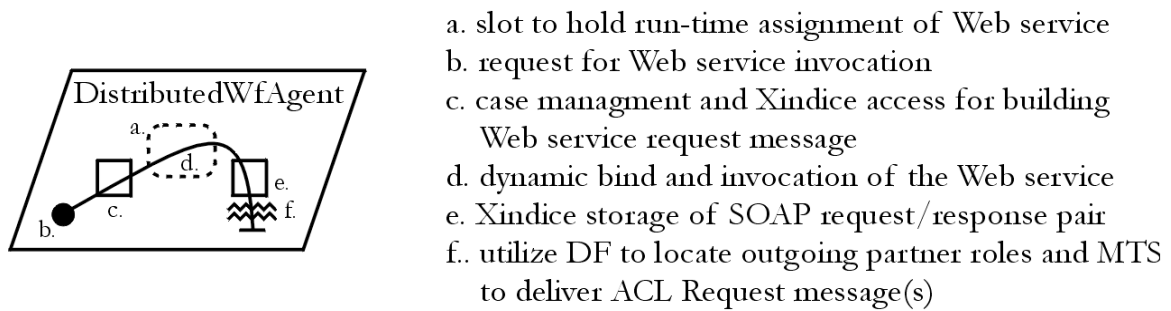


Figure 4.7 UCM diagram of a Distributed Workflow Agent.

4.5.2.1 *Stubless Web service Invocation*

Web service invocation follows the traditional Remote Procedure Invocation (RPI) integration pattern as described in [65]. When viewed generically, RPI is an integration style that achieves Application to Application (A2A) integration by allowing one application to invoke a function published by a second application. The function in the remote application, appears as a local function to other. The underlying mechanism which generates this transparency is based upon providing a function stub to application one, that when called accesses a middleware layer which transports the call and its associated data to application two. The generation of stub functions is typically automated, with tools consuming an interface description of the target function and creating the stub veneer. From a Java Web service perspective, the interface description is the WSDL file and the generation of stubs occurs with a tool such as WSDL2JAVA. The stubs are typically generated during the coding stage of application development. The reason for this is intuitive; the stubs are called directly from the application code and need to be resolved at compile time.

Service-Oriented Architectures provides the mechanisms for Web service partners to be located and invoked at run-time. This obviously requires that more flexible integration styles be developed to support the dynamic publish-find-bind pathways. Functionally,

most Web service toolkits provide some capability for late, run-time binding to Web services. For example, the Glue toolkit from WebMethods [71] provides an IProxy class that can bind to a WSDL description and invoke operations. Similarly, the Web Services Invocation Framework from the Apache project [77] allows for dynamic invocation, as does the JAX-RPC package which is part of the J2EE Web services Developer Package [78].

Unfortunately, seamless dynamic invocation is beyond the capability provided by these toolkits for the simple reason that they are incapable of handling complex types returned from the invoked service. This limitation is due to the fact that the returned data must be unmarshalled from the SOAP message, which in Java is not possible without having a compatible class that implements the serializable interface. In the absence of appropriate classes, the Java run-time environment generates an unmarshall exception. Ironically, the stub generation tools that are not required for dynamic invocation provide these missing classes.

Ideally, there would be a uniform mechanism for handling this problem; however, each toolkit has its own workaround. A singular solution will not be developed until there is broad realization of this problem. Statements such as, “The benefits of using dynamic proxies instead of generated stubs are not clear – it’s probably best to stick with generated stubs” [79, pg 339] only exacerbate the situation. The following Java code snippet illustrates how the Distributed Workflow Agents perform stubless Web service invocation utilizing the webMethods Glue toolkit.

```

public class dynamicInvocationExample {
    public Document dynamicInvocationWithGlue()
        throws Throwable {
        String wsdlName =
            "http://www.ejse.com/WeatherService/Service.asmx?WSDL";
        String operation = "GetWeatherInfo";
        String args[] = { "29424" };

        // create a SOAP interceptor
        SOAPInterceptor responseHandler = new SOAPInterceptor();

        // register the interceptor to catch incoming responses
        ApplicationContext.addInboundSoapResponseInterceptor(
            (ISOAPInterceptor)responseHandler );

        try {
            // obtain a proxy to the Web service via its WSDL
            IProxy proxy = Registry.bind( wsdlName );

            // stubless invoke of the operation
            proxy.invoke( operation, args );
        }
        catch( java.rmi.UnmarshalException e ) {
            // do nothing, the UnmarshalException is expected
        }

        // generate an XML document containing the SOAP body
        return new Document( responseHandler.getResponse() );
    }
}

public class SOAPInterceptor implements ISOAPInterceptor {
    private Element soapBody;

    public void intercept( SOAPMessage message,
        Context messageContext ) {
        try {
            soapBody = message.getBody();
        }
        catch( Exception e ){
            System.err.println( e.toString());
        }
    }

    public Element getResponse() {
        return soapBody;
    }
}

```

In the above code sample, it can be seen that the code negates the effect of the unmarshall exception by catching it and effectively ignoring it. The SOAP interceptor captures the result of the Web service invocation. The dynamicInvocationWithGlue()

method, returns a standalone XML document which contains the body of the SOAP response message. The application can access the returned data via standard XML processing functions by loading the document into a DOM tree, or as in the case of the Distributed Workflow Agents, the XML document is written to the shared XML repository.

4.6 System Configuration

The architecture for the distributed enactment mechanism relies upon many different components that must be properly configured. Figure 4.8 is a high-level diagram that shows the interaction between the major components. Note that the solid lines tipped with arrows indicate synchronous message exchange, while the dashed variation designates asynchronous messaging. The following sections will describe the configuration of the components shown in Figure 4.8.

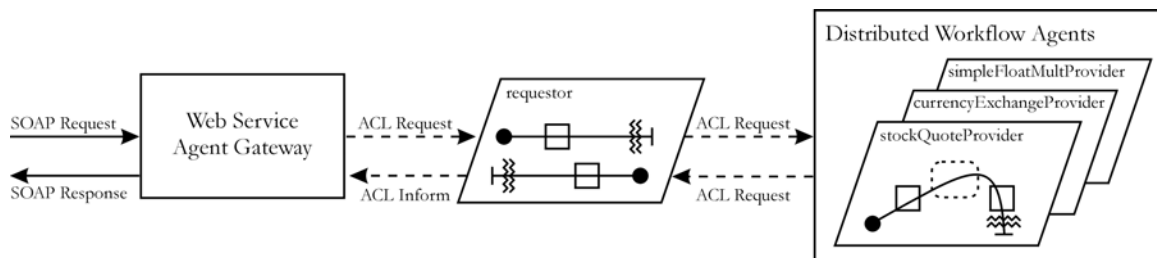


Figure 4.8 The components of the distributed enactment mechanism

4.6.1 Configuring the WSAG

The WSAG provides a Web service interface for services provided by a target agent. In my example, the target agent plays the requestor partner role. As defined in the BPEL4WS, the requestor receives requests from end users and responds with a reply after the workflow runs.

Use of the WSAG requires that a gateway agent is generated and deployed. It is critical that the interface for the gateway agent aligns with the workflow's SOAP request

and response message structure. The gateway agent's interface is specified with a Java interface. The WSAG provides tools that facilitate the generation of gateway agents. These tools consume the Java interface and produce a skeletal gateway agent. The skeletal code is then edited to comply with the messaging interface of the target agent. The gateway agent is then compiled and packaged for deployment. For the example workflow, the following Java interface was used to generate the gateway agent.

```
package stockLookupProcess;

public interface StockLookupProcess
{
    Float request( String symbol,
                  String country );
}
```

Once the gateway agent is built and installed, it needs to be deployed. The deployment step publishes a WSDL interface for the gateway agent, and associates the gateway agent with the target agent. The WSAG management console provides the means to accomplish this task. Figure 4.9 shows the configuration of the stockLookupProcess gateway agent. When the WSAG receives a SOAP request for the stockLookupProcess, the gateway sends an ACL request to the specified target agent. When the workflow is complete the target agent sends an ACL Inform back to the gateway agent, which in turn sends a SOAP response to the workflow consumer.

Type	stockLookupProcess.StockLookupProcess
Target Agent	requestor@PAB: 1099/JADE
Target Platform	http://PAB: 7778/acc

Figure 4.9 Configuration of the Gateway Agent

4.6.2 *Configuring the Workflow Agents*

The workflow agents in the system share a single configuration file, expressed in XML, that is stored in Xindice. The configuration data is derived from the BPEL4WS file and the underlying WSDL files for the individual Web services. Currently, the configuration data is manually generated; however, I believe that much of this process can be automated. A sample of the configuration data is provided and discussed below.

```
<configData workflow="stockLookupProcess">

<messages>
  <message name="request">
    <part name="symbol" type="xsd:string"/>
    <part name="country" type="xsd:string"/>
  </message>
  <message name="response">
    <part name="Result" type="xsd:float">
      q:string(//agent[@role='simpleFloatMultProvider']
        /response//ns1:multiplyReturn)
    </part>
  </message>

  <message name="simpleFloatMultProviderRequest">
    <part name="f1" type="xsd:float">
      q:string(//agent[@role='currencyExchangeProvider']
        /response//Result)
    </part>
    <part name="f2" type="xsd:float">
      q:string(//agent[@role='stockQuoteProvider']
        /response//Result)
    </part>
  </message>
  <message name="simpleFloatMultProviderResponse">
    <part name="multiplyReturn" type="xsd:float"/>
  </message>
</messages>

<partners>
  <partner name="requestor">
    <inputPlaces/>
    <service>
      <wsdl> </wsdl>
      <operation> </operation>
      <messageName>response</messageName>
    </service>
    <outputPlaces>
      <place>stockQuoteProvider</place>
      <place>currencyExchangeProvider</place>
    </outputPlaces>
  </partner>
```

```

<partner name="simpleFloatMultProvider">
  <inputPlaces>
    <place>stockQuoteProvider</place>
    <place>currencyExchangeProvider</place>
  </inputPlaces>
  <service>
    <wsdl>
      http://.../axis/SimpleFloatMult.jws?wsdl
    </wsdl>
    <operation>multiply</operation>
    <messageName>
      simpleFloatMultProviderRequest
    </messageName>
  </service>
  <outputPlaces>
    <place>requestor</place>
  </outputPlaces>
</partner>
</partners>

</configData>

```

The configuration file contains both data-centric and control-centric coordination information relevant to the enactment of the workflow. The data-centric portion is identified with the `<messages>` tag, while the control-centric section is identified with the `<partners>` tag.

The `<messages>` section defines the messages that the individual partners use when interacting with their associated Web service. The message names come directly from the BPEL4WS file, while the message parts are specified in the underlying WSDL files for each Web service. Each message part has an optional value that is either a constant, designated by "c:", or an XPath query designated by a "q:". The associated XPath queries inform the agent how to obtain the data from Xindice. For example, the target agent sends an ACL Inform message to the gateway agent whose contents are the response message defined in the configuration file. The response message contains one part named Result, whose type is `xsd:float`. The XPath query specifies how to obtain the data for the Result part from the repository.

The `<partners>` section contains the control-centric coordination information relevant to each of the partners in the workflow. The partner names are specified in the BPEL4WS file. Each partner is bound to a specific Web service, specified by a wsdl, operation, messageName triplet. The messageName corresponds with a message found in the `<messages>` section of the configuration file.

The agents track each DWfA signed message they receive against the individual workflow cases. When an agent receives a message for a workflow instance from each of the partners specified in the `<inputPlaces>` section, the agent invokes the Web service. Next, the intercepted SOAP request/response pair from the Web service interaction is stored in Xindice. The agent then sends a DWfA message to each of the workflow partners found in the `<outputPlaces>` section. For example, the `simpleFloatMultProvider` will not call the multiplication Web service until it has received messages from both the `stockQuoteProvider` and the `currencyExchangeProvider`. Once these messages are received, the Web service is called, the SOAP interaction stored, and the requestor is sent a DWfA message.

4.6.2.1 Command Line Parameters

The workflow agents are provided the name of the workflow in which they are participating and the name of the partner role they are performing at run time via command line parameters. As previously mentioned, the distributed workflow agents are each instances of the same Java class. It is the command line parameters that distinguish them. The following shows the command line used to establish the `stockQuoteProvider` agent:

```
java jade.Boot -container stockQuoter:DistributedWfAgent  
               (stockLookupProcess stockQuoteProvider)
```

The target agent utilizes a different class file; however, it is established in a similar fashion. The command line to establish the target agent is:

```
java jade.Boot -gui -container-name Target-Container  
requestor:TargetAgent(stockLookupProcess requestor)
```

Figure 4.10 shows a screen shot of the JADE Remote Agent Management console with the entire complement of workflow agents running.

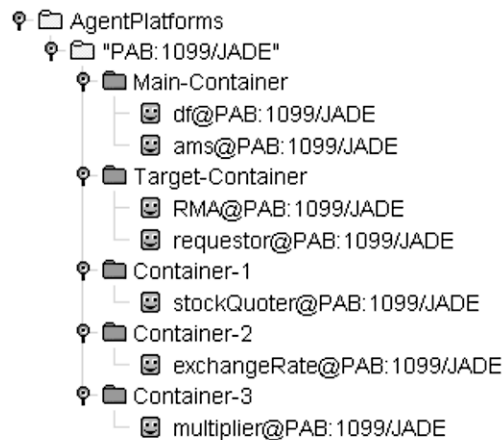


Figure 4.10 The collection of workflow agents in the system

4.7 Conclusion

One of the most important points to make about the distributed workflow enactment mechanism is that it is functional and provides a research platform upon which further refinement and experimentation can be performed. Through its development, many issues have been faced and reasonable and scaleable solutions found. The next chapter provides further context for this work and concludes with a discussion of possible future research directions.

Chapter 5

Conclusion and Future Work

5.1 Major Research Contribution

As described in the research methodology discussion found in Chapter 1, a systems building approach was used to convey and validate the research results. The research described in this dissertation is grounded with two end-to-end demonstration systems. The first of these systems, described in Chapter 3, shows how to integrate agent technologies into contemporary workflow enactment mechanism that exhibit strong, centralized coordination. The second system, described in Chapter 4, is the culmination of this dissertation work and is the implementation of a software architecture for a distributed, functionally equivalent workflow enactment mechanism. The term functionally equivalent means that the workflow uses the same service partners and produces the same result as if it were executed by BPWS4J. This system serves to establish the underlying hypothesis of this work, that software architectures exist that combine agent-based and service-oriented computing concepts for the purpose of workflow enactment. The architecture provides a bridge from current, static workflow enactment technologies to future dynamic workflow engines.

Enabling the transition from static workflow enactment mechanisms toward dynamic ones, requires a software architecture that embodies the properties of weak coordination and loosely coupled interaction. Systems that employ weak coordination exhibit

decentralized control mechanisms, which in turn enable the individual components of the system to exert local control in response to changes sensed in the environment. Loosely coupled interaction, based upon asynchronous messaging, minimizes the integration friction between components allowing possible run-time substitution. These architectural characteristics are necessary to the vision of run-time software adaptation and service-oriented.

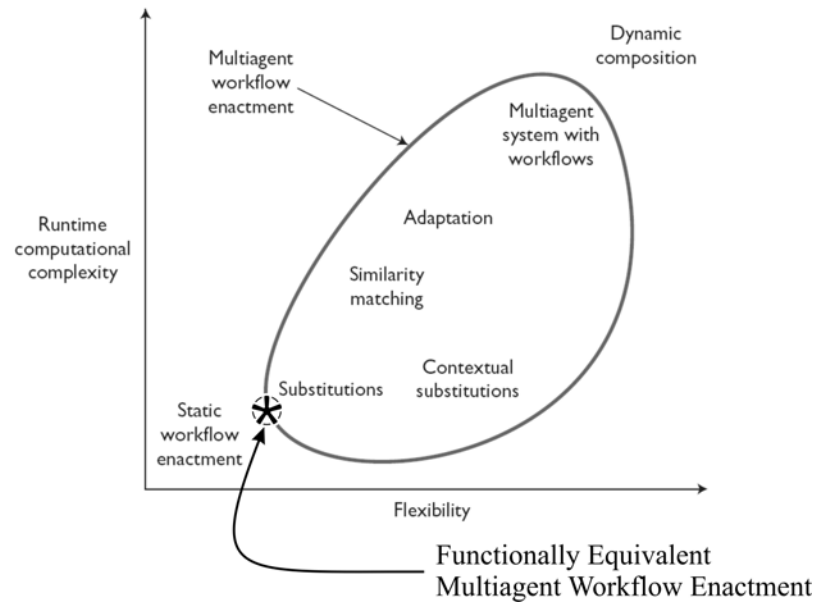


Figure 5.1 Various multiagent workflow adaptation strategies

The article titled *Multiagent Systems with Workflows* by Vidal, Buhler and Stahl [61] provides the context for the current work, as well as a roadmap for future research directions. The software architecture and its instantiation, as described in Chapter 4, provides a research platform upon which further refinement and experimentation can be performed. Figure 5.1 [61] depicts the landscape of strategies through which workflow adaptation may be achieved via multiagent enactment mechanisms. In its current state, the functionally equivalent enactment mechanism, described in this dissertation, is positioned at the intersection of static and multiagent workflow enactment mechanisms.

This location is marked with an asterisk in Figure 5.1. It is a first but important step on the path from static workflow enactment to dynamic on-the-fly composition of workflows.

5.2 Future Research Directions

The work described in this dissertation is but a starting point from which to explore many interesting and challenging problems. This dissertation concludes with a list of potential future research directions.

5.2.1 Externalization of Business Rules

The demonstration system does not support `<switch>` and `<pick>` BPEL4WS constructs. These constructs support selective routing, which can be thought of as the business rules of the workflow process. For example, based upon the response from a Web service invocation, pass control to partner A, otherwise use partner B. It should be possible to preserve the genericity of the distributed workflow agent code-base by augmenting the `<outputPlaces>` section of the configuration file with RuleML. The rules will then be processed as conditional logic scripts in a manner inspired by [80].

5.2.2 Dynamic Business Partner Selection

The hybrid coordination model has proven its relevance with the demonstration system. If a Linda-like tuple space were used to convey control messages, the first agent to consume the message does the work. The use of the DF and asynchronous messaging opens up interesting research opportunities regarding task allocation. For example, consider what might happen when a workflow agent utilizes the DF to locate an agent(s) playing the role identified by an outgoing place and it is discovered that multiple agents are returned. The agent might use a reputation mechanism to select one of the partners, or engage in a bidding scenario managed with a contract net protocol, et al. The point is that

the individual agents maintain the opportunity to do something intelligent and potentially optimize the execution of the workflow at run-time.

5.2.3 Automated Petri Net Creation

The conversion of BPEL4WS into PN form is another area that requires further study. Currently, PNs are generated based upon the replacement property that exists with workflow nets [30]; however, while excellent at modeling positive flow control, it is difficult to capture fault and exception handling. Additionally, the fact that BPEL4WS inherits the calculus-based approach of XLANG presents difficulty when being expressed with PN's graph-based constructs. Ongoing work for developing a PN semantic for BPEL4WS is occurring at Humboldt University. An initial description of this approach can be found in [61].

In the event that the conversion of BPEL4WS into PN form does not mature, it could prove fruitful to switch to YAWL (Yet Another Workflow Language) [81]. YAWL is a developing process description language that supports all the known process patterns [34]. The major benefit of YAWL is that it has been designed with a pure PN semantic.

5.2.4 Semantic Service Replacement

Other opportunities exist to demonstrate the advantages of agent-based workflow enactment. As more semantic Web services become available, I would like to integrate with the Semantic Discovery Service (SDS) [68] as an basic agent service available to the workflow agents. To accomplish this integration, the `<partner>` description in the configuration file would need to be augmented with a semantic description of the Web service the partner represents. At run-time, the workflow agent can use its autonomy to locate other potential Web service partners with the aid of the SDS. This integration would allow the agents to heal the workflow in the event that their primary Web service

becomes unresponsive. Likewise, various Web services would likely provide different QOS levels, which would provide opportunities to explore self-optimizing algorithms.

5.2.5 Multiagent System Design Methodology

Finally, the work described in this dissertation opens up a new avenue of research regarding Agent-Oriented Software Engineering (AOSE). It demonstrates that it is possible to take a BPEL4WS file that was created in graphical workflow design tool, and use it to instantiate a MAS. It should be explored whether a more general MAS design methodology and toolset can be formalized from the Gaia Agent-Oriented Analysis and Design methodology [82], graphical workflow design tools which emit BPEL4WS, and the distributed workflow enactment mechanism described in Chapter 4. These pieces should natural fit together because a workflow essentially represents the sociality of the business process; that is, the relationships between the workflow participants, the necessary conversations they have while processing the work, and the work product itself.

As demonstrated, many challenging and interesting research paths can be chosen from the groundwork laid in this dissertation. Looking forward, it will be exciting to see how the combination of agent-based and service-oriented computing revolutionizes software construction practice over the next decade.

References

- [1] M. Sawhney and J. Zabin, *The Seven Steps to Nirvana : Strategic Insights into eBusiness Transformation*. New York: McGraw-Hill, 2001.
- [2] Sun Microsystems, The Net Effect,
<http://www.sun.com/neteffect/whitepaper.html>
- [3] B. Boehm and V. Basili, "Gaining Intellectual Control of Software Development," in *IEEE Computer*, vol. 33, 2000, pp. 27-33.
- [4] R. Malveau and T. J. Mowbray, *Software Architect Bootcamp*, 2nd ed: Prentice Hall PTR, 2004.
- [5] Business Integration Journal Online, Universal Business Integration: An Idea Whose Time Has Come, <http://www.bijonline.com/PDF/matz%20march.pdf>
- [6] S. Redwine and W. Riddle, "Software Technology Maturation," presented at the 8th International Conference on Software Engineering, 1985.
- [7] M. Shaw, "The coming-of-age of software architecture research," presented at Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, 2001.
- [8] M. N. Huhns, "Interaction-Oriented Software Development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, pp. 259-279, 2001.
- [9] M. Shaw, "What makes good research in software engineering?," *International Journal of Software Tools for Technology Transfer*, vol. 4, pp. 1-7, 2002.
- [10] D. Garland, "Software Architecture: a Roadmap," presented at The Future of Software Engineering, Limerick, Ireland, 2000.
- [11] J.-G. Schneider, M. Lumpe, and O. Nierstrasz, "Agent Coordination via Scripting Languages," in *Coordination of Internet Agents : Models, Technologies, and Applications*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. New York, NY: Springer-Verlag, 2001, pp. 153-175.
- [12] D. Gelernter and N. Carriero, "Coordination Languages and their Significance," in *Communications of the ACM*, vol. 35, 1992, pp. 97-107.
- [13] F. DeRemer and H. Kron, "Programming in the Large versus Programming in the Small," *IEEE Transactions on Software Engineering*, vol. 2, pp. 80-87, 1976.
- [14] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Upper Saddle River, New Jersey: Prentice Hall PTR, 2000.
- [15] O. E. Williamson, S. G. Winter, and R. H. Coase, *The Nature of the firm : origins, evolution, and development*. New York: Oxford University Press, 1991.
- [16] H. V. D. Paranak, "'Go to the Ant': Engineering Principles from Natural Multi-Agent Systems," *Annals of Operations Research*, 1997.
- [17] S. L. Pfleeger, *Software engineering : theory and practice*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2001.
- [18] eAI Journal, Business Process Logic: Half-Empty or Half-Full?,
<http://www.eaijournal.com/Article.asp?ArticleID=629&DepartmentID=7>
- [19] Sun Microsystems, Inc., Java 2 Platform, Enterprise Edition,
<http://java.sun.com/j2ee/>
- [20] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. New York: Addison-Wesley, 2002.

- [21] G. T. Heineman and W. T. Councill, "Definition of a Software Component and Its Elements," in *Component-Based Software Engineering : Putting the Pieces Together*, G. T. Heineman and W. T. Councill, Eds. Boston: Addison-Wesley, 2001, pp. 5-19.
- [22] P. Herzum and O. Sims, *Business Component Factory : A Comprehensive Overview of Component-Based Development for the Enterprise*. New York: John Wiley, 2000.
- [23] J. Sametinger, *Software Engineering with Reusable Components*. New York: Springer-Verlag, 1997.
- [24] B. C. Meyers and P. Oberndorf, *Managing software acquisition : open systems and COTS products*. Boston: Addison-Wesley, 2001.
- [25] R. Weinreich and J. Sametinger, "Component Models and Component Services: Concepts and Principles," in *Component-Based Software Engineering: Putting the Pieces Together*, G. T. Heineman and W. T. Councill, Eds. New York: Addison-Wesley, 2001, pp. 33-48.
- [26] G. Glass, *Web Services, Building Blocks for Distributed Systems*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.
- [27] P. Wegner, "Interoperability," *ACM Computing Surveys*, vol. 28, pp. 285-287, 1996.
- [28] The Workflow Management Coalition, Terminology & Glossary, Document Number WfMC-TC-1011, http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf
- [29] WebServices.Org, The 'big boys' unite forces - What does it mean for you?, <http://www.webservices.org/index.php/article/articleview/633/1/24/>
- [30] W. v. d. Aalst and K. M. v. Hee, *Workflow management : models, methods, and systems*. Cambridge, Mass.: MIT Press, 2002.
- [31] The Workflow Management Coalition, The Workflow Reference Model, Document Number TC00-1003, <http://www.wfmc.org/standards/docs/tc003v11.pdf>
- [32] M. P. Singh and M. N. Huhns, "Multiagent Systems for Workflow," *International Journal of Intelligent Systems in Accounting, Finance and Management*, vol. 8, pp. 105-117, 1999.
- [33] XML Cover Pages, Business Process Execution Language for Web Services (BPEL4WS), <http://xml.coverpages.org/bpel4ws.html>
- [34] W. v. d. Aalst, "Don't go with the flow: Web services composition standards exposed," in *IEEE Intelliegent Systems*, vol. 18, 2003.
- [35] The DAML Services Coalition, "DAML-S: Web Service Description for the Semantic Web," presented at The First International Semantic Web Conference (ISWC), 2002.
- [36] S. A. McIlraith, T. C. Son, and H. Zeng, "Mobilizing the Semantic Web with DAML-Enabled Web Services," presented at Semantic Web Workshop, Hongkong, China, 2001.
- [37] The DAML Services Coalition, DAML-S and Related Technologies, <http://www.daml.org/services/daml-s/0.7/survey.pdf>
- [38] C. Shirky, "Web Services and Context Horizons," in *IEEE Computer*, vol. 35, 2002, pp. 98-100.

- [39] P. A. Buhler and J. M. Vidal, "Towards the Synthesis of Web Services and Agent Behaviors," presented at Proceedings of the Agentcities: Challenges in Open Agent Environments Workshop, Bologna, 2002.
- [40] M. N. Huhns, "Agents as Web Services," in *Internet Computing*, vol. 6, 2002, pp. 93-95.
- [41] M. Griss, "Software Agents as Next Generation Software Components," in *Component-based software engineering: putting the pieces together*, G. T. Heineman and W. T. Councill, Eds. Boston: Addison-Wesley, 2001, pp. 641-657.
- [42] Z. Maamar and J. Sutherland, "Toward Intelligent Business Objects," in *Communications of the ACM*, vol. 43, 2000, pp. 99-101.
- [43] D. C. Marinescu, *Internet-based workflow management : toward a semantic web*. New York: Wiley-Interscience, 2002.
- [44] M. Wooldridge, "Agents and Software Engineering," *AI*IA Notizie*, vol. XI, pp. 31-37, 1998.
- [45] N. R. Jennings, "An Agent-Based Approach for Building Complex Software Systems," in *Communications of the ACM*, vol. 44, 2001, pp. 35-41.
- [46] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 177, pp. 277-296, 2000.
- [47] M. J. Wooldridge, *Reasoning about rational agents*. Cambridge, Mass.: MIT Press, 2000.
- [48] M. N. Huhns and L. M. Stephens, "Multiagent Systems and Societies of Agents," in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999, pp. 79-120.
- [49] Petri Nets World, <http://www.daimi.au.dk/PetriNets/>
- [50] The Foundation for Intelligent Physical Agents, www.fipa.org
- [51] IBM, Autonomic Computing: IBM's Perspective on the State of Information Technology, <http://www.research.ibm.com/autonomic/manifesto/>
- [52] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," in *IEEE Computer*, vol. 36, 2003, pp. 41-50.
- [53] S. Cowley, "BPM market primed for growth," in *InfoWorld*, September 23 ed, September 23, 2002.
- [54] IBM, BPWS4J, <http://www.alphaworks.ibm.com/tech/bpws4j>
- [55] WfMC, "Press Release," September 12, 2002.
- [56] BPML.org, "BPML|BPEL4WS: A Convergence Path toward a Standard BPM Stack," August 15, 2002.
- [57] J. Korhonen, L. Pajunen, and J. Puustijarvi, "Using Web Services and Workflow Ontology in Multi-Agent Systems," presented at Workshop on Ontologies for Multi-Agent Systems, Siguenza, Spain, 2002.
- [58] G. Anthes, "Agents of Change," in *Computerworld*, January 27, 2003, pp. 26-27.
- [59] P. Buhler, J. M. Vidal, and H. Verhagen, "Adaptive workflow = web services + agents," presented at Proceedings of the First International Conference on Web Services, Las Vegas, Nevada, 2003.
- [60] P. Buhler and J. M. Vidal, "Towards adaptive workflow enactment using multiagent systems," *Information Technology and Management Journal: Special Issue on Universal Enterprise Integration*, vol. 6, pp. 61-87, 2005.

- [61] J. M. Vidal, P. Buhler, and C. Stahl, "Multiagent Systems with Workflows," in *Internet Computing*, vol. 8, 2004, pp. 76-82.
- [62] R. J. A. Buhr and R. S. Casselman, *Use case maps for object-oriented systems*: Prentice Hall, 1996.
- [63] Whitestein Information Technology Group AG, Web services Agent Integration Project, <http://wsai.sourceforge.net/index.html>
- [64] Telecom Italia Lab, JADE (Java Agent DEvelopment Framework), <http://sharon.cselt.it/projects/jade/>
- [65] G. Hohpe and B. Woolf, *Enterprise integration patterns : designing, building, and deploying messaging solutions*. Boston: Addison-Wesley, 2003.
- [66] Agentcities Web Services Working Group, Integrating Web Services into Agentcities Technical Recommendation, <http://www.agentcities.org/rec/00006/>
- [67] L. Brownsword, T. Oberndorf, and C. A. Sledge, "Developing New Processes for COTS-Based Systems," in *IEEE Software*, vol. 17, 2000, pp. 48-55.
- [68] D. J. Mandell and S. A. McIlraith, "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation," presented at Proceedings of the Second International Semantic Web Conference, 2003.
- [69] The OWL Services Coalition, OWL-S: Semantic Markup for Web Services, <http://www.daml.org/services/owl-s/1.0/owl-s.pdf>
- [70] F. Curbera and R. Khalaf, "Implementing BPEL4WS: The Architecture of a BPEL4WS Implementation," presented at Proceedings of the Grid Workflow Workshop at GGF-10, Berlin, Germany, 2004.
- [71] webMethods, Inc., Glue Overview, http://www.webmethods.com/solutions/wM_Glue/
- [72] The Apache XML Project, Xindice Homepage, <http://xml.apache.org/xindice>
- [73] S. A. DeLoach, "Analysis and Design of Multiagent Systems Using Hybrid Coordination Media," presented at Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, 2002.
- [74] World Wide Web Consortium, XML Path Language (XPath) Version 1.0, <http://www.w3.org/TR/xpath>
- [75] W. v. d. Aalst, "The Application of Petri Nets to Workflow Managment," *Journal of Circuits, Systems, and Computers*, vol. 8, pp. 21-66, 1998.
- [76] M. Klusch and K. Sycara, "Brokering and Matchmaking for Coordination of Agent Societies: A Survey," in *Coordination of Internet agents : models, technologies, and applications*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. Berlin ; New York: Springer, 2001, pp. 197-224.
- [77] Apache <Web Services /> Project, Introduction to WSIF, <http://ws.apache.org/wsif/>
- [78] Sun Microsystems, Inc., Java Web Services Developer Pack, <http://java.sun.com/webservices/webservicespack.html>
- [79] R. Monson-Haefel, *J2EE Web services*. Boston: Addison-Wesley, 2004.
- [80] M. Nadelson, "Stay Flexible with Logic Scripts," in *JavaPro*, vol. 7, 2003.
- [81] Queensland University of Technology, YAWL: Yet Another Workflow Language, <http://www.citi.qut.edu.au/yawl/index.jsp>

- [82] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 285-312, 2000.